Problem

This exercise concerns TM M_2, whose description and state diagram appear in Example 3.7. In each of the parts, give the sequence of configurations that M_2 enters when started on the indicated input string.

a. 0.

A**b.** 00.

c. 000.

d. 000000.

EXAMPLE 3.7

Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

M_2 = "On input string w:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1."

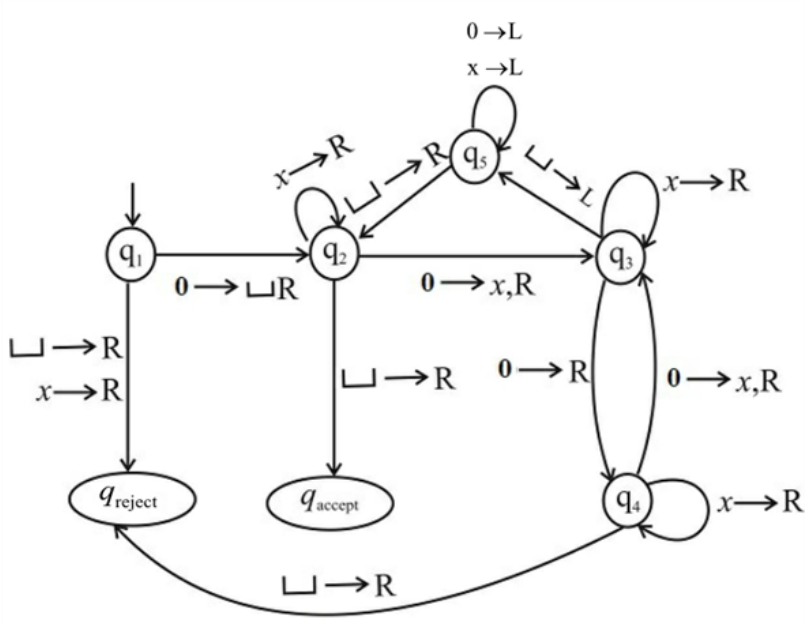
Step-by-step solution

Step 1 of 7

Consider the language $A = \{0^{2^n} \mid n \geq 0\}$, consisting of all strings of $0s$ whose length is a power of 2.

Turing machine M_2 decides a language A.

The state diagram for M_2 is as follows.



Comment

Step 2 of 7

In this state diagram the label $0 \to x, R$, appears on the transition from q_4 to q_3. This label signifies that, the state q_4 with head reading 0, the machine goes to state q_3, writes *x*, and moves the head to the right. In the similar manner, other transitions also occur.

Comment

Step 3 of 7

a. 0

Run the machine M_2 on the input 0. The starting configuration is $q_1 0$.The sequence of configurations that the machine enters when started on the input string is as follows:

$$\begin{aligned} &q_1 0 && (\text{At } q_1, 0 \to \sqcup, \text{R goes to the } q_2) \\ &\sqcup q_2 \sqcup && (\text{At } q_2, \sqcup \to \text{R goes to the accept state}) \\ &\sqcup \sqcup q_{accept} \end{aligned}$$

The state q_1 on 0, the machine goes to state q_2, writes $\sqcup$ and moves the head to the right. The state q_2 on $\sqcup$, the machine goes to state q_{accept}, then halts.

As M_2 enters q_{accept} state the input 0 is accepted.

Comment

Step 4 of 7

b. 00

Run the machine M_2 on the input 00. The starting configuration is $q_1 00$.The sequence of configurations that the machine enters when started on the input string is as follows:

$$\begin{aligned} &q_1 00 && (\text{At } q_1, 0 \to \sqcup, \text{R goes to the } q_2) \\ &\sqcup q_2 0 && (\text{At } q_2, 0 \to \text{ x, R goes to the } q_3) \\ &\sqcup x q_3 \sqcup && (\text{At } q_3, \sqcup \to \text{L goes to the } q_5) \\ &\sqcup q_5 x \sqcup && (\text{At } q_5, x \to \text{L goes to the } q_5) \\ &q_5 \sqcup x \sqcup && (\text{At } q_5, \sqcup \to \text{R goes to the } q_2) \\ &\sqcup q_2 x \sqcup && (\text{At } q_2, x \to \text{R goes to the accept state}) \\ &\sqcup x q_2 \sqcup && (\text{At } q_2, \sqcup \to \text{R goes to the accept state}) \\ &\sqcup x \sqcup q_{accept} \end{aligned}$$

The state q_1 on 0, the machine goes to state q_2, writes $\sqcup$ and moves the head to the right. The state q_2 on 0, the machine goes to state q_3, writes x and moves the head to the right. The state q_3 on $\sqcup$, the machine goes to state q_5, moves the head to the left. The state q_5 on x, the machine goes to state q_5, moves the head to the left. The state q_5 on $\sqcup$, the machine goes to state q_2, moves the head to the right. The state q_2 on x, the machine goes to state q_2 itself, moves the head to the right. The state q_2 on $\sqcup$, the machine goes to state q_{accept}, then halts.

Finally, M_2 enters q_{accept} state. Thus, the input 00 is accepted.

Comments (2)

Step 5 of 7

c. 000

Run the machine M_2 on the input 000. The starting configuration is q_1000.The sequence of configurations that the machine enters when started on the input string is as follows:

Comment

Step 6 of 7

$$
\begin{aligned}
&q_1000 && (\text{At } q_1, 0 \to \sqcup, \text{R goes to the } q_2)\\
&\sqcup q_200 && (\text{At } q_2, 0 \to \text{x, R goes to the } q_3)\\
&\sqcup x q_3 0 && (\text{At } q_3, 0 \to \text{R goes to the } q_4)\\
&\sqcup x 0 q_4 \sqcup && (\text{At } q_4, \sqcup \to \text{R goes to the } q_{\text{reject}})\\
&\sqcup x 0 \sqcup q_{\text{reject}}
\end{aligned}
$$

The state q_1 on 0, the machine goes to state q_2, writes $\sqcup$ and moves the head to the right. The state q_2 on 0, the machine goes to state q_3, writes x and moves the head to the right. The state q_3 on 0, the machine goes to state q_4, moves the head to the right. The state q_4 on $\sqcup$, the machine goes to state q_{reject}, moves the head to the right.

Finally, M_2 enters q_{reject} state. Thus, input 000 is rejected.

Comment

Step 7 of 7

d. 000000

Run the machine M_2 on the input 000000. The starting configuration is $q_1000000$.The sequence of configurations that the machine enters when started on the input string is as follows:

$q_1 000000$ (At q_1, 0 → ␣, R goes to the q_2)

␣$q_2$00000 (At q_2, 0 → x, R goes to the q_3)

␣x$q_3$0000 (At q_3, 0 → R goes to the q_4)

␣x0$q_4$000 (At q_4, 0 → x, R goes to the q_3)

␣x0x$q_3$00 (At q_3, 0 → R goes to the q_4)

␣x0x0$q_4$0 (At q_4, 0 → x, R goes to the q_3)

␣x0x0xq_3␣ (At q_3, ␣ → L goes to the q_5)

␣x0x0q_5x␣ (At q_5, x → L goes to the q_5)

␣x0x$q_5$0x␣ (At q_5, 0 → L goes to the q_5)

␣x0q_5x0x␣ (At q_5, x → L goes to the q_5)

␣x$q_5$0x0x␣ (At q_5, 0 → L goes to the q_5)

␣q_5x0x0x␣ (At q_5, x → L goes to the q_5)

q_5␣x0x0x␣ (At q_5, ␣ → R goes to the q_2)

␣q_2x0x0x␣ (At q_2, x → R goes to the q_2)

␣x$q_2$0x0x␣ (At q_2, 0 → x, R goes to the q_3)

␣xxq_3x0x␣ (At q_3, x → R goes to the q_3)

␣xxx$q_3$0x␣ (At q_3, 0 → R goes to the q_4)

␣xxx0q_4x␣ (At q_4, x → R goes to the q_4)

␣xxx0xq_4␣ (At q_4, ␣ → R goes to the q_{reject})

␣xxx0x␣q_{reject}

Finally, M_2 enters q_{reject} state. Hence, the input 000000 is rejected.

