Problem

Show that EQ_{CFG} is undecidable.

Step-by-step solution

Step 1 of 1

9640-5-1E AID: 1112 I 27/06/2014

RID: 6385 I 07/08/2015

Undecidable language:

The problem of determining whether a string or input can be accepted by a Turing machine or not is called Undecidability. The decidability of the Context-free grammar depends on the decidability of the Turing machine.

Proof to show that $\mathrm{EQ}_{\mathrm{CFG}}$ is undecidable:

Step-1:

Consider a context-free grammar CFG $G_0 = (V, \Sigma, R, S)$ where $V = \{S\}$ and S is a starting variable. Assume that there is a rule $S \rightarrow lS$ in R for every terminal $l \in \Sigma$. The grammar G_0 includes a $\in$ notation by using the rule $S \rightarrow \in$.

Example:

For the CFG, the rules in G_0 are defined as $S \rightarrow aS \mid bS \mid \in$ over the alphabet set $\Sigma = \{a, b\}$.So, the grammar CFG G_0 satisfies all the alphabets in the alphabet set Σ.

So, $L(G_0) = \Sigma^*$. Thus, the Turing Machine is decidable.

Step-2:

Assume that the CFG is decidable by using the Turing machine R that decides $\mathrm{EQ}_{\mathrm{CFG}}$. Construct another Turing machine S which uses R to decide ALL_{CFG} by using the following procedure:

$S = On\ input\ \langle G_0 \rangle,$

1. Run R on the input $\langle G_0, G_1 \rangle$. G_1 is a CFG, which generates Σ^*.
2. Accept the grammar, when R accepts.
3. Otherwise reject.

一般利用ATM，但其他其实也行

Thus, if the Turing machine R decides $\mathrm{EQ}_{\mathrm{CFG}}$, S also decides ALL_{CFG} which is impossible. So, $\mathrm{EQ}_{\mathrm{CFG}}$ is also undecidable.

Comment

Problem

Next

Show that EQ_{CFG} is co-Turing-recognizable.

Step-by-step solution

Step 1 of 1

Remember that EQ_{CFG} is co-Turing-recognizable language if and only if its complement $\overline{EQ_{CFG}}$ is a Turing-recognizable language.

Now, $\overline{EQ_{CFG}} = A \cup B$, where

$$A = \{ w \mid w \textit{ does not have the form } \langle G1, G2 \rangle \textit{ for some CFGs } G1 \textit{ and } G1 \},$$

$$B = \{ \langle G1, G2 \rangle \mid \text{G1 and G2 are CFGs and } L(G1) \neq L(G2) \}.$$

- *A* contains strings that defy the syntax for encoding $\langle G1, G2 \rangle$) is simple to accept.
- The set B is realized in the following way:
- Transform the G1, G2 to Chomsky Normal Form (CNF).
- Then begin numbering strings in Σ^* lexicographically, where Σ is the group of terminals for G1, G2.
- For every string *w* numbered, check if it is produced by G1 and by G2.
- If the 2 Context Free Grammars or both of them cannot produce *w*, then TM goes on to generate the next string in the lexicographic order.
- Else, precisely one of the CFGs produces the string, and the TM accepts.
- Therefore, B is a Turing recognizable language.
- It is already proved that Turing-recognizable languages are closed under union, so EQ_{CFG} is Turing-recognizable language.
- Let the list of strings that are listed in lexicographic order are as follows:

s1, s2, s3 . . . over the input Σ^* .

Now EQ_{CFG} is realized by the following *TM M*:

M = "On input $\langle G1, G2 \rangle$, where G1, G2 are Context Free Grammars:

1. Examine if G1, G2 are valid Context Free Grammars. If atleast 1 does not, accept.

2. Transform G1, G2 into corresponding Context Free Grammars G'_1 , G'_2 .both into CNF.

3. Replicate the below step 4 for j = 1, 2, 3 . . .

4. Examine if G'_1 and G'_2 produce s_j if precisely one of them does, *accept*."

Hence it is proved that *EQ* $_{CFG}$ is co-Turing-recognizable language.

Comment

Problem

Find a match in the following instance of the Post Correspondence Problem.

$$\left\{ \left[\frac{ab}{abab}\right], \left[\frac{b}{a}\right], \left[\frac{aba}{b}\right], \left[\frac{aa}{a}\right] \right\}$$

Step-by-step solution

Step 1 of 2

Post Correspondence Problem can be considered as an example of undecidability problem concerning with manipulation of strings to find a match.

A match can be found if the string made or created by combining all the symbols of upper side and string made by combining all the symbols of lower side, both are same.

Comment

Step 2 of 2

Consider the instance of Post Correspondence Problem with the collection of dominos as follows:

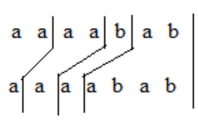
$$\left\{ \left[\frac{ab}{abab}\right], \left[\frac{b}{a}\right], \left[\frac{aba}{b}\right], \left[\frac{aa}{a}\right] \right\}:$$

The match for the given problem is a sequence of 4, 4, 2, and 1 using the dominos 1, 2 and 4:

$$\left\{ \left[\frac{aa}{a}\right], \left[\frac{aa}{a}\right], \left[\frac{b}{a}\right], \left[\frac{ab}{abab}\right] \right\}$$

The sequence produces same string aaaabab while reading off the top and bottom of the sequence 4,4,2,1.

The match can be depicted as follows:



Therefore, a match 4,4,2,1 is found for the given Post Correspondence Problem.

Comments (2)

Problem

If A $\leq_m$ B and B is a regular language, does that imply that A is a regular language? Why or why not?

Step-by-step solution

Step 1 of 1

No, *A* is not a regular language.

• Assume that the languages *A* is defined as follows:

$A = \left\{ a^n b^n \mid n \geq 0 \right\}$ *and* $B = \{b\}$, over the input $\Sigma = \{a, b\}$.

• Specify the function $f : \Sigma^* \rightarrow \Sigma^*$ in the following way:

$$f(w) = \begin{cases} b & \text{if } w \in A, \\ a & \text{if } w \notin A. \end{cases}$$

• Notice that if *A* is a context-free language, then it is Turing-decidable.

• Therefore, f is a computable function.

• Besides, $w \in A$ if and only if $f(w) = b$, which is true if and only if $f(w) \in B$.

Hence it is proved that language A is not-regular, but language B is a regular language, because it is finite.

Comment

Problem

Next

Show that A_{TM} is not mapping reducible to E_{TM}. In other words, show that no computable function reduces A_{TM} to E_{TM}. (Hint: Use a proof by contradiction, and facts you already know about A_{TM} and E_{TM}.)

Step-by-step solution

Step 1 of 3

Refer theorem 5.2 in the textbook. It states that E_{TM} is undecidable. It is known that A_{TM} is Turing recognizable but not co-Turing recognizable.

Comment

Step 2 of 3

The complement of E_{TM} is denoted with $\overline{E_{TM}}$. The TM for $\overline{E_{TM}}$ is as follows:

M="On input <M>,

1. For l = 1, 2, 3, ...
   a. Run M on all strings of length l for l steps.
   b. If any string is accepted then accept.
2. Reject if no string is accepted."

There exists a TM that recognizes $\overline{E_{TM}}$. Thus, E_{TM} is co-Turing recognizable.

Comment

Step 3 of 3

Assume that A_{TM} is mapping reducible to E_{TM}. Thus, $\overline{A_{TM}}$ is mapping reducible to $\overline{E_{TM}}$. $\overline{A_{TM}}$ is not Turing recognizable but $\overline{E_{TM}}$ is Turing recognizable which is a contradiction to the theorem 5.28. This a contradiction to the earlier assumption.

Therefore, A_{TM} is not mapping reducible to E_{TM}.

Comment

Problem

Show that $\leq_m$ is a transitive relation.

Step-by-step solution

Step 1 of 2

Proving relation is transitive

Transitive relations are those in which one element is related to the second element and second element is related to the third element.

In this situation, the first element should be related to the third element. If the condition is fulfilled, then it can be said that the relation is transitive.

Comment

Step 2 of 2

Now, for proving transitivity of $\leq_m$ two relations are required.

