Problem

Give an example in the spirit of the recursion theorem of a program in a real programming language (or a reasonable approximation thereof) that prints itself out.

Step-by-step solution

Step 1 of 2

Recursion theorem: Let T be a Turing machine that computes a function $t:\Sigma^*\times\Sigma^*\to\Sigma^*$. There is a Turing machine R that computes a function $r:\Sigma^*\to\Sigma^*$, where for every w,

$$r(w)=t\left(\langle R\rangle,w\right)$$

• The recursion theorem produces a new machine *R,* which operates exactly as *T* does.

• It has connection to the theory of self- reproducing system.

SELF = "On any input:

1. Obtain, via the recursion theorem, own description $\langle SELF\rangle$.

2. Print $\langle SELF\rangle$."

Comment

Step 2 of 2

• In the real programming Language, LISP plays the same role as a recursion theorem.

• The following program in LISP is an example in the spirit of recursion theorem.

```
(define self (lambda (x)
  (list x (list (quote quote) x))
)
(quote (lambda (x)
(list x (quote quote) x)
)
)
```

(quote x) represents in Lisp as , (lambda(x) (list x(quote quote) x)) is printing of word x is initialization. Whole program represent self printing.

Comment

Problem

Show that any infinite subset of MIN_{TM} is not Turing-recognizable.

Step-by-step solution

Step 1 of 3

Definition of MIN_{TM} =

If *M* is a Turing machine, then we say that the length of the description $\langle M \rangle$ of *M* is the number of symbols in the string description *M*.

Say that *M* is minimal if there is no Turing machine equivalent to *M* that has a shorter description. $\boxed{\text{MIN}_{TM} = \{\langle M \rangle \mid M \text{ is a minimal TM}\}}$

Comment

Step 2 of 3

Now we have to prove that any infinite subset of MIN_{TM} is not Turing – recognizable.

We will prove by taking contradiction.

We assume that there exists *A*, an infinite subset of MIN_{TM}, such that *A* is Turing – recognizable.

We know that

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

So, Let *E* be the enumerator that enumerates *A* .

By using this *E*, we construct another *TM* (Turing – machine) *N* as follows.

$N =$"On input *w*:

1. from recursion theorem. Own description $\langle c \rangle$ is obtained

2. Run the Enumerator *E* until a machine *P* is obtained with a longer

description than that of *N*.

3. Simulate *P* on input *w*".

Comment

Step 3 of 3

As we know that MIN_{TM} is infinite *A* is infinite subset of MIN_{TM}.

1. When *A* is infinite, *E*'s list must contain a *TM* with longer description than $N's$ description. So obviously *N* terminates with some *TM* P which is longer than *N*. Then *N* simulates *P* and so is equivalent to *P*.

2. It also notify that N is shorter than *P*. So *P* cannot be minimal. But *P* appears on the list that is produced by *E*.

3. *E's* list must contain a *TM* with longer description than *N*'s description.

From above three conditions we have a contradiction.

Thus our assumption that *A* is Turing – recognizable is wrong.

Therefore *A* is not Turing – recognizable.

Thus an infinite subset of MIN_{TM} is not Turing – recognizable.

Comment

Problem

Show that if A $\leq_T$ B and B $\leq_T$ C, then A $\leq_T$ C.

Step-by-step solution

Step 1 of 1

Definition of $A \leq_T B$:

Language *A* is Turing reducible to Language *B*, written as $A \leq_T B$, if *A* is decidable relative to *B*. That is the oracle for language *B* decides Language *A*.

Given that

- $A \leq_T B$

That is, Let the oracle M_1^B for Language *B* decides the Language *A*. and

- $B \leq_T C$

That is, Let the oracle M_2^C for Language *C* decides the Language *B*.

We have to prove that

- $A \leq_T C$

That is, there exists an oracle M_3^C for Language *C* which decides the Language *A*.

That means, machine M_3 have to simulate machine M_1.