Comment

Problem

This exercise concerns TM M_1, whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

A**a.** 11.

b. 1#1.

c. 1##1.

d. 10#11.

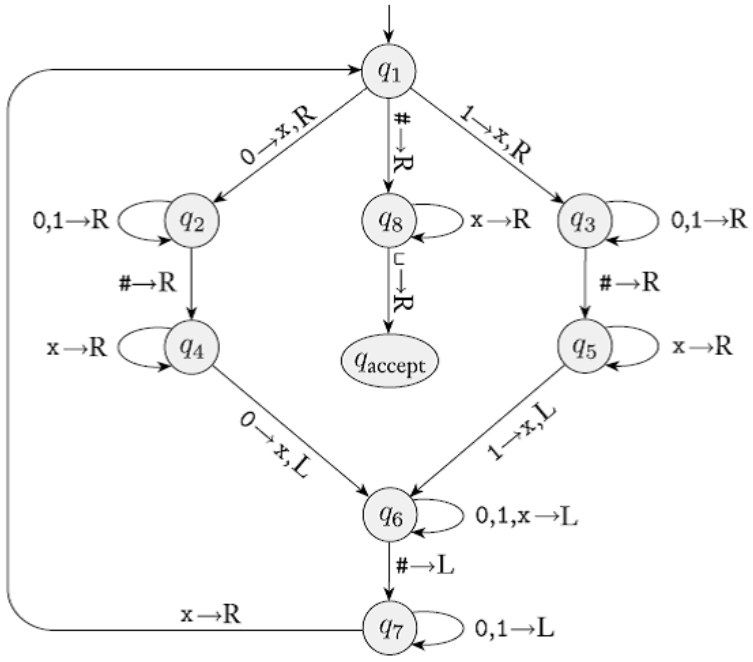
e. 10#10.

Example 3.9

EXAMPLE 3.9

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described (page 167) for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.



Step-by-step solution

Step 1 of 6

Consider the language $B = \{w\#w \mid w \in \{0,1\}^*\}$. The Turing machine M_1 that decides the language B.

In the state diagram, reject state has not shown for simplicity. If the state does not have an outgoing transition for any symbol, then it moves to the reject state q_{reject}.

In the state diagram, the label $0 \to x, R$ appears on the transition from q_1 to q_2. This label signifies that, the state q_1 with head reading 0, the machine goes to state q_2, writes *x*, and moves the head to the right.

Comment

Step 2 of 6

a.

Consider the input string 11. The sequence of configurations that M_1 enters are as follows:

$$\begin{aligned}
&\to q_1 11 \\
&\to \mathrm{x} q_3 1 \\
&\to \mathrm{x} 1 q_3 \sqcup \quad [\because\ q_3 \text{ is not reading} \sqcup, \text{ so it enters to reject state}] \\
&\to \mathrm{x} 1 \sqcup q_{\text{reject}}
\end{aligned}$$

Finally, M_1 enters the q_{reject} state. Hence input 11 is rejected.

Comment

Step 3 of 6

b.

Consider the input string 1#1. The sequence of configurations that M_1 enters are as follows:

$$\begin{aligned}
&\to q_1 1\#1 \\
&\to \mathrm{x} q_3 \#1 \\
&\to \mathrm{x} \# q_5 1 \\
&\to \mathrm{x} q_6 \# \mathrm{x} \\
&\to q_7 \mathrm{x} \# \mathrm{x} \\
&\to \mathrm{x} q_1 \# \mathrm{x} \\
&\to \mathrm{x} \# q_8 \mathrm{x} \\
&\to \mathrm{x} \# \mathrm{x} q_8 \sqcup \\
&\to \mathrm{x} \# \mathrm{x} \sqcup q_{\text{accept}}
\end{aligned}$$

Finally, M_1 enters q_{accept} state. Thus, the input 1#1 is accepted.

Comment

Step 4 of 6

c.

Consider the input string 1##1. The sequence of configurations that M_1 enters are as follows:

$$\begin{aligned}
&\to q_1 1\#\#1 \\
&\to \mathrm{x} q_3 \#\#1 \\
&\to \mathrm{x} \# q_5 \#1 \quad [\because\ \mathrm{q}_5 \text{ is not reading \#, so it enters to reject state}] \\
&\to \mathrm{x} \#\# q_{\text{reject}} 1
\end{aligned}$$

Finally, M_1 enters q_{reject} state. Thus, the input 1##1 is rejected.

Comment

Step 5 of 6

d.

Consider the input string 10#11. The sequence of configurations that M_1 enters are as follows:

$$\begin{aligned}
&\rightarrow q_1 10\#11\\
&\rightarrow \mathrm{x}q_3 0\#11\\
&\rightarrow \mathrm{x}0q_3 \#11\\
&\rightarrow \mathrm{x}0\#q_5 11\\
&\rightarrow \mathrm{x}0q_6 \#\mathrm{x}1\\
&\rightarrow \mathrm{x}q_7 0\#\mathrm{x}1\\
&\rightarrow q_7 \mathrm{x}0\#\mathrm{x}1\\
&\rightarrow \mathrm{x}q_1 0\#\mathrm{x}1\\
&\rightarrow \mathrm{xx}q_2 \#\mathrm{x}1\\
&\rightarrow \mathrm{xx}\#q_4 \mathrm{x}1\\
&\rightarrow \mathrm{xx}\#\mathrm{x}q_4 1\ [\because q_4 \text{is not reading 1, so it enters to reject state}]\\
&\rightarrow \mathrm{xx}\#\mathrm{x}1q_{\text{reject}}
\end{aligned}$$

Finally, M_1 enters q_{reject} state. Thus, the input 10#11 is rejected.

Comment

Step 6 of 6

e.

Consider the input string 10#10. The sequence of configurations that M_1 enters are as follows:

$\rightarrow q_1 10\#10$

$\rightarrow \mathrm{x}q_3 0\#10$

$\rightarrow \mathrm{x}0q_3 \#10$

$\rightarrow \mathrm{x}0\#q_5 10$

$\rightarrow \mathrm{x}0q_6 \#\mathrm{x}0$

$\rightarrow \mathrm{x}q_7 0\#\mathrm{x}0$

$\rightarrow q_7 \mathrm{x}0\#\mathrm{x}0$

$\rightarrow \mathrm{x}q_1 0\#\mathrm{x}0$

$\rightarrow \mathrm{xx}q_2 \#\mathrm{x}0$

$\rightarrow \mathrm{xx}\#q_4 \mathrm{x}0$

$\rightarrow \mathrm{xx}\#\mathrm{x}q_4 0$

$\rightarrow \mathrm{xx}\#q_6 \mathrm{xx}$

$\rightarrow \mathrm{xx}q_6 \#\mathrm{xx}$

$\rightarrow \mathrm{x}q_7 \mathrm{x}\#\mathrm{xx}$

$\rightarrow \mathrm{xx}q_1 \#\mathrm{xx}$

$\rightarrow \mathrm{xx}\#q_8 \mathrm{xx}$

$\rightarrow \mathrm{xx}\#\mathrm{x}q_8 \mathrm{x}$

$\rightarrow \mathrm{xx}\#\mathrm{xx}q_8 \sqcup$

$\rightarrow \mathrm{xx}\#\mathrm{xx}\sqcup q_{\text{accept}}$

Finally, M_1 enters q_{accept} state. Thus, the input 10#10 is accepted.