Consider the first relation for showing the transitivity:

$A \leq_m B$

Consider the second relation for showing the transitivity:

$B \leq_m C$

Now, computational function for first relation is as shown:

$x \in A \Leftrightarrow f(x) \in B$

Here, *x* is considered as input string on *A* and output of *A* is *f(x)*

Now, computational function for second relation is as shown:

$y \in B \Leftrightarrow g(y) \in C$

Here, *y* is considered as input string on *B* and output of *B* is *g(y)*

Now, consider a composition function that can be considered as the mapping function between *f* and *g*.

$h(x) = g(f(x))$

Now, find the mapping between these two functions, it may be required to create a Turing Machine that computes *h* as follows:

• Simulate a Turing Machine on input *x* and call the output *y*, this will simulate a Turing Machine for *g* on *y*.

• Now, the output of this input can be viewed as considered function that is, $h(x) = g(f(x))$.

In this way:

$x \in A \Leftrightarrow h(x) \in C$

Thereby, $h(x)$ is computable function.

Here, input *x* on *A* is directly dependent on *h(x).*

Hence, if $A \leq_m B$ and $B \leq_m C$ then $\mathrm{A} \leq_m \mathrm{C}$.

Therefore, $\leq_m$ is Transitive Relation.

Comment

Problem

Show that if A is Turing-recognizable and $A \leq_m \overline{A}$, then A is decidable.

Step-by-step solution

Step 1 of 1

Proving decidability of language

Consider the Turing machine *M* which is use for recognizing the language *A* in such a way that $A = A(M)$.

So, it can be said that language *A* is Turing-recognizable or even it can be said that it is recognizable.

A Turing machine is use for deciding the language *A* if $A = A(M)$ and Turing machine *M* hold for each and every input.

So, it can be said that *A* is decidable if and only if Turing machine *M* is use for deciding *A*.

Suppose, $A \leq_m \overline{A}$, then it is quite obvious $\overline{A} \leq_m A$ also exists by using the same mapping reducibility function.

A is Turing recognizable then $\overline{A}$ is also recognizable as follows:

Assume *M* is the recognizer for Turing Machine $\overline{A}$ and *N* is the recognizer for *A*. *F* is the reduction function for *A* to $\overline{A}$.

N can be described as:

N is the recognizer and recognizes input or string *w*.

N = Input *w*:

• Compute *F(w)*: *F(w)* function is mapping function that computes mapping reducibility between Turing Machines *P* and *Q*.

• Run *M* on input *F(w)* and output whatever *M* output.

As *M* is the recognizer for $\overline{A}$, now run the output *F(w)* on *M* to Find Mapping reducibility between Turing Machines *A* and $\overline{A}$.

This implies that $\overline{A}$ is also Turing Recognizable.

If A and $\overline{A}$ is Turing recognizable then it can be proved that $A \leq_m \overline{A}$ is also decidable.

A language is decidable if its components are Recognizable or co-recognizable as it is already proved that *A* and $\overline{A}$ both are recognizable then consider *P* for deciding the language *A*.

Let P_A and $\overline{P_A}$ is use for deciding that A and $\overline{A}$ is recognizable.

• For any value of input *x* whether it is 1,2, 3 user need to simulate the value for P_A and $\overline{P_A}$ for the finite number of steps. If $x \in A$ then simulation is accepted and if there is the situation that $x \notin A$ then simulation is halted.

• Run both the decider P_A and $\overline{P_A}$ in parallel for the particular input *x* till either of them accepts.

• If $\overline{P_A}$ is accepted then accept it for the particular value of *x* and then halt. If $\overline{P_A}$ is accepted then reject the particular value of *x* and after that halt the Turing machine.

Running P_A and $\overline{P_A}$ in parallel means Turing Machine have 2 tapes 1 for simulating P_A and another for simulating $\overline{P_A}$ it continue until one of them accepts.

Now, it is quite obvious that input *x* is whether running on P_A or $\overline{P_A}$ so it must be accepted by one of them Turing Machine is halted whenever P_A or $\overline{P_A}$ accepts *x*. It accepts all strings in *A* and rejects all strings in *A* so P_A is decider for *A* and *A* is decidable.

As, for every input, Turing machine is halted for each and every input, then it can be said that $A \leq_m \overline{A}$ is decidable.

Comments (1)

Problem

In the proof of Theorem 5.15, we modified the Turing machine M so that it never tries to move its head off the left hand end of the tape. Suppose that we did not make this modification to M. Modify the PCP construction to handle this case.

THEOREM **5.15**

PCP is undecidable.

Step-by-step solution

Step 1 of 1

Given:

In theorem 5.15 Turing Machine *M* is modified so that head is never moved to the left hand side of the tape. Now, suppose modification is not made to Turing Machine *M.*

Proof:

Here, it is considered that changes are not made to Turing Machine so users are supposed to modify the Post Correspondence Problem to handle this case. Modification of Post Correspondence Problem is done as follows:

It is assumed that changes are not made to the Turing Machine so modification of PCP is done. As it is also given in the question that modification is done in a way so it head is never moved to the left end of the tape. So assume a case where its head at the leftmost tape cell and attempts to move left to do so add dominos left further:

$$[\frac{\#qa}{\#rb}]$$

Here, for every $q,r \in Q$ and $a,b \in r$, where $\delta(q,a)=(r,b,L)$.

And then replace the first dominos in the following manner:

$$[\frac{\#}{\#\#q_0,w_1,w_2....w_n}]$$

Hence, this Approach should be used to handle the case where head tries to move to the left.

Comment

Problem

Let

$$T = \{\langle M \rangle |\ M \text{ is a TM that accepts } w^{\mathcal{R}} \text{ whenever it accepts } w\}.$$

Show that T is undecidable.

Step-by-step solution

Step 1 of 4

Consider the problem statement provided in the textbook.

Comment

Step 2 of 4

Let $T = \{ < M >| M \text{ is a TM that accepts } w^R \text{ whenever it accepts } w \}$.

• It is already known that $L = \{(w, M): w \textit{ is accepted} \text{ by } M\}$ is undecidable.

• Assume that T is decidable, then there must exist a TM by which T can be decided. Let's say P is the Turing Machine that decides T.

Comment

Step 3 of 4

For any input (w, M), M' can be constructed as follows:

If $w = w^R$, simulate M on w. The Σ is the alphabet set of M and let $a, b \notin M$.

Let $\Sigma \cup \{a, b\}$ be the alphabet set of M'. Then for input ab, M' will reject all the other strings except ab.

Now, simulate M on w.

• If M accepts w, M' rejects.

• If M rejects w, M' accepts.

Claim: P accepts M' iff M accepts w.

Proof: If P accepts M'. Since, M' rejects all the other strings which include ba also, then M' rejects ab which implies M accepts w.

If w is accepted by M, then M' rejects ab. Since, M' rejects all the other strings, M' is accepted by P.

Now, construct a TM, Q for L. Construct M' on input (w, M) and run P on it. Q accepts iff P accepts.

This contradict the fact that L is undecidable.

Comment

Step 4 of 4

Therefore, T is undecidable. Hence Proved.

Comments (2)

Problem

Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape when it is run on input w. Formulate this problem as a language and show that it is undecidable.

Step-by-step solution

Step 1 of 2

Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape when it is run on input *w*.

Comment

Step 2 of 2

The language that denotes the problem is,

$L = \{< M, w > | \; M$ is a two-tape Turing machine writes a nonblank symbol on second tape when it is run on input $w\}$

In order to check the decidability, show that Construct a TM A_{TM} reduces to L. Use proof by contradiction method to prove decidability of the language. Assume TM T decides L. Construct a TM B that uses T to decide A_{TM}.

B = "On input $< M, w >$:

1. Use M to construct the two-tape Turing machine S.

   S="On input x:

   1. Simulate M using the first tape on input x.
   2. If M is accepted then write a nonblank symbol on the second tape."

2. Run T on $< S, w >$ to check whether S on input w writes a nonblank symbol on second tape.
3. If T accepts, M accepts w then accept. Otherewise, reject."

Thus, T decides A_{TM} which is undecidable. Therefore, L is undecidable.

Comment

Problem

Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape during the course of its computation on any input string. Formulate this problem as a language and show that it is undecidable.