We will explain this simulation in detail as follows

• Let M_1 queries an oracle about some String *x*.

• M_3 does not have an oracle for *B*. M_3

• So M_3 does not perform the test whether $x \in B$ or not directly.

• Thus M_3 first simulates M_2 on input *x* and get the result.

• Then M_3provides that answers to M_1.

• But the queries of machine M_2 are directly answered by M_3. Because M_3 and M_2 use same oracle *C*.

In this way the oracle for Language *C* decides Language *A*. That is $A \leq_T C$.

Thus If $A \leq_T B$ and $B \leq_T C$ then $A \leq_T C$.

Comment

Problem

Let

$$A_{\mathrm{TM}}' = \{\langle M, w\rangle \mid M \text{ is an oracle TM and } M^{A_{\mathrm{TM}}} \text{ accepts } w\}.$$ is undecidable relative to A_{TM}.

Step-by-step solution

Step 1 of 1

Given that

$$A_{TM}' = \{\langle M, w\rangle \mid M \text{ is an oracle TM (turing machine) and } M^{A_{TM}} \text{ accepts } w\}$$

We have to show that A_{TM}' is undecidable relative to A_{TM}.

Take a contradiction of A_{TM}' is decidable relative to A_{TM}.

Hence there exists an oracle *TM T* with oracle access to A_{TM} which decides A_{TM}'.

Now we construct another oracle *TM N* as follows:

N="on input

1. Run $T^{A_{TM}}$ on input >

2. If *T* accepts, *reject*.

3. Else if *T* rejects, *accept*".

• So $N^{A_{TM}}$ accepts if and only if *M* rejects.

• When the input of *N* is <*N*>, we have $N^{A_{TM}}$ accepts <*N*> if and only if *N* rejects <*N*>.

This is a contradiction to our hypothesis, that A_{TM}' is decidable relative to A_{TM} is wrong. Hence, A_{TM}' is un-decidable relative to A_{TM}.

Comment

Problem

Is the statement $\exists x \, \forall y \, [\, x+y=y \,]$

Step-by-step solution

Step 1 of 1

Theory of *N* :-

If *N* is a model, theory of *N*, written $Th(N)$, be the collection of true sentences in the language of that model.

• Given sentence is $\phi_1 = \exists x \forall y [x + y = y]$ and the given theory of model is $Th(N,+)$.

The statement ϕ_1 is true in model $(N,+)$.Because for $N = \{0,1,2,3...\}$ and for $x = 0$ the statement $0 + y = y$ is true.

So the statement $\exists x \forall y [x + y = y]$ is a member of $Th(N,+)$

• Given statement is $\phi_2 = \exists x \forall y [x + y = x]$

This statement ϕ_2 is false in the model $(N,+)$

Because for any *x* the statement $x + y = x$ is false, for all y (except if $y = 0$).

So the statement ϕ_2 is false in the model $(N,+)$

Hence the statement $\exists x \forall y [x + y = x]$ is not a member of $Th(N,+)$

Comment

Problem

Describe two different Turing machines, M and N, where M outputs $\langle N \rangle$, when started on any input.

Step-by-step solution

Step 1 of 2

Given: There are two different Turing machines *M* and *N.*

We have to prove that, it started with any input *w*, *M* outputs $\langle N \rangle$ and *N* outputs $\langle M \rangle$

Here we need to know the recursion theorem.

Recursion theorem: Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. There is a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every w,

$r(w) = t(\langle R \rangle, w)$.

To make a Turing machine that can get its own description and computes with it. Machine T in the statement receives the description of machine as extra input. The recursion theorem produces a new machine R, which operates exactly T, does but R's description filled automatically.

Comment

Step 2 of 2

By using Recursion theorem, remember that $r(w)$ is a Turing machine that prints *w* on its tape, and halts.

The Turing machine *M* is as follows:

$M =$" on input *w*:

1. No need to note the input

2. Obtain $\langle M \rangle$ by using the recursion theorem.

3. Compute $r(\langle M \rangle)$ where r the computable function is write

$r(\langle M \rangle)$ on the tape.