Comment

Problem

Modify the proof of Theorem 3.16 to obtain Corollary 3.19, showing that a language is decidable iff some nondeterministic Turing machine decides it. (You may assume the following theorem about trees. If every node in a tree has finitely many children and every branch of the tree has finitely many nodes, the tree itself has finitely many nodes.)

THEOREM **3.16**

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

COROLLARY **3.19**

A language is decidable if and only if some nondeterministic Turing machine decides it.

Step-by-step solution

Step 1 of 4

As per the theorem 3.16, Every DTM (Deterministic Turing machine) has an equivalent NDTM (Non-deterministic Turing machine). The corollary 3.19, states that a language is decidable iff some NDTM (Non-deterministic Turing machine) decides it.

In order to modify the proof of theorem 3.16 to obtain corollary 3.19, prove the following:

(i) If a Language *L* is decidable then the language *L* can be decided by NDTM (non-deterministic Turing machine).

Let *L* be a language. The Language *L* is decidable if it is decided by a deterministic Turing machine. Any DTM (Deterministic Turing machine) is automatically a NDTM (Non-deterministic Turing machine).

Thus, language L is decidable if it is decided by NDTM (Non-deterministic Turing machine) TM also.

Comment

Step 2 of 4

(ii) If some NDTM (Non-deterministic Turing machine) decides a language then it is decidable.

Let *L* be an any language and it is decided by NDTM (Non-deterministic Turing machine).

Construct a deterministic TM *D* that decides *L* and prove that *D* simulates *N*.

Construction of *D* and Simulation of *D* with *N*:

The simulating deterministic TM '*D*' has 3 tapes

1. Input tape

2. Simulating tape

3. Address tape

By the theorem

"Every multi tape TM has an equivalent single tape TM"

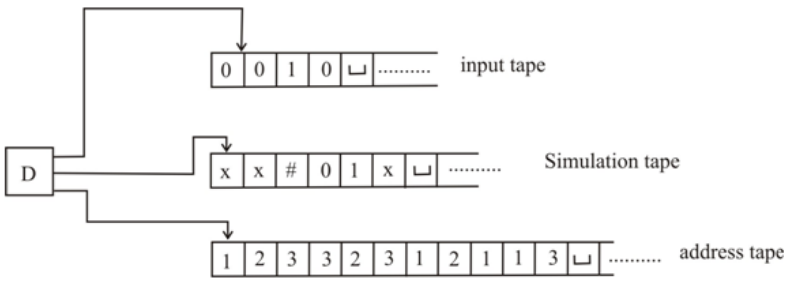
D's arrangement is equivalent to having a single tape TM.

The machine *D* uses its three tapes in the following way:

• Tape 1 always contains the input string and is never altered.

• Tape 2 maintains a copy of *N*'s tape on some branch of its non-deterministic computation

• Tape 3 keeps track of *D*'s location in N's non-deterministic computation tree.



Comment

Step 3 of 4

The description of D in detail as follows:

Stage 1: Initially tape 1 contains the input w, and tapes 2 and 3 are empty.

Stage 2: Copy tape 1 to tape 2.

Stage 3: Use tape 2 to simulate N with input w on one branch of its non-deterministic computation. Before each step of N check the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function. If no more symbols left on tape 3 or if this non-deterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, then accept the input.

Stage 4: Replace the string on tape 3 with the lexicographically next string. Simulate the next branch of N's computation by going to stage 2.

Stage 5: Reject if all branches of non-determinism of N are exhausted.

Comment

Step 4 of 4

Now it can be said that D is a decider of L.

• If N accepts its input, then D will find an accepting branch and accept.

• If N rejects its input, then all its branches halt and reject.

Thus, the language L is decidable.

From (i) and (ii),

The language L is decidable if and only if some non-deterministic Turing machine decides it.

Comment

Problem

Give a formal definition of an enumerator. Consider it to be a type of two-tape Turing machine that uses its second tape as the printer. Include a definition of the enumerated language.

Step-by-step solution

Step 1 of 3

An enumerator is Turing machine(TM) variant with as attached printer. The enumerator uses the printer as output device to print strings. Every time Turing machine wants to add a string to the list, it sends the string to the printer.

Formal definition of an enumerator:

An Enumerator is a 7-tuple $\langle Q,\Sigma,\Gamma,\delta,q_0,q_{print},q_{halt}\rangle$

Where Q,Σ,Γ are all finite sets and

(i) Q is the set of states

(ii) Γ is the work tape alphabet,

(iii) Σ is the output/print tape alphabet

(iv) $\delta: Q\times\Gamma\to Q\times\Gamma\times\{L,R\}\times\Sigma_{\in}$ is the transition function,

(v) $q_0\in Q$ is the start state

(vi) $q_{print}\in Q$ is the print state, and

(vii) $q_{halt}\in Q$ is the halt state, where $q_{print}\neq q_{halt}$

Comment

Step 2 of 3

Computation of an enumerator:

The computation of an enumerator E is defined as an ordinary TM, except for following three points.

1. It has two tapes,

• First tape is work tape and

• Second tape is print tape

Both are initially blank.

2. At each step, the machine may write a symbol from Σ on the output tape, or nothing, as determined by δ.

For example,

If $\delta(q_{prev},a)=(q_{next},b,L,c)$, it means that in state q_{prev}, reading a, enumerator E enters state q_{next}, writes *b* on the work tape moves the work tape head left and writes *c* on the print tape, and moves the print tape head to the right if $c\neq\in$.

3. Whenever state q_{print} is entered, the output tape is reset to blank and the head returns to the left hand end and the output is printed.

When q_{Halt} is entered the machine will halt.

Comment

Step 3 of 3

Enumerated language:

The language enumerated by Enumerator is the collection of all the strings that Enumerator eventually prints out.

Enumerator may generate the strings of the language in any order, possibly with repetitions.

L(E): be the language enumerated by enumerator E

Thus *L(E)* is the language that contains the strings which are printed out.

$$L(E) = \{w \in \Sigma^* \mid w \text{ appears on the work tape if } q_{print} \text{ is entered}\}$$

Comment

Problem

Examine the formal definition of a Turing machine to answer the following questions, and explain your reasoning.

a. Can a Turing machine ever write the blank symbol $\sqcup$ on its tape?

b. Can the tape alphabet Γ be the same as the input alphabet $\sum$?

c. Can a Turing machine's head *ever* be in the same location in two successive steps?

d. Can a Turing machine contain just a single state?

Step-by-step solution

Step 1 of 4

(a) **Yes.** A Turing machine can write the blank symbol $\sqcup$ on its tape.

According to the definition, a Turing machine is a 7 – tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$

Where Q,Σ,Γ are all finite sets.

Σ is the input alphabet not containing the blank symbol $\sqcup$.

But Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$.

A Turing machine can write any characters in Γ on its tape.

Thus Turing machine can write blank symbol on its tape.

Comment

Step 2 of 4

(b) **No.** The tape alphabet Γ never same as input alphabet Σ

Because tape alphabet Γ contains blank symbol $\sqcup$ where as input alphabet does not contain blank symbol.

Always Σ is subset of Γ, but never same as Σ.