Step-by-step solution

Step 1 of 1

Undecidability of the Turing machine problems

In the problem it is given that a single tape Turing machine writes a blank symbol over a non-blank symbol or not in the time of computation on any string which is provided as input. For showing the language generated by formulating the problem is undecidable following proof has been written:

$$K = \left\{ \langle M, w \rangle \middle| \begin{array}{l} M \text{ is a Turing machine for input w,} \\ \text{Writes a non blank symbol on second tape.} \end{array} \right\}$$

Now, define a two tape Turing machine N_1 . This Turing machine takes a pair of M and *w* or $\langle M, w \rangle$ as input. Here, M is a Turing machine and w is a string which is passed as input to Turing machine. Turing machine N_1 is stimulating the Turing machine M , by using the first tape of machine M on input w .

Turing machine N_1 will ignore the second tape of the Turing machine M till end of the string does not come. If the Turing machine M accepts the input string w then the Turing machine N_1 will write a non-blank symbol on its second tape. If the simulation is halt then the Turing machine has stopped working.

Now assume T decides *K*. Then, user create *T'* for deciding A_{TM} . When the input $\langle M, w \rangle$ is passed to the Turing machine, firstly, it simulates the Turing machine N_1 and after that it will run on the Turing machine T on N_1. The language K is accepted when T accept it otherwise, vice versa.

So, Turing machine T' is accepted if and only if when T accepts. T accepts the $\langle M_1, \langle M, w \rangle \rangle$ if and only if *M1* has written on second tape and M accepts w. So, ATM is decided by T'. But, A_{TM} cannot contain any decider. Here, in the above problem, for deciding *K*, any T is not present so the language K is undecidable.

Comment

Problem

Consider the problem of determining whether a single-tape Turing machine ever writes a blank symbol over a nonblank symbol during the course of its computation on any input string. Formulate this problem as a language and show that it is undecidable.

Step-by-step solution

Step 1 of 3

Formulating the given problem as a language:

$$L=\left\{\langle M,w\rangle \middle| \begin{array}{l} M \text{ is a single tape Turing machine which writes a blank symbol} \\ \text{on non blank symbol while computing any input string} \end{array}\right\}$$

Comment

Step 2 of 3

Proving that the given problem is undecidable:

By using contradiction, assume that the language L is decidable. Suppose that N is a decider for proving the decidability of the language L. A Turing machine *N* can be constructed as:

N = "On Input $\langle M,s\rangle$

• Construct a Turing machine A' now:

a. A'writes # (a non-blank symbol) if *M* writes a blank symbol

b. Whenever A'reads #, use the transitions specified by the blank symbols.

c. A'Writes # on the tape before accepting and overwrites it with a blank symbol.

• Output of A'' will be input for decider N. If $N(\langle M',s\rangle)$ accepts, accept, otherwise reject.

Comments (2)

Step 3 of 3

Now, the conclusion can be made that a blank symbol is written by A' only when A' takes the input s. That is, N is a decider for A_{TM} which is a contradiction. Hence, **the given problem is undecidable.**

Comment

Problem

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

Step-by-step solution

Step 1 of 1

Undecidability of the Turing machine problem:

• The given problem is defined as the following language:

$USELESS_{TM} = \{\langle T, q\rangle \mid q \text{ is a useless state in TM T}\}.$

• Show that $USELESS_{TM}$ is undecidable by reducing E_{TM} to $USELESS_{TM}$,where $E_{TM} = \{ \langle T1\rangle \mid \text{T1 is a TM and } L(T1) = \emptyset \}$.

• Using the Theorem 5.2. it is already proved that E_{TM} is undecidable

• Suppose that $USELESS_{TM}$ is decidable and that *TM R* decides it.

• Note that for any Turing machine M with accept state q_{accept}, q_{accept} is useless if and only if $L(T1) = \emptyset$.

• Accordingly, since *TM* R solves $USELESS_{TM}$, *R* can be used to check if q_{accept} is a useless state to decide E_{TM} .

Specifically, below is a *TM S* that decides E_{TM} by using the decider R for $USELESS_{TM}$ as a subroutine:

S = "On input $\langle T\rangle$, where *M* is a *TM*:

1. Run TM R on input $\langle T, q_{accept}\rangle$, where q_{accept} is the accept state of *T*.
2. If *R* accepts, accept. If *R* rejects, reject."

However, since it is known E_{TM} is undecidable and there cannot be a *TM* that decides $USELESS_{TM}$.

Hence it is proved, that the given problem is undecidable.

Comments (5)

Problem

Consider the problem of determining whether a Turing machine M on an input w ever attempts to move its head left when its head is on the left-most tape cell. Formulate this problem as a language and show that it is undecidable.

Step-by-step solution

Step 1 of 2

Consider the problem of determining whether a Turing machine *M* on an input *w* ever attempts to move its head left when its head is on the left-most tape cell. This problem is formulated as a language:

$L = \{< M, w >| \; M \text{ attempts to move its head left when its head is on the leftmost tape cell}\}$

Comment

Step 2 of 2

Assume that the language *L* is decidable and $\hat{M}$ be a TM that decides the language *L*. Construct a TM, *A* that decides the halting problem.

A = "on input $<M, \; w>$:

1. Construct a TM, A', from A. The TM A' moves w one tape cell to the right and marks the leftmost cell with #.
2. Run the TM A' on $<M, \; w>$.
3. If A' encounters # then A' moves to the right side and simulates M reaching the leftmost tape cell.
4. If M halts and accepts on w then A' simulates to move its head left when its head is on the leftmost tape cell."

Now, TM *A* runs $\hat{M}$ on the input $<A, \; w>$. If $\hat{M}$ accepts, *A* accepts. Otherwise, *A* rejects. It is assumed that $\hat{M}$ be a TM that decides the language *L*. If *M* halts and accepts on *w,* then only A' moves its head left when its head is on the left-most tape cell. If *A* decides the halting problem, then halting problem is decidable. Thus, the halting problem is undecidable. It is a contradiction.

Therefore, the language *L* is undecidable.

Comments (1)

Problem

Consider the problem of determining whether a Turing machine M on an input w ever attempts to move its head left at any point during its computation on w. Formulate this problem as a language and show that it is decidable.

Step-by-step solution

Step 1 of 2

Consider a problem of determining whether a Turing machine M on input w ever attempts to move its head left at any point during its computation on w. The language that describes the problem is,

$$L = \{< M, w >| \ M \text{ moves its head left on input } w\}$$

Comment

Step 2 of 2

Construct a Turing machine A that decides the problem.

A = "On input $<M,\ w>$:

1. Run the machine for $|Q|+|w|+1$ steps.
2. If the Turing machine M's head moved to the left then accept. Otherwise, reject."

Here, $|Q|$ represents the number of states and $|w|$ represents the length of the input string. The problem is said to be decidable, if the Turing machine M moves its head left on input w within the first $|Q|+|w|+1$ steps. The problem is decidable because, there exists a Turing machine for it.

Therefore, the language *L* is decidable.

Comments (1)

Problem

Let $\Gamma = \{0, 1, \sqcup\}$ as follows. For each value of k, consider all k-state TMs that halt when started with a blank tape. Let *BB*(k) be the maximum number of 1s that remain on the tape among all of these machines. Show that *BB* is not a computable function.

Step-by-step solution

Step 1 of 1

Given: A tape of alphabets Γ for all the Turing machines is given. Γ Is composed of string of three terminals which is as follows: $\Gamma = \{0,1,\sqcup\}$

For every value of k all k states, of Turing machine halts, when the Turing machine starts from the blank tape. Maximum number of 1s is $BB(k)$ which remains in tape.

Proof:

By using contradiction, assume that the busy beaver function BB is computable. If function BB is computable then there exists a Turing machine F for computing it. Now without loss of generality a Turing machine F on input 1^n, and the Turing machine F halts with $1^{BB(n)}$ for each value of n.

Now build a Turing machine M, and this Turing machine halts when it will start from a blank tape based on F.

Construction of Turing machine:

Here M is a Turing machine which halts when starts from blank tape.

Step1: Now the Turing machine M writes n number of 1s on the tape.

Step2: Turing machine M doubles the number of 1s on the tape.

Step 3: Now M executes the Turing machine F on the input 1^{2n}.