4. Halt."

• When *M* runs, it writes $N = \langle r(\langle M \rangle) \rangle$ in its tape.

• When *N* runs, it writes $\langle M \rangle$ in its tape automatically.

$\langle N \rangle$ is a member of the turing machine M

$\langle M \rangle$ is a member of the turing machine N

Therefore *M* outputs $\langle N \rangle$ and *N* outputs $\langle M \rangle$.

Comment

Problem

In the fixed-point version of the recursion theorem (Theorem 6.8), let the transformation t be a function that interchanges the states q_{accept} and q_{reject} in Turing machine descriptions. Give an example of a fixed point for t.

THEOREM 6.8

Let $t\colon \Sigma^* \longrightarrow \Sigma^*$ be a computable function. Then there is a Turing machine F for which $t(\langle F\rangle)$ describes a Turing machine equivalent to F. Here we'll assume that if a string isn't a proper Turing machine encoding, it describes a Turing machine that always rejects immediately.

In this theorem, t plays the role of the transformation, and F is the fixed point.

Step-by-step solution

Step 1 of 4

Fixed point of a function is value that is not changed by application of the function. We need to consider functions are computable transformations of Turing machine descriptions. Turing machine behavior is unchanged by transformations that theorem called fixed-point version of recursion theorem.

Fixed – point version of the recursion theorem:- Let $t:\Sigma^* \to \Sigma^*$ be a computable function then there is a Turing machine *F* for which $t(\langle F\rangle)$ describes a Turing machine equivalent to *F*. If a string is not proper Turing machine encoding, then it describes Turing machine rejects immediately.

t: is a role of transformation

F: fixed point

Comment

Step 2 of 4

Given that,

The transformation *t* is a function that interchanges the states q_{accept} and q_{reject} in Turing machine descriptions.

Now we have to define the fixed point for *t*.

Comment

Step 3 of 4

Let $\langle M\rangle$ be a fixed point machine.

(i) *M* Halts on input *w*

• If *M* halts on any input *w*, then clearly *M* cannot be a fixed point since q_{accept} and q_{reject} are replacing.

• The Language recognized by $\langle M\rangle$ and the Language recognized by $t\langle M\rangle$ must differ in *w*.

• So *M* does not halt on any input, instead of that it rejects then only *M* accepts as fixed point.

(ii)

• Suppose *M* loops infinitely on all inputs.

• Then $(t\langle M\rangle)$ must also loop infinitely, because it is the same machine but with accept and reject states swapped.

• Therefore *M* is a fixed – point.

Hence any machine which loops infinitely on all inputs is a fixed – point since it rejects.

Comment

Step 4 of 4

Example: A machine which can never leave the start state is an example of fixed point.

Start with *n*-state Turing machine start with blank tape but the productivity is zero.

Machine *M* on blank input

• Get description for $\langle M \rangle$ from recursion theorem for *n* states of $\langle M \rangle$

• Compute Productivity *p*(*n*)

• Add 1 to obtain *p*(*n*) + 1

• Once end of end of tape print *p*(*n*) + 1

• But *M* is an *n*-state Turing machine that outputs more than the productivity function. It is a contradiction.

Comment

Problem

Show that $EQ_{\mathrm{TM}} \not\leq_{\mathrm{m}} \overline{EQ_{\mathrm{TM}}}$.

Step-by-step solution

Step 1 of 3

TM equality:

The TM equality is represented as follows:

$EQ_{TM} = \{(\langle M \rangle, \langle N \rangle) \text{ where } M \text{ and } N \text{ are Turing machines and } L(M) = L(N)\}$.

$EQ'_{TM} = \{(\langle M' \rangle, \langle N' \rangle) \text{ where } M' \text{ and } N' \text{ are Turing machines and } L(M') = L(N')\}$

$EQ_{TM} \not\leq_m \overline{EQ_{TM}}$ means that EQ_{TM} is not mapping reducible to $\overline{EQ_{TM}}$. This means that EQ_{TM} is not mapping reducible to its complement.