Comment

Step 3 of 4

(c) **Yes.** Turing machines head be in the same location in two successive steps.

When the Turing machines head is at the left _end of the tape, if Turing machine again try to move left side then Turing machine's head is in the same location as previous can move.

Comment

Step 4 of 4

(d) **No.**

According to definition, A Turing machine is a 7- tuple ($(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$

Where Q is the set of states.

q_{accept} is the accept state and is belongs to Q

q_{reject} is the reject state and is belongs to Q.

But $q_{accept} \neq q_{reject}$

Thus there must be two distinct states q_{accept} and q_{reject}

So, a Turing machine contains at least two states.

Comment

Problem

In Theorem 3.21, we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didn't we use the following simpler algorithm for the forward direction of the proof? As before, $s_1, s_2, \ldots$ is a list of all strings in Σ^*.

E = "Ignore the input.

1. Repeat the following for i = 1, 2, 3, ...

2. RunM on s_i.

3. If it accepts, print out s_i."

THEOREM **3.21**

A language is Turing-recognizable if and only if some enumerator enumerates it.

Step-by-step solution

Step 1 of 1

Theorem: - A language is Turing – recognizable if and only if some enumerator enumerates it.

The given simpler algorithm for the forward direction of the proof of this theorem is

Say that $s_1, s_2 \cdots$ is a list of all strings in Σ^*

If Turing Machine M recognizes a language L, then we can construct following enumerator E for L.

The Enumerator E works as follows:

$E = "\text{Ignore the input}$

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

2. Run M on S_i;

3. If it accepts, print out S_i"

Defects in this proof:

In stage 2 of this algorithm (Run M on S_i)

If M loops on a certain input S_i runs forever, E could not check any input after S_i.

If it occurs, then E might fail to enumerate its language L as required.

Thus this procedure does not give the effect of running M in parallel on all possible input strings.

So, this proof is not suited for forward direction of above theorem.

Comment

Problem

Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "On input $\langle p \rangle$, a polynomial over variables $x_1, \ldots, x_k$:

1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.

2. Evaluate p on all of these settings.

3. If any of these settings evaluates to 0, *accept* ; otherwise, *reject* ."

Step-by-step solution

Step 1 of 2

Consider the steps for the legitimate turing machine:

M_{bad} =" The input is a polynomial P over variables $x_1,\ldots,x_k$

Step 1: Try all possible values of $x_1,\ldots,x_k$ to integer values.

Step 2: Evaluate P on these values

Step 3: If any of these settings evaluates to 0, accept; otherwise, reject"

• From the step 1 to store all the values turing machine require infinite memory location. So, this is not possible to accept the machine.

• For the Step 2, infinite processing time to require all the values for evaluation. So, this step also not possible to accept the machine.

• If the above two steps are not possible, then the step 3 is rejected.

Comment

Step 2 of 2

So, the Turing machine M_{bad} could require **infinite time and infinite steps** to try all of them.

But the above settings are required that every stage in the Turing machine description be completed in a finite number of steps.

Thus, this description is not suitable for the legitimate Turing machine.

Comment

Problem

Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

A**a.** {wl w contains an equal number of 0s and 1s}

b. {wl w contains twice as many 0s as 1s}

c. {wl w does not contain twice as many 0s as 1s}

Step-by-step solution

Step 1 of 3

Turing decidable: If some Turing machine decides the language then it is said to be Turing decidable.

a.

Given language L_1 is { $w \mid w$ contains an equal number of 0's and 1's} over the alphabet $\{0,1\}$.

Let M_1 be the Turing machine that decides the language L. The implementation level description of M_1 is as follows:

M_1 ="on input string w:

Step 1:

Mark the first unmarked 0 by scanning the tape and mark it. If there are no unmarked 0, go to step 4. Else, move the head back to the front of the tape.

Step 2:

Mark the first unmarked 1 by scanning the tape and mark it. If there is no unmarked 1, then reject.

Step 3:

Place the head back to the front of the tape and repeat step 1.

Step 4:

Place the head back to the front of the tape and check if any unmarked 0's or 1's remain by scanning the tape. If there are none, accept else it is unacceptable.

Comments (1)

Step 2 of 3

b.

Given language L_2 is { $w \mid w$ Contains twice as many 0's as 1's} over the alphabet $\{0,1\}$.

Let M_2 be the machine that decides the language L. The implementation level description of M_2 is as follows:

M_2 ="on input w:

Step 1:

Scan the tape for the first unmarked 1, mark it. If there are no unmarked 1's, and go to step 5. Else place the head at the start of the tape.

Step 2:

Scan the unmarked 0 is found in the tape and mark it. If there are no 0's, reject.

Step 3:

Scan the tape again till the unmarked 0's is said to be found and mark them. If there is no unmarked 0's, reject.

Step 4:

Move the head back to the front of the tape and repeat step 1.

Step 5:

Place the head back to the front of the tape scan the tape to see if there are any unmarked 0's are found. If none are found, then accept. Else, reject.

Comments (1)

c.

Given language L_3 is { $w \mid w$ does not contain twice as many 0's as 1's} over the alphabet $\{0,1\}$.

Let M_3 be the machine that decides the language L. The implementation level description of M_3 is as follows.

M_3 = " on input w:

Step 1:

Scan the tape and mark the first unmarked 1, mark it. If there is no unmarked 1's, go to stage 5. Else, place the head at the start of the tape.

Step 2:

Scan the tape till the unmarked 0's are found and mark it. If there are no 0's accept.

Step 3:

Scan the tape once again and marks the unmarked 0's are found. If there are no 0's, accept.

Step 4:

Place the head back to the front of the tape and repeat step 1.

Step 5:

Place the head back to the front of the tape. Scan the tape to see if there are unmarked 0's. If there are none, reject. Else, accept.

Comments (3)

Problem

Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.

a. Show that 2-PDAs are more powerful than 1-PDAs.

b. Show that 3-PDAs are not more powerful than 2-PDAs.

(Hint: Simulate a Turing machine tape with two stacks.)

Step-by-step solution

Step 1 of 3

Push down automata:

Push down automata (PDA) is like nondeterministic finite automata but have an extra component called stack. Stack provides additional memory beyond the finite amount available in the control.

• *k*-PDA is a push down automaton that has $k-$ stacks

• $0-\text{PDA}$ is a NFA

• $1-\text{PDA}$ is a conventional PDA.

It is known that $1-\text{PDAs}$ are more powerful than $0-\text{PDAs}$.

It means $1-\text{PDAs}$ are recognized by a larger class of languages than $0-\text{PDAs}$.

Comment

Step 2 of 3

(a)

Now it is required to show that $2-\text{PDAs}$ are more powerful than $1-\text{PDAs}$.

Prove this by taking an example.

As it is known that “A Language is context free if and only if some pushdown automaton recognizes it.”

Let $B=\left\{a^n b^n c^n \mid n \geq 0\right\}$ be the language.

P is the pumping length of the B given by the pumping lemma.

Now prove that B is not a CFL (Context-Free Language).

To show that B is not a CFL, it’s enough to show that a string $s=1^p 0^p 1^p 0^p$ cannot be pumped.

Consider *s* is of the form *s=uvxyz*.

• If both *v* and *y* contain at most one type of alphabet symbol, the string will be of the form uv^2xy^2z runs of $0\text{'s and }1\text{'s}$ of unequal length. Hence the string *s* cannot be a member of B.

• If either *v* or *y* contains more than one type of alphabet symbol, the string will be of the form uv^2xy^2z which does not contain the symbols in correct order. Hence the string *s* cannot be a member of B.