Hence Turing machine M will halts with $BB(2n)$ number of 1s if it is starts from the blank tape.

For implementing the Turing machine M, at most n numbers of states are required for the step 1 of the Turing machine and c numbers of states are required for step 2 and 3, c is a constant.

Conclusion:

By definition, $BB(n+c)$ is the maximum number of1s on which Turing machine with states $(n+c)$ will halt is at least the number of 1s on which the Turing machine M halts. Which means $BB(n+c) \geq BB(2n)$ and this relationship will hold for all values of n.

Therefore, $BB(k)$ is strictly increasing function so $BB(n+c) < BB(2n)$. It proves wrong to our contradiction.

Hence, it is clear that $BB(k)$ is not a computable function.

Comment

Problem

Show that the Post Correspondence Problem is decidable over the unary alphabet ∑ = {1}.

Step-by-step solution

Step 1 of 1

Decidability of PCP over unary alphabet

Post Correspondence Problem is basically concerned with manipulation of string and used to find match. Conceptually this concept is quite obvious by its statement itself as there is only 1 alphabet in the Post Correspondence Problem. PCP is decidable over the unary alphabet as follows:

Consider a Turing Machine M that runs on input $<P>$

1. Check for some i, a_i is equals to b_i then accept the string.

2. Check if $a_i > b_i$ or $a_j < b_j$ if there exist some i and j then accepts the string otherwise reject it.

For the first stage, Turing machine M verifies for a domino which is forming a match. In the second stage Turing machine M is looking for two dominos which are forming a match. If the Turing machine finds such pair then it builds a math by extracting $\left(b_j - a_j\right)$ copies of the i^{th} dominos. After extracting copies the Turing machine puts them together with $\left(a_i - b_i\right)$ copies of the j^{th} dominos.

This construction has $a_i\left(b_j - a_j\right) + a_j\left(a_i - b_i\right) = a_i b_j - a_j b_i$

1's on the top and $b_i\left(b_j - a_j\right) + b_j\left(a_i - b_i\right) = a_j b_i - a_j b_i$

1's on bottom. If any stage of Turing machine is not accepted then problem instance includes dominos with all upper parts having either more or less 1's than lower parts.

In this case, any match is not present hence Turing machine M rejects.

Comment

Problem

Show that the Post Correspondence Problem is undecidable over the binary alphabet ∑ = {0,1}.

Step-by-step solution

Step 1 of 1

Post Correspondence Problem is basically concerned with manipulation of string and used to find match.

Conceptually this concept is quite obvious by its statement itself as there are 2 alphabets in the Post Correspondence Problem. PCP is un-decidable over the two alphabets 0 and 1. This can be proved by using contradiction.

Assume the instance of PCP where the size of alphabet is *n* such that $n \leq 2^m$, *m* is a constant. M-bit code is generated for the alphabets and then substituted in the tile.

Consider the tile in PCP given below:

$$\{[\frac{aba}{acd}],[\frac{aabd}{acd}],[\frac{a}{bc}],[\frac{acd}{bc}],[\frac{bcd}{abc}]\}$$

If the values of *a*, *b*, *c* and *d* is assigned as 00, 01, 10 and 11, the tiles given above will be written as:

$$\{[\frac{000100}{001011}],[\frac{00000111}{001011}],[\frac{00}{0110}],[\frac{001011}{0110}],[\frac{011011}{000110}]\}$$

The new PCP instance have alphabet size 2 will have solution if the original PCP instance have some solution. In the code, one-for-one substitution takes place. So, if the new instance has a solution, the solution will also exist for the original instance.

It should be taken care of the case when the new tiles provide a solution but the original tiles don't provide any solution. But this will never happens it is because if the tiles are obtained after inserting the values in the original instance.

For unary alphabets, the proof does not work. The codes are applied for *a*, *b*, *c* and *d* as 1, 11, 111 and 1111. If tracing is done backwards, it will be difficult to identify whether 1111 refers to *d*, *bb*, *ac* or *ca*(there will be other possibilities too). So, PCP is un-decidable.

over{1} 则是decideable

Comment

Problem

In the *silly Post Correspondence Problem*, *SPCP*, the top string in each pair has the same length as the bottom string. Show that the *SPCP* is decidable.

Step-by-step solution

Step 1 of 2

In silly Post correspondence problem or SPCP, each pair of both strings (top and bottom) have same length. Here, the concern is related to whether the instance contains a match or not. SPCP problem is decidable. Decidability of SPCP can be proved as follows:

Consider an instance of SPCP given below:

$$\{[\frac{a_1}{b_1}], [\frac{a_2}{b_2}], \ldots\ldots, [\frac{a_n}{b_n}]\}$$

Here, $|\mathrm{a}_i| = |b_k|$ for all $1 \le i \le k$.

Comment

Step 2 of 2

After finding the match of SPCP instance, it is checked whether the numerator is equal to denominator or not for the domino. Here, the length of the strings at top and bottom are similar.

Trivial match in a single domino of SPCP is formed whenever the top and bottom string is same.

After examining the instance having same top and bottom string, it is said to be as decidable. Dominos are working as decider to find whether the top string and bottom string are same or not. This it is quite easy to find decidability of SPCP.

Comment

Problem

Prove that there exists an undecidable subset of $\{1\}^*$.

Step-by-step solution

Step 1 of 2

Un-decidable subset of $\{1\}^*$ exists. Existence of un-decidable subset can be found as follows:

Consider there is an i^{th} position in any string generated from $\{1\}^*$. It is quite obvious that there are uncountable number of strings can be made using $\{1\}^*$.

Comment

Step 2 of 2

Let *S* be the subset of $\{1\}^*$. The i^{th} position of the generated string would be 1 if the generated string is in *S*. The number of subsets of $\{1\}^*$ is equal to the infinite number of binary strings which is uncountable. The uncountable set is not Turing recognizable because there is more number of subsets than the number of Turing machines.

Therefore, as it is proved above it is not recognizable so it will be un-decidable.

Comment

Problem

Let $AMBIG_{CFG}$ = $\{\langle G\rangle \mid G \text{ is an ambiguous CFG}\}.$ Show that *AMBIG*CFG is undecidable. (Hint: Use a reduction from *PCP*. Given an instance

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

of the Post Correspondence Problem, construct a CFG G with the rules

$$\begin{aligned} S &\to T \mid B \\ T &\to t_1 T \mathtt{a}_1 \mid \cdots \mid t_k T \mathtt{a}_k \mid t_1 \mathtt{a}_1 \mid \cdots \mid t_k \mathtt{a}_k \\ B &\to b_1 B \mathtt{a}_1 \mid \cdots \mid b_k B \mathtt{a}_k \mid b_1 \mathtt{a}_1 \mid \cdots \mid b_k \mathtt{a}_k , \end{aligned}$$

where $a_1, \ldots, a_k$ are new terminal symbols. Prove that this reduction works.)

Step-by-step solution

Step 1 of 1

Un-decidability

1. If P has match with $t_{i1}t_{i2}\cdots t_{il} = b_{i1}b_{i2}\cdots b_{il}$ then it can be observed that string $t_{i1}t_{i2}\cdots t_{il}a_{il}\cdots a_{i2}a_{i1}$ has minimum two derivations, first from T and other one from B.

2. If the Context free grammar G is ambiguous, then some string s should have multiple derivations. As G generate s, s can be written as $wa_{j1}a_{j2}\cdots a_{jm}$ for some w that do not have symbols from $a_i's$.

After checking the grammar G, It can be observe that the derivation of B and derivation of T can each generate maximum one strings of same form as s. The multiple derivations of s as follows:

$$S \Rightarrow T \overset{*}{\Rightarrow} s = t_{jm}t_{jm-1}\cdots t_{j1}a_{j1}a_{j2}\cdots a_{jm}$$
$$S \Rightarrow B \overset{*}{\Rightarrow} s = b_{jm}b_{jm-1}\cdots b_{j1}a_{j1}a_{j2}\cdots a_{jm}$$

Thus, $t_{jm}t_{jm-1}\cdots t_{j1} = b_{jm}b_{jm-1}\cdots b_{j1}$

By combining (1) and (2), P has a match iff G is ambiguous.

So, the reduction from PCP to $AMBIG_{CFG}$ works. Thus, $AMBIG_{CFG}$ is un-decidable.