Comment

Step 2 of 3

Proof:

In order prove that $EQ_{TM} \not\leq_m \overline{EQ_{TM}}$, first prove that EQ_{TM} is not Turing-recognizable.

According to Theorem 5.28 and Corollary 5.29, $A \leq_m B$ only if both A and B are Turing recognizable or not Turing recognizable.

$\overline{EQ_{TM}}$ is complement of EQ_{TM}. So, if EQ_{TM} is not Turing-recognizable then, $\overline{EQ_{TM}}$ is Turing-recognizable and vice-versa. This, result in not mapping reducibility between EQ_{TM} and $\overline{EQ_{TM}}$.

Comment

Step 3 of 3

Example:

Assume A_{TM} is a Turing machine and is mapping is mapping reducible to $\overline{EQ_{TM}}$ that is $A_{TM} \leq_m \overline{EQ_{TM}}$.

The function $f_2 : A_{TM} \to \overline{EQ_{TM}}$ is defined as follows:

$f_2 : On\ input\ \langle M, w \rangle$
$Construct\ machine\ M_3 : on\ any\ input,\ reject.$
$Construct\ machine\ M_4 : on\ any\ input\ x, run\ M\ on\ w.$
$If\ it\ accepts,\ accept\ x.$
$Output\ \langle M_3, M_4 \rangle$

Explanation:

• The machine M_1 accepts nothing.

• If M accepts w, then M_2 accepts everything. Otherwise it accepts nothing.

• So, $\langle M, w \rangle \in A_{TM} \Leftrightarrow \langle M_3, M_4 \rangle \in \overline{EQ_{TM}}$ and f_2 is clearly computable. Thus, it is a reduction from A_{TM} to $\overline{EQ_{TM}}$.

• So, EQ_{TM} is not Turing-recognizable.

Thus, if EQ_{TM} is not Turing-recognizable and $\overline{EQ_{TM}}$ is Turing-recognizable then EQ_{TM} is not mapping reducible to $\overline{EQ_{TM}}$. That is, $EQ_{TM} \not\leq_m \overline{EQ_{TM}}$.

Problem

Use the recursion theorem to give an alternative proof of Rice's theorem in Problem 5.28.

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 2

Rice's theorem: Let *P* be any nontrivial property of the language of a Turing machine.

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.

Then we have to prove that *P* is an undecidable Language.

Assume that *P* is a decidable Language satisfying the Properties of Rice theorem.

Comment

Step 2 of 2

Let the Turing machine *T* decides the language *P*. According to the first condition of Rice theorem, *P* contains some, but not all, *TM* descriptions.

From above properties we assume $TM_1 \& TM_2$ be the Turing machines and $\langle TM_1 \rangle \in P$ and $\langle TM_2 \rangle \notin P$.

Now construct the Turing machine TM_3 using $TM_1 \& TM_2$.

TM_3"on input *w*:

1. By using recursion theorem, construct our own description for $\langle TM_3 \rangle$

2. Run *T* on Turing machine $\langle TM_3 \rangle$

3. If *T* accepts $\langle TM_3 \rangle$, simulate $\langle TM_2 \rangle$ on *w*.

If *T* rejects $\langle TM_3 \rangle$, simulate $\langle TM_1 \rangle$ on *w*.

• Suppose $\langle TM_3 \rangle \in P$

Then *T* accepts $\langle TM_3 \rangle$ and $L(TM_3) = L(TM_2)$

According to second condition of Rice's theorem $\langle TM_2 \rangle \in P$ but our condition is $\langle TM_2 \rangle \notin P$. It is a contradiction here.

• So, we will get similar contradiction if $\langle TM_3 \rangle \notin P$.

As a result of that our assumption is wrong for *P* is a decidable language. Hence every property satisfying the conditions of Rice's theorem is undecidable.