Since the string *s* cannot be pumped without violating the pumping lemma condition, *B* is not a CFL (context-free language). Thus $1-\text{PDA}$ does not recognize *B*.

But the following $2-\text{PDA}$ recognizes *B*.

• Push all $a's$ that appear in front of the input tape to stack 1.

• Then push all $b's$ that follow the $a's$ in the input stream into stack2. If it sees any $a's$ at this stage, ***reject.***

• When it sees the first *c*, pop stack 1 and stack2 at the same time. After this, reject the input if it sees any $a's$ (or) $b's$ in the input stream.

• Pop stack1 and stack2 for each input character *c* it reads. If the input ends when both stacks are empty, ***accept***. Otherwise ***reject.***

Therefore, B is recognized by $2-\text{PDA}$.

Thus $2\text{PDA}s$ are more powerful then $1-\text{PDA}s$.

Comments (2)

Step 3 of 3

(b)

To show that 3-PDAs are not more powerful than 2PDAs,

It is known that 3-PDA is not powerful than Turing machine, so it is required to show that the $2-\text{PDA}$ can simulate a Turing machine, then obviously 3-PDAs are not powerful than 2-PDAs.

Simulation of TM by 2PDA:

Now split the tape of TM (Turing machine) into two stacks.

• Stack1 stores the characters on the left of the head, with the bottom of stack storing the left most character of tape in the TM.

• Stack 2 stores the characters on the right of the head, with the bottom of the stack storing the right most character on the tape in the TM.

Show the Transition example for left and right.

Left Transition:

• If $\delta(q_i, w_i) = (q_j, w_j, L)$, then w_i on top of second stack.

• If $ is on top of stack then reject. Move left edge of tape.

o Otherwise pop w_i from top of second stack. Push w_j in second stack.

• Pop the symbol of first stack and push it to second stack. Switching the state of machine q_j.

• If q_j is accept state, then accepts *w*. If q_j is reject state, then rejects *w.*

Right Transition:

• If $\delta(q_i, w_i) = (q_j, w_j, R)$, then w_i on top of second stack.

• Pop w_i from second stack and push w_j in second stack

• If $ is on top of second stack, push null on to second stack. Either case, switching the state of machine q_j.

• If q_j is accept state, then accepts *w*. If q_j is reject state, then rejects *w.*

In this way $2-\text{PDA}$ simulates the Turing Machine.

Furthermore,

Now simulate 4-tape non-deterministic Turing machine with 3-PDA.

• First tape as input tape, second, third, and fourth tapes respectively first, second and third stack.

• If stack changed by push without pop then N moves the tape head to right without change the current symbol. Now write the symbol to the tape which is pushed on stack.

• If stack is changed by pop without push, then N replaces tape head with blank symbol and moves left.

• If stack is changed by push and pop, then N changes the tape by written symbol pushed in current cell.

• If stack is not changed, then N writes same symbol on tape head and then back to tape.

• Symbol is used from the tape then N moves first tape to right.

• If symbol is not used then N copies the element at first tape head.

• These actions can parallel in all tapes at same time since they take only one turn.

• 2nd point mentioned Push without pop, in this case all tape heads kept same except stacks affected by push without pop. Moves right to get the blank symbol at tape.

• N is nondeterministic to guess which transition to take same way PDA does.

So, it proves that Turing Machine are powerful as 3-PDAs, so it can be concluded that 3-PDAs are no more powerful than 2-PDAs. Both the PDAs are equally powerful.

Comments (1)

Problem

Say that a ***write-once Turing machine*** is a single-tape TM that can alter each tape square at most once (including the input portion of the tape). Show that this variant Turing machine model is equivalent to the ordinary Turing machine model. (Hint: As a first step, consider the case whereby the Turing machine may alter each tape square at most twice. Use lots of tape.)

Step-by-step solution

Step 1 of 2

To show that write – once Turing machine is equivalent to an ordinary Turing machine, first we simulate an ordinary Turing machine by a write – twice Turing machine.

Simulation of an ordinary Turing machine by a write – twice Turing machine:

Let the original machine be M. The two techniques by which it can be shown that write-once Turing machine is equivalent to an ordinary Turing machine.

The first technique is to copy the entire tape between transitions. In this one extra tape symbol is required as a delimiter and an expanded alphabet is required to record the position of the tape as it is copied.

In this case two writes are required: The first write is used while copying to record the symbol over fresh tape, the second write is used to mark a symbol as copied.

Comment

Step 2 of 2

The second technique is that the use of tape is not required. In this two tape squares are required to be used, instead of recording each symbol in one tape square. The first square is used to record a tape symbol and the second is used to mark the symbol as being copied. Each tape square can be changed only once.

Comment

Problem

A ***Turing machine with doubly infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

Step-by-step solution

Step 1 of 2

To prove the double infinite tape Turing machine *D* is equals to Normal ordinary Turing machine *M*.

In order to show the equivalence between *D* and *M*, it is necessary show the following two things:

• First one is, any language *L* that can be recognized by *M* can be recognized by *D* as well, and

• Second part is, any Language *L'* that can be recognized by *D* can be recognized by *M*.

This can be showed by simulation. That is, simulate *D* to act like *M* and vice-versa.

(i) **Simulating *M* by *D*:**

Step 1: Mark the left hand end of *D*.

Step 2: Prevent *D* from moving its head to the left of the mark.

In this way *D* simulates *M*.

Comment

Step 2 of 2

(ii) **Simulating *D* by *M*:**

To simulate the doubly infinite tape TM(Turing machine) *D* by an ordinary TM *M,* simulate it with a 2-tape TM.

As the 2 tape TM was already equivalent is power to an ordinary TM.

Simulation of doubly infinite tape TM ' *D* ' with 2-tape TM ' M_1 ':

The first tape of M_1 is written with input string and the second tape is blank.

Now cut the tape of doubly infinite tape Turing machine *D* into two parts, at the starting cell of the input string.

• One part containing input string and all the blank spaces to its right.

• Second part containing left of the input string appears on the second tape, in reverse order.

In this way *M* simulates *D*.

Thus from (i) and (ii)

D is equivalent to *M* i.e., Turing machine with a doubly infinite tape is equivalent to an ordinary Turing machine.

Comment

Problem

A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{RESET}\}.$$

If ? (q, a) = (r, b,RESET), when the machine is in state q reading an a, the machine's head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

Step-by-step solution

Step 1 of 2

To show the Turing machine with left reset, recognize the class of Turing recognizable class of language.

• It must be shown that it can simulate ordinary Turing machine.

• Let *M* be an ordinary Turing machine and M_L be the Turing machine with left reset.

• M_L simulates *M* in the following way.

• When *M* makes a right transition then M_L follows it in the same way as *M* do.

• When *M* makes a left transition with symbol *a*, *b* in *M*, M_L replaces it with *A* or *B* respectively. So, the alphabet set $\sum M_L = \sum MU\{A, B\}$ and does a left RESET.

• Shifts all content of the tape by one position to the right for all symbols other than $\{A, B\}$.

Comment

Step 2 of 2

The above process is repeated until all content of the tape are shifted to the right and does the following.

• M_L does a RESET again.

• All right transitions are checked.

• Whenever it reaches to some $\{A, B\}$, it works in the same way as *M* does.

Comment

Problem

A ***Turing machine with stay put instead of left*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{S}\}.$$

At each point, the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is *not* equivalent to the usual version. What class of languages do these machines recognize?