Comment

Problem

Show that A is Turing-recognizable iff $A \leq_m A_{TM}$.

Step-by-step solution

Step 1 of 4

Turing Machine:

• This is a type of mathematical model of computation that specify the conceptual machine.

• The Turing machine that operate a strings symbols on the tape as per given set of rules.

• The Turing machine can be manufactured in order to imitate the algorithms.

Comment

Step 2 of 4

Turing Recognizable:

• Any language will be recognizable or not if and only if it depends on Turing machine which will stop and receive only strings in that language and for strings not in the language.

• So, the Turing machine then refuse or does not stop at all.

Comment

Step 3 of 4

Theorem of Turing Recognizable:

• As per given in the theorem 5.28 from the book.

• If $A <=_m A_{TM}$ in this A_{TM} is Turing Recognizable then A is Turing Recognizable.

• A is Turing recognizable that can be prove below which is as follows:

Comment

Step 4 of 4

Consider the following details which is as follows:

A is Turing Recognizable iff $\mathbf{A <=_m A_{TM}}$.

• If $\mathbf{A <=_m A_{TM}}$ that means then A is Turing recognizable since $\mathbf{A_{TM}}$ is Turing recognizable.

• Suppose that A is Turing Recognizable then there will exists a Turing machine T that identify A.

• That means T gets the inputs which is w and receive if $w \in A$ that means w belongs to A.

• Apart from that T does not receive.

• In order to exhibit the $\mathbf{A <=_m A_{TM}}$.

• Describe a Turing machine that does the following works which is as follows:

• Take input w and write the (T, w) on the tape and stop.

• Then check the (T, w) is in $\mathbf{A_{TM}}$.

• The above condition can be checked if and only if w is in A .

• So, get the mapping reduction of $\mathbf{A}$ to $\mathbf{A}_{\mathbf{TM}}$.

• So, the reduction proofs that A is Turing Recognizable.

Hence, it is proved that A is Turing Recognizable.

Comments (4)

Problem

Show that A is decidable iff $A \leq_m 0^*1^*$.

Step-by-step solution

Step 1 of 3

Decidability

Assume $B = 0^*1^*$. Thus it is required to prove that A is decidable iff $A \leq_m B$.

Solution can be divided into two parts.

1: If A is decidable then $A \leq_m B$.

2: If $A \leq_m B$ then A is decidable.

Comment

Step 2 of 3

Part 1: If A is decidable then $A \leq_m B$.

Proof: Firstly define a function f as follows:

$$f(s) = 01 \text{ if } s \in A$$
$$f(s) = 10 \text{ otherwise.}$$

Since A is decidable, decider can be used for A to compute f. Also, $s \in A \text{ iff } f(s) \in B.$

Hence, f is mapping reduction from A to B.

Comment

Step 3 of 3

Part 2: If $A \leq_m B$, then A is decidable.

Proof: Since $A \leq_m B$, there exist a function f, such that $w \in A$ iff $f(w) \in B$.

Now consider Turing Machine M:

M= On input w

1. Compute $f(w)$

2. If $f(w)$ is in form of 0^*1^*, then accept, Otherwise, reject.

Now

$$\begin{aligned} &w \in A \\ &\Leftrightarrow f(w) \text{is of the form } 0^*1^* \\ &\Leftrightarrow M \text{ accept } w. \end{aligned}$$

Thus, M decides A.

Comment

Problem

Let J =

$$\{w \mid \text{either } w = 0x \text{ for some } x \in A_{\mathsf{TM}}, \text{ or } w = 1y \text{ for some } y \in \overline{A_{\mathsf{TM}}}\ \}.$$

is Turing-recognizable.

Step-by-step solution

Step 1 of 2

Turing-recognizable

Firstly demonstrate the reduction $f:\Sigma^* \to \Sigma^*$ of $\overline{A_{TM}}$ to J.

Assume a string $z \in \Sigma^*$. So that $f(z) = 1z$.

By definition of J, $z \in \overline{A_{TM}}$ iff $1z \in J$.

Hence f is a reduction of $\overline{A_{TM}}$ to J, Thus $\overline{A_{TM}} \leq_m J$.

By using the Corollary:

"If $\overline{A_{TM}} \leq_m B$, A is not a Turing-recognizable, then B is not Turing-recognizable."

Because $\overline{A_{TM}}$ is not Turing-recognizable, by Corollary J is not Turing-recognizable.

Comment

Step 2 of 2

Now demonstrate the reduction $f:\Sigma^* \to \Sigma^*$ of A_{TM} to J.

Assume a string $t \in \Sigma^*$. So that $g(t) = 0t$.

By definition of J, $t \in A_{TM}$ iff $0t \in J$.

Hence g is reduction of A_{TM} to J, Thus $A_{TM} \leq_m J$.

A function which reduces language L_1 to language L_2 also reduces $\overline{L_1}$ to language $\overline{L_2}$. Hence, g is reduction from $\overline{A_{TM}}$ to $\overline{J}$, Thus $\overline{A_{TM}} \leq_m \overline{J}$.

By using the Corollary:

"If $\overline{A_{TM}} \leq_m B$, A is not a Turing-recognizable, then B is not Turing-recognizable."

Because $\overline{A_{TM}}$ is not Turing-recognizable, by Corollary $\overline{J}$ is also not Turing-recognizable.

Therefore neither J nor $\overline{J}$ is Turing-recognizable.

Comment

Problem

Give an example of an undecidable language $B, \text{where } B \leq_{\mathrm{m}} \overline{B}.$

Step-by-step solution

Step 1 of 2

Undecidable language:

A language is an undecidable language, if it is not Turing-decidable. In other words, a language is undecidable language when there exists no Turing machine that can decide the language.

For example, let $B_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and accepts the input}'w'\}$ is undecidable.

Comments (1)

Step 2 of 2

Proof by contradiction:

Assume that B_{TM} is decidable.

Assume that the Turing machine A decides B_{TM}. So, the decidabilty of the Turing machine *A* is defined as:

$$A\langle M, w\rangle = \begin{cases} accept & if \ M \ accepts \ input \ w \\ reject & if \ M \ does \ not \ accept \ the \ input \ w \end{cases}$$

Using the Turing machine *A*, construct another Turing machine *X* that decides whether a machine *M* accepts its own encoding $\langle M\rangle$ is:

1. Input is $\langle M\rangle$, where M is some Turing machine.
2. Run *A* on $\langle M, \langle M\rangle\rangle$.
3. If *A* accepts the language, *reject*. Otherwise, *accept*.

So, the decidabilty of the Turing machine *X* is defined as:

$$X\langle M\rangle = \begin{cases} accept & if \ M \ does \ not \ accept \ \langle M\rangle \\ reject & if \ M \ accepts \ \langle M\rangle \end{cases}$$

The above specification cannot be satisfied by the machine. The Turing decidability of X on its own encoding $\langle X\rangle$ is:

$$X\langle X\rangle = \begin{cases} accept & if \ D \ does \ not \ accept \ \langle D\rangle \\ reject & if \ D \ accepts \ \langle D\rangle \end{cases}$$

Hence, neither *X* nor *A* can exist. That is, neither *X* nor *A* can accept the Turing machine *M*.

Thus, B_{TM} is undecidable.

Comments (7)

Problem

Define a ***two-headed finite automaton*** (2DFA) to be a deterministic finite automaton that has two read-only, bidirectional heads that start at the left-hand end of the input tape and can be independently controlled to move in either direction. The tape of a 2DFA is finite and is just large enough to contain the input plus two additional blank tape cells, one on the left-hand end and one on the right-hand end, that serve as delimiters. A 2DFA accepts its input by entering a special accept state. For example, a 2DFA can recognize the language $\{a^n b^n c^n | n \geq 0\}$.

a. Let $A_{\text{2DFA}} = \{\langle M, x\rangle |\ M \text{ is a 2DFA and } M \text{ accepts } x\}.$ Show that A_{2DFA} is decidable.

b. Let $E_{\text{2DFA}} = \{\langle M\rangle |\ M \text{ is a 2DFA and } L(M) = \emptyset\}.$ Show that E_{2DFA} is not decidable.