Step-by-step solution

Step 1 of 2

A Deterministic Finite State automaton can be simulated on the Turing Machine with stay put instead of left. The modifications can be done if transitions are added from state in F to q_{accept} and from the states outside F to q_{reject} when a blank symbol is read.

Assume there is a Turing Machine M, such that $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ with stay put instead of left. Create a DFA such that the DFA $(Q', \Sigma', \delta', q_0', F)$ recognizes the same language.

The machine M cannot move left and cannot write anything that it can written on the tape while moving to the right. Thus, the access is one-way.

For every DFA, there exists a Turing Machine that accepts the same language because a DFA is a Turing Machine with read only tape and tape head with moves to right.

Comments (2)

Step 2 of 2

The transition function δ' for the NFA is as follows:

First, set $\delta'(q_{\text{start}}, P) = \{q_{OP}\}$ where q_{op} is the start state of TM variant.

Next, set $\delta'(q_{accept}, i) = \{q_{\text{accept}}\}$ For any *i*

If $\delta(p,a) = (q_{\text{accept}}, b, w)$ where $w = R$ or S, set $\delta'(q_{pa}, \in) = \{q_{\text{acceept}}\}$

R is RIGHT *S* is stay put.

If $\delta(p,a) = (q_{reject}, b, w)$ where $w = R$ or S, we set $\delta'(q_{pa} \in) = \{q_{reject}\}$

• For each $a \in \Sigma$, set $\delta'(q_{start}, a) = \{\langle q_0, a\rangle\}$, where q_0 is start state of S.

• For each $p, q \in Q$ where $p \notin \{q_{accept}, q_{reject}\}$, for each $a \in \Gamma$, if S has transition of form $\delta(p,a) = (q_{\text{accept}}, b, w)$ or $\delta(p,a) = (q_{reject}, b, w)$, *w* becomes R for each $c \in \Sigma$, set $\delta'(\langle p, a\rangle, c) = \{\langle q, c\rangle\}$.

• For each $p, q \in Q$ where $p \notin \{q_{accept}, q_{reject}\}$, for each $a \in \Sigma$, if S has transition of form $\delta(p,a) = (q_{\text{accept}}, b, w)$ or $\delta(p,a) = (q_{reject}, b, w)$, *w* becomes S then set $\delta'(\langle p, a\rangle, \varepsilon) = \{\langle q, b\rangle\}$

Thus, an NFA is constructed which is defined as follows:

$(Q' = Q, \Sigma' = \Sigma, \delta', q_{op} = q_{start}, F)$ From our TM variant *S*.

The language recognized by NFA is regular languages.

Comment

Problem

A ***queue automaton*** is like a push-down automaton except that the stack is replaced by a queue. A ***queue*** is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we'll call it a *push*) adds a symbol to the left-hand end of the queue and each read operation (we'll call it a *pull*) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.

Step-by-step solution

Step 1 of 3

The equivalence between the queue automaton Q and the Turing machine M is required to be shown.

It means it is required to be shown that the language that can be recognized by the automaton, it can also be recognized by the Turing machine and vice versa.

This can be done by showing simulation which means that simulate the automaton Q to behave exactly like the Turing machine M and vice versa.

Comment

Step 2 of 3

The Automaton can be simulated by the Turing machine as follows:

Consider the entire tape as a queue. One by one each symbol is altered and the movement of the tape takes place to the right. If more than one numbers are to be pushed in the queue, then it is done by shifting contents to the right. If the end of the tape is reached, the leftmost symbol of the tape is approached.

Comment

Step 3 of 3

The Turing machine can be simulated by the automaton as follows:

The alphabet of the Turing machine M is expanded by inserting an extra symbol. A left end marker # is inserted to the queue. The symbols are pushed to left and read (popped) from the right.

Comment

Problem

Show that the collection of decidable languages is closed under the operation of

A**a.** union.

b. concatenation.

c. star.

d. complementation.

e. intersection.

Step-by-step solution

Step 1 of 6

A language is said to be decidable if and only if some Turing machine decides it. The Turing machine is a decider if all braches halts on all inputs.

Comments (2)

Step 2 of 6

a.

Let L_1 and L_2 be two decidable Languages. M_1 and M_2 be the Turing machines that decides L_1 and L_2 respectively.

There exists a Turing machine M' such that, decides $L_1 \cup L_2$ i.e. $L(M') = L_1 \cup L_2$.

The description of M' is as follows:

$M' =$on input w:

1. Run M_1 on *w*. if M_1 accepts, then ***accept***.

2. Else Run M_2 on *w*. If M_2 accepts, then ***accept***

3. Else ***reject***

M' Accepts *w* if either M_1 or M_2 accepts it. If both rejects, M' rejects.

Therefore, $L(M') = L_1 \cup L_2$ **. The decidable languages are closed under union.**

Comment

Step 3 of 6

b.

Let L_1 and L_2 be two decidable Languages. M_1 and M_2 be the Turing machines that decides L_1 and L_2 respectively.

There is a Turing machine M' such that, it decides concatenation of L_1 and L_2 i.e., $L(M') = L_1 \circ L_2$.

The description of M' is as follows:

$M' =$on input *w*:

1. Split *w* into two parts w_1, w_2 such that $w = w_1 w_2$

2. Run M_1 on w_1. If M_1 rejected then ***reject***.

3. Else run M_2 on w_2. If M_2 rejected then ***reject***.

4. Else ***accept***

Try each possible cut of *w*. If first part is accepted by M_1 and the second part is accepted by M_2 then *w* is accepted by M'. Else, *w* does not belong to the concatenation of languages and is rejected.

Therefore, $L(M') = L_1 o L_2$ **. The decidable languages are closed under concatenation.**

Comment

Step 4 of 6

c.

Let *L* be a Turing decidable Language and *M* be the Turing machine that decides *L*.

There is a Turing machine M' such that, it decides star of *L* i.e., *L*(M')=L^*.

The description of M' is as follows:

$M' =$ On input *w*:

1. Split *w* into *n* parts such that

$w = w_1 w_2 ... w_n$ in different ways.

2. Run *M* on w_i for $i = 1, 2, ...n$.

If *M* accepts each of these strings w_i, ***accept***.

3. All cuts have been tried without success then ***reject***.

When *w* is cut into different substrings such that every string is accepted by *M*, then *w* belongs to the star of *L* and thus M' accepts *w* after finite number of steps, else *w* will be rejected. Since, there are finitely many possible cuts of *w*, M' will halt after finitely many steps.

Therefore, $L(M') = L^*$ **. The decidable languages are closed under star.**

Comment

Step 5 of 6

d.

For a Turing decidable language L, Turing machine decides language *M* then the complement is M' on input *w.*

The description of M' is as follows:

$M' =$ on input *w*:

1. Accepts if *M* rejects

2. Else ***accept.***

Since M' does the opposite of what ever M does, it decides the complement of L.

Therefore, decidable languages are closed under complementation.

Comments (3)

Step 6 of 6

e.

Let L_1 and L_2 be two Turing decidable Languages. M_1 and M_2 be the Turing machines that decides L_1 and L_2 respectively.

There is a Turing machine M' such that, it decides intersection of L_1 and L_2 i.e., $L(M') = L_1 \cap L_2$.

The description of M' is as follows:

$M' =$ on input *w*:

1. Run M_1 on *w*. if M_1 rejects then ***reject***.

2. Else run M_2 on *w*. if M_2 rejects then ***reject***.

3. Else ***accept***.

M' Accepts *w* if both M_1 and M_2 accept it. If either of them rejects then M' rejects *w*.