Step-by-step solution

Step 1 of 2

Decidability of 2DFA

Consider a Turing machine to check whether *M* the 2DFA accept the input *x*. if any configuration is repeated in M then it will not terminate because it is a deterministic finite automata.

Consider a Turing machine *W* which encodes M the 2DFA and input *x*. It also simulate M on *x* and check *M* accepts *x*. *W* has four tapes.

• Input tape to store input.

• Work tape, which has two bidirectional head to read.

• Another work tape to store configuration occurs during simulation.

• Scratch tape used to create representation of *M* configuration which becomes helpful for work tape in searching and updating.

W Turing machine works as follow:

$W = \text{on input}\langle M, x\rangle, \text{where } M \text{ is a 2DFA and } x \text{ is a string}$

1. Check the input tape $\langle M, x\rangle$ has a proper legal encoding or not. If it does not contain legal coding, then reject and halt, otherwise continue.

2. Copy input to work tape. Initialize the two head of work tape so that they are their starting position. Also initialize the second work tape as empty.

3. When *M* current state has halt then accepts and halt the states. When *M* current state has no move then reject and halt the states.

4. Create configuration on scratch tape. When current configuration already exist in the work tape then reject and halt otherwise store configuration at the end of second work tape.

5. Simulate one move of *M* on input tape.

6. Move to step 3.

When the input *x* are accepted by 2DFA that is M, the simulation will completed this in finite number of steps, then *W* will accept the input. Otherwise, when input codes are not legal or M does not ends or terminates. *W* determines this in finite number of steps and rejects the input.

All this shows that language A_{2DFA} is decidable.

Comment

Step 2 of 2

Assume on contrary that E_{2DFA} is decidable. Consider *W* a decider Turing machine which decides the E_{2DFA}.

Now, create Turing machine E which is based on *W* for deciding E_{TM} which works as follow:

$E = \text{on input}\langle M\rangle$

Create another 2DFA *M'*. The accepting computation history of *M* is recognizing by this *M'*.

Execute *W* on *M'*. When *W* accepts, **accepts**. Else **reject**.

Since, the E_{TM} is not decidable therefore the contradiction occurs. Hence E_{2DFA} is undecidable.

Problem

Next

A ***two-dimensional finite automaton*** (2DIM-DFA) is defined as follows. The input is an m × n rectangle, for any m, n ≥ 2. The squares along the boundary of the rectangle contain the symbol # and the internal squares contain symbols over the input alphabet Σ. The transition function $\delta\colon Q \times (\Sigma \cup \{\#\}) \longrightarrow Q \times \{\mathrm{L}, \mathrm{R}, \mathrm{U}, \mathrm{D}\}$ indicates the next state and the new head position (Left, Right, Up, Down). The machine accepts when it enters one of the designated accept states. It rejects if it tries to move off the input rectangle or if it never halts. Two such machines are equivalent if they accept the same rectangles. Consider the problem of determining whether two of these machines are equivalent. Formulate this problem as a language and show that it is undecidable.

Step-by-step solution

Step 1 of 2

To check **the un-decidability** of $E_{2DIM-DFA}$, first of all, it will have to $m-reduce$ A_{TM} to $E_{2DIM-DFA}$ by using a mapping which takes $\langle M,w\rangle$ **to** $\langle B\rangle$. It means that if w **is accepted by** M **then** $L(B)$ **is desired to be non-empty and when** w **is not accepted by** M **then** $L(B)$ **is desired to be empty.**

Comment

Step 2 of 2

This can be achieved by making $L(B)$ be the set of accepting, all the histories of M on w Here, B can be used as a rectangle (to represent the computation history of $C_1, C_2, \ldots, C_k$) with C_1 in the first row and C_2 in the next row and so on.

- For a given input rectangle, checking is performed. Here, B checks that the initial configuration of M on w is in first row or not and the last row is an accepting or final configuration .
- It also checked that each rows is followed from a previous row by the given rules of M .
- Now, if w is accepted by M , then there exists an accepting configuration history of M on w and $L(B)$ is not equals to $\varnothing$ or $L(B) \neq \varnothing$.
- In the same way, $L(B) = \varnothing$ and there is no accepting configuration history of M on w, when w is not accepted by M . Thus, $m-reduce$ A_{TM} to $\overline{E}_{2DIM-DFA}$ and $E_{2DIM-DFA}$ is un-decidable.

So, It follows that $EQ_{2DIM-DFA}$ **is un-decidable.**

Comment

Problem

Rice's theorem. Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turingmachine's language has property P is undecidable.

In more formal terms, let P be a language consisting of Turing machine descriptions where P fulfills two conditions. First, P is nontrivial—it contains some, but not all, TM descriptions. Second, P is a property of the TM's language whenever $L(M_1) = L(M_2)$, we have

$\langle M_1 \rangle \in P \text{ iff } \langle M_2 \rangle \in P.$ Here, M_1 and M_2 are any TMs. Prove that P is an undecidable language.

Step-by-step solution

Step 1 of 1

Rice's theorem

Given P non trivial property of language of Turing machine, it is required to prove that P is un-decidable. Consider on the contrary that P is decidable language that satisfies the properties. Consider R_P be a Turing machine that decides P. Now it is required to show that how to decide A_{TM} using R_P by constructing Turing machine S.

First, let T_θ be a Turing machine that always reject, so $L(T_\theta) = \phi$. It can be consider that $\langle T_\theta \rangle \notin P$ without loss of generality, because it is possible to proceed with $\overline{P}$ instead of P if $\langle T_\theta \rangle \in P$. Because P is non-trivial, there exist a Turing machine T with $\langle T \rangle \in P$. Now construct S based on T and R_P as follows:

$S ="$ On input $\langle \text{M,w} \rangle$:
1. Use M and w to construct the following Turing machine M_w.

$M_w ="$ On input x:
1. Simulate M on w. If it halts and rejects,reject.
2. Simulate T on x. If T accepts x, accept."

2. Use TM R_p to determine if $\langle M_w \rangle \in P$. If YES, accept,else reject."

Note that TM M_w has property that (1) if M accept w, $L(M_w) = L(T)$, and (2) if M does not accept w, $L(M) = \phi = L(T_\phi)$.

In other words, $\langle M_w \rangle \in P$ if and only if M accept w.

Since the construction of M_w from T, M and w takes finite steps, the TM S is decider for A_{TM}. This creates a contradiction since A_{TM} is an un-decidable language. In conclusion, P is un-decidable.

Comment

Problem

Show that both conditions in Problem 5.28 are necessary for proving that P is undecidable.

Problem 5.28

Rice's theorem. Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine's language has property P is undecidable.

In more formal terms, let P be a language consisting of Turing machine descriptions where P fulfills two conditions. First, P is nontrivial—it contains some, but not all, TM descriptions. Second, P is a property of the TM's language whenever $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$. Here, M_1 and M_2 are any TMs. Prove that P is an undecidable language.

Step-by-step solution

Step 1 of 1

Condition to show that some language un-decidable

First, consider P be the language $\{\langle M \rangle \mid M \text{ is a Turing Machine with 5 states}\}$. P is non-trivial, and so it satisfies the second condition of Rice's Theorem but P can be easily decided by checking the number of states of the input Turing Machine.

Second, consider P be the empty set. Then, it does not contain any Turing Machine and so, it satisfies the first condition of Rice's Theorem, but P can be decided by a Turing Machine that always rejects. Hence, both the properties are required for proving P un-decidable.

Comment

Problem

Use Rice's theorem, which appears in Problem 5.28, to prove the undecidability of each of the following languages.

A **a.** $INFINITE_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) \text{ is an infinite language}\}$.

b. $\{\langle M\rangle \mid M \text{ is a TM and } 1011 \in L(M)\}$.

c. $ALL_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \Sigma^*\}$.

Problem 5.28

Rice's theorem. Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine's language has property P is undecidable.

In more formal terms, let P be a language consisting of Turing machine descriptions where P fulfills two conditions. First, P is nontrivial—it contains some, but not all, TM descriptions. Second, P is a property of the TM's language whenever $L(M_1) = L(M_2)$, we have $\langle M_1\rangle \in P$ iff $\langle M_2\rangle \in P$. Here, M_1 and M_2 are any TMs. Prove that P is an undecidable language.