Therefore, $L(M') = L_1 \cap L_2$ **. The decidable languages are closed under intersection.**

Comments (2)

Problem

Show that the collection of Turing-recognizable languages is closed under the operation of

A**a.** union.

b. concatenation.

c. star.

d. intersection.

e. homomorphism.

Step-by-step solution

Step 1 of 8

• Suppose, X and Y be two Turing recognizable languages that have Turing machines M_X and M_Y respectively.

• Now the union of these languages is denoted by L_{XY} and the Turing machine recognizing this language is $M_{XY.}$

"On input *w:"*

1. Run X and Y alternately on w step by step. If either accepts, *accept*. If both halt and reject, *reject.*

Suppose s be a word from L_{XY}. M_{XY} works for an input string s as shown:

• It executes M_X and M_Y on s individually.

• If at least any one of M_X orM_Y accepts s then M_{XY} also accepts after a finite number of steps and reach to its accepting state.

• If both M_X and M_Y reject and either of them do so by looping then M_{XY} will loop.

Comment

Step 2 of 8

Hence, it can be said that collection of Turing recognizable languages is closed under union operation.

Comment

Step 3 of 8

• Suppose, X and Y be two Turing recognizable languages that have Turing machines M_X and M_Y respectively. • Now the concatenation of these languages is denoted by L_{XY} and the Turing machine recognizing this language is $M_{XY.}$

Let s be a word from L_{XY}. M_{XY} works for an input string s as shown:

• It divides each string of XY into s_1 and s_2 non-deterministically.

• It runs s_1 to M_X. If M_X halts and rejects, *reject.*

• If runs s_2 to M_Y. If M_Y accepts, *accepts.* If M_Y halts and rejects, *reject.*

Comments (2)

Step 4 of 8

Hence, it can be said that collection of Turing recognizable languages is closed under concatenation operation.

Comment

Step 5 of 8

• Suppose, X be a Turing recognizable language and the Turing machine is M_X.

• Now X^* is the language obtained from star operation on X.

The Turing machine for this language be M_{X^*}.

M_{X^*} works as follows:

• For an input string s of X, it non-deterministically divided the string into $s_1, s_2 \ldots s_n$.

• For each of those divided parts M_{X^*} runs, Suppose, M_{X^*} all divided parts then s is accepted by M_{X^*} else s is rejected by M_{X^*}.

Comment

Step 6 of 8

Hence, it can be said that collection of Turing recognizable languages is closed under star operation.

Comment

Step 7 of 8

• Suppose, X, Y be two Turing recognizable languages that have Turing machines M_X and M_Y respectively.

• Now the intersection of these languages is denoted by L_{XY} and the Turing machine recognizing this language is M_{XY}.

For an input string s from L_{XY}, M_{XY} works for as shown:

• Turing machine M_X runs on s. If it accepts s then M_Y runs on s. Else s is rejected.

• Suppose, M_Y accepts s then it is accepted by the Turing machine otherwise s is rejected.

Hence, it can be said that collection of Turing recognizable languages is closed under intersection operation.

Comments (5)

Step 8 of 8

• Suppose, X a Turing recognizable language that have Turing machine M_X.

• To recognize $h(X)$ the other Turing machine M_Y is simulated in such a way that:

On input s, it will consider all strings w such that $h(w) = s$.

• The *TM* M_X will execute on input w by going through all strings in w.

If $h(w) = s$ start executing M_X on input w, using merging to interleave with other executions on M_X. *Accept* if any executions accept.

• M_Y will accept s it any of those executions of M_X accepts s. Else s will be rejected.

Hence, it can be said that collection of Turing recognizable languages is closed under homomorphism operation.

Comment

Problem

Let $B = \{\langle M_1 \rangle, \langle M_2 \rangle, \ldots\}$ be a Turing-recognizable language consisting of TM descriptions. Show that there is a decidable language C consisting of TM descriptions such that every machine described in B has an equivalent machine in C and vice versa.

Step-by-step solution

Step 1 of 2

In order to solve this problem, we need to know the definition of enumerator and some theorems

Enumerator:-

An enumerator is a Turing machine that consist a work tape and the output tape. It outputs the strings by using the work tape without accepting any input.

Also we use the following theorem

Theorem 1:

"A language is Turing – decidable if and only if some enumerator enumerates the strings of this language in lexicographic order"

Comment

Step 2 of 2

Consider the language $B = \{\langle M_1 \rangle, \langle M_2 \rangle \ldots\}$.

B is a Turing recognizable language.

C is a language consisting of Turing machines descriptions.

Consider *E* be the enumerator for the Turing recognizable language *B*.

Construct an enumerator E_o which output the strings of *C* in lexicographic order.

From the above Theorem1, *C* is decidable.

Enumerator E_o simulates *E*.

When *E* gives the i^{th} TM $\langle M_i \rangle$ as output, then enumerator E_o pads M_i by adding sufficiently many extra useless states to obtain a new TM M_i' where the length of $\langle M_i' \rangle$ is greater than the length of $\langle M'_{i-1} \rangle$. Then E outputs $\langle M_i' \rangle$.

Thus simulation occurs in both directions.

Therefore, E_o and *E* are equivalent.

Comments (1)

Problem

Show that a language is decidable iff some enumerator enumerates the language in the standard string order.

Step-by-step solution

Step 1 of 3

To prove this problem:

We need to show equivalence between a Turing machine that decides a language and an enumerator that enumerates it. Hence we have to show the proof in both directions i.e.,

1. Language is decidable then enumerator enumerates the language in lexicographic order.

2. Enumerator enumerates the language in lexicographic order and then it is decidable.

Comment

Step 2 of 3

1. Language is decidable then enumerator enumerates the language in lexicographic order.

Proof:

If direction:

Let us assume that we have a Turing machine M to decide a language L.

Now we can use this M to construct an enumerator E as follows.

We generate the strings in the lexicographic order and input each string into M for L.

If M accepts then print that string. Therefore E prints all strings of L in lexicographic order.

Comment

Step 3 of 3

2. Enumerator enumerates the language in lexicographic order and then it is decidable.

Proof:

And if direction: -

Now we need to consider the other direction.

That is, if we have an enumerator E for a language L, then we can use E to construct a Turing machine M that decides L.

Here we have to consider two cases.

Case (i): If L is finite:

If L is finite language, then it is decidable, because all finite languages are decidable.

Case(ii) If L is infinite:-

If L is infinite then a decider for L operates as follows.

• On receiving the input w, the decider enumerates all strings of L in lexicographic order until a string greater than w appears in the lexicographic order.

• This must eventually occur since L is infinite.

• If w has appeared in the enumeration already, then *accept*.

• If w has not yet appeared in the enumeration then it will never appear and hence we can *reject*.

So in both cases L is decidable. The theorem proved in both directions.

Therefore,

A language is decidable if and only if some enumerator enumerates the language in lexicographic order.

Comment

Problem

Show that every infinite Turing-recognizable language has an infinite decidable subset.

Step-by-step solution

Step 1 of 2

Let *L* be an infinite Turing recognizable language.

We know that

“A language is Turing recognizable if and only if some enumerator enumerates it”

So, let M be the enumerator that enumerates L.

Consider a language $L' = \{w_1, w_2, w_3, ...\}$ where

$\rightarrow w_1$ is the first string enumerated by *M*

$\rightarrow$ For every $i > 1$,

w_i is the first string enumerated by *M* and that is lexicographically larger than w_{i-1}.

(i) **First we prove L' is infinite and is subset of L:**

Let us assume that L' is finite

Then w_i is the last and lexicographically largest element in L' and all strings enumerated by M must be lexicographically less than w_i.