Step-by-step solution

Step 1 of 3

Un-decidability using Rice Theorem

a. $INFINITE_{TM}$ is a language of TM descriptions. It satisfies the two conditions of Rice's Theorem. First, is that it is non-trivial because some TMs have infinite languages and others have not. Second, is that it depends only on language. If two Turing Machine recognize same language, then either both should have descriptions in $INFINITE_{TM}$ or neither does. Consequently, Rice's theorem implies that $INFINITE_{TM}$ is un-decidable.

Comment

Step 2 of 3

b. Consider $\{\langle M\rangle \mid M \text{ is a Turing Machine and } 1011 \in L(M)\}$. P is language of Turing Machine descriptions. It satisfies the two conditions of Rice's Theorem. First, it is non-trivial because some TMs contain the string 1011 in their language and others do not. Second, it only depends on the language. If two TMs recognize same language then either both should have descriptions in P (because they both accept 1011), or neither do. Thus, Rice's theorem implies that P is un-decidable.

Comments (1)

Step 3 of 3

c. ALL_{TM} is a language of TM descriptions. It satisfies the two conditions of Rice's Theorem. First, it is non-trivial because some TMs accept all possible strings of an alphabet Σ and others do not. Second, is that it depends only on language. If two TMs recognize same language, then either both should have descriptions in ALL_{TM} or neither does. Therefore, Rice's theorem implies that ALL_{TM} is un-decidable.

Comment

Problem

Let

$$f(x) = \begin{cases} 3x + 1 & \text{for odd } x \\ x/2 & \text{for even } x \end{cases}$$

for any natural number x. If you start with an integer x and iterate f, you obtain a sequence, x, f(x), f(f(x)), ... Stop if you ever hit 1. For example, if x = 17, you get the sequence 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. Extensive computer tests have shown that every starting point between 1 and a large positive integer gives a sequence that ends in 1. But the question of whether all positive starting points end up at 1 is unsolved; it is called the 3_{x+1} problem.

Suppose that A_{TM} were decidable by a TM H. Use H to describe a TM that is guaranteed to state the answer to the 3x + 1 problem.

Step-by-step solution

Step 1 of 1

The solution can be built in stages. Later stages will use Turing machines that are constructed in earlier stages. The first step is to construct a Turing machine M_{query} that takes an input x and accepts if iterating f starting from x eventually yields 1 and loops forever otherwise.

M_{query}. On input $\langle x \rangle$:

1. If $x = 1$, then accept.
2. If x is odd, update $x \leftarrow 3x+1$. If x is even, update $x \leftarrow \frac{x}{2}$.
3. Go to step 1.

Our next step is to construct a Turing machine M_{loop} which iterates over all positive integers, looking for a counter-example to the $3x+1$ conjecture. That is, M_{loop} searches for some x such that iterating f starting from x never reaches 1 and accepts if it finds such an x. To do this, it seems tempting to have M_{loop} simulated M_{query} first on $x = 1$, next on $x = 2$, and so forth. Whenever iterating f on x yields 1, the simulation of M_{query} will eventually end and M_{loop} would then proceed to the next number $x+1$. But what if M_{loop} actually finds a counter-example x? In this case, the simulation of M_{query} on x will never terminate, and M_{loop} will be in the unfortunate situation that it has found what it is looking for, but it doesn't know it has found it!

To get around this, use H. Instead of simulating M_{query} on x, It has M_{loop} check whether or not M_{query} would accept x by passing $\langle M_{query}, x \rangle$, to H.

M_{loop}. On input $\langle w \rangle$:

1. Ignore the input w.
2. For each natural number $y = 1, 2, 3, \ldots$:
3. Run H on $\langle M_{\text{query}}, y \rangle$.
4. If H rejects, then accept. Otherwise, continue the loop.

Finally, in order to solve the $3x+1$ problem, it is required to know whether or not M_{loop} finds a counter-example. Again, it might be tempted to simulate M_{loop} and see if it ever finds a counter-example and accepts. The problem is that there may not be any counter-example, in which case M_{loop} will loop forever and our simulation will not terminate. The trick is to use H again to see if M_{loop} finds a counter example.

$M_{3x+1.}$ On input $\langle \text{w} \rangle$:

1. Ignore the input w.
2. Run H on $\langle M_{\text{loop}}, \varepsilon \rangle$.
3. If H accepts, then print ="There is a counter-example to the $3x+1$ conjecture."

Comment

Problem

Prove that the following two languages are undecidable.

a. $OVERLAP_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs where } L(G) \cap L(H) \neq \emptyset\}$. (Hint: Adapt the hint in Problem 5.21.)

b. $PREFIX\text{-}FREE_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG where } L(G) \text{ is prefix-free}\}$.

Step-by-step solution

Step 1 of 7

The given languages have to be proven to be un-decidable.

a)

$OVERLAP_{CFG} = \{\langle G,H\rangle \mid G \text{ and } H \text{ are CFGs where } L(G)\cap L(H)\neq\phi\}$

Comment

Step 2 of 7

Assume that $OVERLAP_{CFG}$ is decidable. Given an instance for the problem of Post Correspondence $P=\left\{\left[\frac{t_1}{b_1}\right],\left[\frac{t_2}{b_2}\right],\dots,\left[\frac{t_n}{b_n}\right]\right\}$, introduce unique new terminals $a_1,a_2,\dots,a_n$ for the CFGs.

Comment

Step 3 of 7

- Define the CFG G as:

$G=t_1,t_2,\dots,t_n$

$L(G)=\{s \mid s=t_it_j\dots t_ka_k\dots a_ja_i\}$

$S_G \to t_1S_Ga_1 \mid \dots \mid t_nS_Ga_n \mid t_1a_1 \mid \dots \mid t_na_n$

- Similarly, define the CFG H as follows:

$H=b_1,b_2,\dots,b_n$

$L(H)=\{s \mid s=b_ib_j\dots b_ka_k\dots a_ja_i\}$

$S_H \to b_1S_Ha_1 \mid \dots \mid b_nS_Ha_n \mid b_1a_1 \mid \dots \mid b_na_n$

As $L(G)\cap L(H)\neq\phi$, we get $t_it_j\dots t_ka_k\dots a_ja_i = b_ib_j\dots b_ka_k\dots a_ja_i$.

Comment

Step 4 of 7

- Since the new terminals $a_1,a_2,\dots,a_n$ are unique, which can be cancelled from both sides resulting in:

$t_it_j\dots t_k = b_ib_j\dots b_k$.

This is a way to solve for **the Post Correspondence Problem** P**.** This is a contradiction as the Post Correspondence **Problem is un-decidable**.

Therefore, the assumption taken that $OVERLAP_{CFG}$ **is decidable, is incorrect**.

Comment

Step 5 of 7

b)

$PREFIX\text{-}FREE_{CFG} = \{\langle G\rangle \mid G \text{ is a CFG where } L(G) \text{ is prefix-free}\}$

A mapping reducibility from the language A_{TM} (does a Turing machine accept a string?) to the language $PREFIX\text{-}FREE_{CFG}$ is given by the function f. Here, $\langle M,w\rangle$ is taken as an input of the computable functions f and $\langle M',w'\rangle$ is returned in such a way that:

$\langle M,w\rangle \in A_{TM}$ iff $\langle M',w'\rangle \in PREFIX\text{-}FREE_{CFG}$

Comment

Step 6 of 7

The machine F to compute the function f is:

F ="when $\langle M,w\rangle$ is taken as an input:

1. The machine M' is constructed

M' ="On input x:

1. For all proper prefixes y of x:

1. M will be run on y.

2. then *reject*.

2. Run M on x.

3. If it is accepted by M, then *accept*.

4. If it is rejected by M,, then *reject*."

2. Output $\langle M',w'\rangle$.

The output machine M' only accepts an input string w if the language $L(M)$ is prefix-free. It does so by checking if any of the proper prefixes of w do not lie in $L(M)$ and w lies in it.

Comment

Step 7 of 7

• It has been shown that A_{TM} **is mapping reducible to** $PREFIX\text{-FREE}_{CFG}$, that is:

$A_{TM} \le_m PREFIX\text{-}FREE_{CFG}$

Thus as A_{TM} **is un-decidable, the language** $PREFIX\text{-FREE}_{CFG}$ **is also un-decidable.**

Comment

Problem

Consider the problem of determining whether a PDA accepts some string of the form {wwl w ? {0,1}*} . Use the computation history method to show that this problem is undecidable.

Step-by-step solution

Step 1 of 1

When M does not accept w, then M_{new} will not go to left of $\$$. The Turing machine M_{new} writes something on the original input when M accept w.

Consider a Turing machine *T* having input *x*. Create a PDA which accepts string in $h\$h\$$ form when $x \in L(T)$.

The description history of *T* for *x* is the configuration sequence $C_0, C_1, C_2 \dots C_n$ where C_0 is the starting configuration of *T* in input *x*. The configuration C_n is the end or accepting configuration of *T* in input *x*.

Working of Turing machine:

1. At first PDA insert C_0 into stack and then confirms whether C_0, is the initial configuration in Turing machine T on input *w*.

2. At each odd configuration the PDA pops the last configuration from stack and then checks that the new configuration is the successor of last one or not.

3. When PDA read and first $ encountered, then it skips the C_0 configuration and insert the C_1 in the stack. At this time, it has confirmation that even configuration is the successor of odd configuration.

4. The PDA gets ACCEPTS when all check are true which means all new configuration is the successor of previous one and last configuration is the accepting configuration.

The PDA accepts the string ww if *x* is accepted by Turing machine *T*.

When *x* is not accepted by Turing machine *T* even then this PDA can accept the string in $y\$z\$$ form where $y \neq z$. This all happens because the construction depends on the different check whether input is in ww form or which does not happen in PDA to confirm that two passes check the same history.

Hence, PDA accepts string is un-decidable.

Comments (3)

Problem

Let X = { $\langle M, w \rangle |$ M is a single-tape TM that never modifies the portion of the tape that contains the input w}. Is X decidable? Prove your answer.

Step-by-step solution

Step 1 of 1

X is un-decidable which can be proof by reducibility. Reduce *X* from $D_{TM} = \{< M, w >| \text{Turing machine } M \text{ accepts } w\}$.

Assume *W* be a Turing machine which decides *X*. Use Turing machine *W* to create Turing machine *S* which decides D_{TM}.

1. TM *S*: on input $< M, w >$

2. TM M_{new}: on input $< M, w >$

3. The right end of the input mark with symbol $\$ \notin \Gamma_M$.

4. Copy the string which comes after \$. This parts of input denotes as w'

5. Simulate M on w'.

6. If M accepts, then write all strings written in first cell of input tape, **accept**

7. Else **reject.**

8. TM *W*: on input $< M_{new}, w >$

9. When *W* accepts, then **accept**, else **reject**.

When M does not accept w , then M_{new} will not go to left of \$. The Turing machine M_{new} writes something on the original input when M accept w
.

It is shown above that M_{new} modifies whenever *M* accepts the input *w*. Hence, D_{TM} has a contradiction.

Comments (3)

Problem

Say that a variable A in CFG G is ***necessary*** if it appears in every derivation of some string w ? G. Let *NECESSARY*$_{CFG}$ = { $\langle G, A \rangle |$ A is a necessary variable in G}.

a. Show that *NECESSARY*$_{CFG}$ is Turing-recognizable.

b. Show that *NECESSARY*$_{CFG}$ is undecidable.

Step-by-step solution

Step 1 of 2

Consider a CFG *G* which contains variables *A* in his each production of some string *w* which belongs to *G*.

Consider a Turing machine *D* which works as follow:

- $D = \text{On input} \langle G, A \rangle$, where G is CFG and A is non terminal

1. Create context free grammar $\frac{G}{A}$ by eliminating variable A from the derivations of *G*.

2. Create list of strings *w* generated by grammar G. Create a decider for A_{CFG} and then check each string of *w* can also be generated by $\frac{G}{A}$.

3. If *w* strings cannot generated by $\frac{G}{A}$ then **accept**, else **continue**.

• *D*= On other input instead of $\langle G, A \rangle$, **reject**

The $L\left(\frac{G}{A}\right)$ language contains all and only that strings of $w \in L(G)$ which does not require non terminal *A* for their derivation. If variable *A* is necessary for grammar *G* then few strings of $w \in L(G)$ cannot produce without use of non-terminal *A*.

The Turing machine *D* finds out those strings of *w* which cannot derived without use of *A*.

On the other hand, when variable A is not necessary for grammar G, which means $L(G) = L\left(\frac{G}{A}\right)$, then *D* continue move in a loop.

Hence, NECESSARY_{CFG} recognize by Turing machine *D*.

Comment

Step 2 of 2

It is already known that ALL_{CFG} is un-decidable, this shows that NECESSARY_{CFG} is also un-decidable. Therefore, NECESSARY_{CFG} must also un-decidable.

When any language complement is un-decidable then that language will also become un-decidable.

This can be proved by reduction R. The computation of reduction R is as follow:

$\text{R} = \text{On input} \langle G \rangle :$

• Create production of *G* after adding variable A and their productions in G.

$S \to A$

$A \to \in$

And

$A \to aA$ for each $a \in \Sigma$

• The output $\langle G, A \rangle$

The Grammar *G* produce with the help of reduction *R* is always $L(G) = \Sigma_*$. Thus, if $L(G) = \Sigma_*$, then there is no need of variable *A* in *G* because each string of $w \in \Sigma_*$ can derived by G without using *A*.

Also, if $L(G)=\Sigma_*$ then variable *A* is necessary for production of *G* because if $w \in L(G)$ then production of *G* is only possible with the help of *A*.

Thus, $\langle G\rangle \in ALL_{CFG}$, then $\langle G, A\rangle \in \text{NECESSARY}_{CFG}$

Hence, *R* reduces ALL_{CFG} to NECESSARY_{CFG}

Comments (6)

Problem

Say that a CFG is *minimal* if none of its rules can be removed without changing the language generated. Let MIN_{CFG} =

$$\{\langle G\rangle \mid G \text{ is a minimal CFG}\}.$$

a. Show that MIN_{CFG} is T-recognizable.

b. Show that MIN_{CFG} is undecidable.

Step-by-step solution

Step 1 of 6

a)

A minimal context-free grammar MIN_{CFG} is one in which no rule can be modified without changing the language generated. Now, it can Proved that MIN_{CFG} is T-recognizable.

• The grammar MIN_{CFG} can be converted to an equivalent grammar in Chomsky normal form.

• The generated CFG can be simulated by a Turing machine because for a string of length k there will be a finite number of derivations with $2k-1$ steps.

• This Turing machine U will accept the strings that are in the language.

Comment

Step 2 of 6

The Turing machine is:

U ="On input $\langle G, w\rangle$, where G is a minimal CFG and w is a string:

1. Create an equivalent grammar H in Chomsky normal form from G.

2. Let $n = |w|$.

3. If $n = 0$, then check all the derivations with one step in H, else check all the derivations with $2n-1$ steps.

4. *Accept* if any of the derivations accept w, else *reject*.

Comments (1)

Step 3 of 6

Therefore, **a minimal context-free grammar is T-recognizable.**

Comment

Step 4 of 6

b)

Consider the MIN_{CFG}. Now, it can be shown that MIN_{CFG} is un-decidable. It can be proved by contradiction, that is, by taking an assumption that MIN_{CFG} is decidable. Assume the opposite, which is the grammar MIN_{CFG} is decidable.

Comment

Step 5 of 6

It is known that if a language is decidable/ undecidable if and only if the complement of this language is decidable/ undecidable. Now consider ALL_{CFG}, which is already known as undecidable.

On input $\langle G \rangle$:

• Construction of G' takes place by adding to G a new terminal, together with productions: $G'S \rightarrow A, A \rightarrow \varepsilon, and\ A \rightarrow aA\ for\ each\ a \in \Sigma$

• Output $\langle G', A \rangle$.

Comment

Step 6 of 6

The construction of the grammar G' is takes place by f in such a way that it always show $L(G') = \Sigma^*$. Thus, if $L(G) = \Sigma^*$ then it is not necessary that A is for G'.

• The reason behind it is that every string $l = \Sigma^*$ can already be derived from G, which shows a contradiction.

Hence, from the above explanation it can be said that," MIN_{CFG} **is undecidable**".

Comment