Since there are only a finite number of strings that are less than w_i, L would then be finite.

This is a contradiction because *L* is infinite.

Therefore, our assumption that L' is finite is wrong.

Hence L' is infinite.

All the strings in L' were at some point enumerated by M.

So clearly L' is subset of *L*.

Comment

Step 2 of 2

(ii) **Next we have to prove L' is decidable:**

Now we will show that L' is decidable for the given enumerator *M*, we can construct an enumerator M' in lexicographic order.

M' does the following

• Let *w* be the last string that M' emitted, and initialize *w* to a dummy value that comes lexicographically before all strings.

• Simulate *M* until it emits a string *t*

• If: $t > w$ lexicographically

Then set $w = t$ and Let M' emits *t*.

• Else: Ignore *t*.

• Resume simulating *M*, and go to step 2

Thus M' is constructed and that enumerated L' in lexicographic order.

We know that,

“A language is decidable if and only if some enumerator enumerates the language in lexicographic order”

So by this theorem L' is decidable.

Therefore from (i) and (ii) L' is an infinite decidable subset of *L*.

Comment

Problem

Show that single-tape TMs that cannot write on the portion of the tape containing the input string recognize only regular languages.

Step-by-step solution

Step 1 of 3

Let M be the single – tape Turing machine that cannot write on the portion of the tape containing the input string.

$M = (Q, \Sigma, \tau, q_0, q_{accept}, q_{reject})$

M works on an input string *x* as follows.

Here we consider two events.

(i) **Out event:**

In out event, the tape head moves from input portion to non – input portion, i.e., the portion of the tape on the right of the $(x)^{th}$ cell.

(ii) **In event:**

In In-event tape head moves from non – input portion to input portion.

Comment

Step 2 of 3

Consider the state q_x for Turing machine $M = (Q, \Sigma, \tau, q_0, q_{accept}, q_{reject})$ when it first enters the non – input portion (i.e., after it's first *out event*)

• In case M never enters the non – input portion.

(a) If M accepts *x,* assign $q_x = q_{accept}$

(b) If M does not accept *x,* assign $q_x = q_{reject}$

For any $q \in Q$, define a characteristic function f_x such that

$f_x(q) = q'$

That implies

If *M* is in the state *q* and about to perform an "*in event*", the next "*out event*" will change M in state q'

• In case M never enters the non – input portion again,

(a) If M enters the accept state inside the input portion, assign $f_x(q) = q_{accept}$

(b) If M does not enter the accept state, assign $f_x(q) = q_{reject}$

For two strings *x* and *y*,

If $q_x = q_y$ for all *q*, $f_x(q) = f_y(q)$, then *x* and *y* are indistinguishable by M. That is, M accepts *xz* if and only if M accepts *yz*.

Comments (1)

Step 3 of 3

As there are finite choice of q_x and f_x $\left(\text{Precisely } |Q|^{|Q|+1} \text{ such choices}\right)$, the number of indistinguishable strings are finite.

"Myhill – Nerode theorem" is used to prove whether the language is regular or not.

Statement:

A language L over alphabet Σ is regular if and only if the set of equivalent classes of I_L is finite.

I_L is the relation on Σ^* such that for two strings *x* and *y* of Σ^*

$$x I_L y \Leftrightarrow \{z \mid xz \in L\} = \{z \mid yz \in L\}$$

That is $xI_L y$ if and only if they are indistinguishable with respect to L

So, by Myhill – Nerode theorem the language recognized by M is regular.

Comments (1)

Problem

Let $c_1x^n + c_2x^{n-1} + \cdots + c_nx + c_{n+1}$ be a polynomial with a root at x = x_0. Let c_{max} be the largest absolute value of a ci. Show that

$$|x_0| < (n+1)\frac{c_{\max}}{|c_1|}.$$

Step-by-step solution

Step 1 of 3

Consider a polynomial $f(x) = c_1x^n + c_2x^{n-1} + \cdots + c_nx + c_{n+1}$.

Since above polynomial has a root at

$x = x_0$

So $f(x_0) = 0$ implies that

$c_1{x_0}^n + c_2{x_0}^{n-1} + \cdots + c_nx_0 + c_{n+1} = 0$

Rearrange the above equation as:

$c_1x_0^n = -\left(c_2x_0^{n-1} + \ldots + c_nx_0 + c_{n+1}\right)$

Now, take modulus on both the sides of the equation as:

$\left|c_1{x_0}^n\right| = \left|-\left(c_2{x_0}^{n-1} + \cdots + c_nx_0 + c_{n+1}\right)\right|$

$|c_1|{x_0}^n = \left|\left(c_2{x_0}^{n-1} + \cdots + c_nx_0 + c_{n+1}\right)\right|$

Comments (4)

Step 2 of 3

Use sub-additive property $|a+b| \le |a| + |b|$ of modulus function,

$\left|c_1{x_0}^n\right| \le \left|c_2{x_0}^{n-1}\right| + \ldots + |c_nx_0| + |c_{n+1}|$

Also, as c_{max} is the largest absolute value of c_i then for each $i = 1, 2, \cdots, (n+1)$,

$c_{\max} = |c_{n+1}|$

So, from above equation,

$\left|c_1{x_0}^n\right| \le c_{\max}\left(1 + |x_0| + \ldots + \left|{x_0}^{n-1}\right|\right)$

Comments (2)

Step 3 of 3

Substitute $n{x_0}^{n-1}$ for $1 + |x_0| + \ldots + \left|{x_0}^{n-1}\right|$ where, ${x_0}^{n-1}$ is the largest one if $x_0 > 1$:

$$\left|c_1 {x_0}^n\right| \leq c_{\max} \cdot n\left|{x_0}^{n-1}\right|$$

$$\left|\frac{{x_0}^n}{{x_0}^{n-1}}\right| \leq n.\frac{c_{\max}}{|c_1|}$$

$$\left|{x_0}^{n-(n-1)}\right| \leq n.\frac{c_{\max}}{|c_1|}$$

$$|x_0| \leq n.\frac{|c_{\max}|}{|c_1|}$$

It also can be written as:

$$|x_0| \leq n.\frac{c_{\max}}{|c_1|}$$

To make the term $n.\frac{c_{\max}}{|c_1|}$ strictly greater than $|x_0|$, we can re-write as:

$$|x_0| < (n+1)\frac{c_{\max}}{|c_1|} \quad (\text{since, } n < n+1 \text{ always holds})$$

Comment

Problem

Let A be the language containing only the single string s, where

$$s = \begin{cases} 0 & \text{if life never will be found on Mars.} \\ 1 & \text{if life will be found on Mars someday.} \end{cases}$$

Is A decidable? Why or why not? For the purposes of this problem, assume that the question of whether life will be found on Mars has an unambiguous YES or NO answer.

Step-by-step solution

Step 1 of 1

Decidable language: A language is decidable if and only if some Turing machine decides it.

We call a Turing machine a decider if all branches halt on all inputs.

The given data is, A is the language containing only the single string s, where

$$s = \begin{cases} 0 & \text{if life never will be found on Mars} \\ 1 & \text{if life will be found on mars someday} \end{cases}$$

So, the language A may contain either 0 or 1 but not both

Thus $A = \{0\}$ (or) $A = \{1\}$

In both these cases A are finite and string s is fixed, so we know the same finite language is always decidable.

We are not able to determine whether

$A = \{0\}$ or $A = \{1\}$.

So we will not able to describe the decider for A.

In this case, we need to give two Turing machines for both the cases. So, definitely one of them will be the decider of *A*.

Comment