Comment

Problem

Give a model of the sentence

$$\begin{aligned}\phi_{\text{eq}} \;=\; & \forall x \left[R_1(x,x) \right] \\ & \wedge \forall x,y \left[R_1(x,y) \leftrightarrow R_1(y,x) \right] \\ & \wedge \forall x,y,z \left[(R_1(x,y) \wedge R_1(y,z)) \rightarrow R_1(x,z) \right].\end{aligned}$$

Step-by-step solution

Step 1 of 5

Given sentence is

$$\begin{aligned}\phi_{eq} &= \forall x\left[R_1(x,x)\right] \\ &\wedge \forall x,y\left[R_1(x,y) \leftrightarrow R_1(y,x)\right] \\ &\wedge \forall x,y,z\left[\left(R_1(x,y)\right) \wedge R_1(y,z) \rightarrow R_1(x,z)\right]\end{aligned}$$

ϕ_{eq} gives three conditions of equivalence relations i.e., Reflexive relation, Symmetric relation and Transitive relations.

Comment

Step 2 of 5

Let R_{11} and R_{12} are two equivalence relations on some set, definition of R_1 by

$$\forall x,y\left[R_1(x,y) \equiv R_{11}(x,y) \wedge R_{12}(x,y)\right]$$

Where *x, y* are elements from set.

Reflexive relation: $\forall x\left[R_1(x,x)\right]$

$\equiv$ {definition of R_1}

$R_{11}(x,x) \wedge R_{12}(x,x)$

$\equiv$ {R_{11}, being an equivalence relation, is reflexive}

similarly R_{12}

$\equiv \{true\} \wedge \{true\}$

$\equiv true$

Comment

Step 3 of 5

Symmetric relation: $\forall x,y\left[R_1(x,y) \leftrightarrow R_1(y,x)\right]$

$\equiv$ {definition of R_1}

$R_{11}(x,y) \wedge R_{12}(x,y)$

$\equiv$ {R_{11}, being an equivalence relation, is symmetric}

similarly R_{12}

$\equiv$ {definition of R_1}

$R_1(y,x)$

Comment

Step 4 of 5

Transitive relation: $\forall x,y,z\left[\left(R_1(x,y)\right)\wedge R_1(y,z)\rightarrow R_1(x,z)\right]$

$\equiv$ {definition of R_1}

$\left(R_{11}(x,y)\wedge R_{12}(x,y)\right)\wedge\left(R_{11}(y,z)\wedge R_{12}(y,z)\right)$

$\equiv$ {rearranging the conjucts}

$\left(R_{11}(x,y)\wedge R_{11}(y,z)\right)\wedge\left(R_{12}(x,y)\wedge R_{12}(y,z)\right)$

$\equiv$ {R_{11}, being an equivalence relation, is transitive}

similarly R_{12}

$\equiv$ {definition of R_1}

$R_1(x,z)$

Comment

Step 5 of 5

A model (U,R_1), where *U* is universe and R_1 is equivalence relation over *U,* is a model of ϕ_{eq}.

Comment

Problem

Let ϕ_{eq} be defined as in Problem 6.10. Give a model of the sentence

$$\begin{aligned}\phi_{\text{lt}} \;=\; & \phi_{\text{eq}} \\ & \wedge \forall x,y \left[R_1(x,y) \rightarrow \neg R_2(x,y) \right] \\ & \wedge \forall x,y \left[\neg R_1(x,y) \rightarrow (R_2(x,y) \oplus R_2(y,x)) \right] \\ & \wedge \forall x,y,z \left[(R_2(x,y) \wedge R_2(y,z)) \rightarrow R_2(x,z) \right] \\ & \wedge \forall x \, \exists y \left[R_2(x,y) \right].\end{aligned}$$

Give a model of the sentence

$$\begin{aligned}\phi_{\text{eq}} \;=\; & \forall x \left[R_1(x,x) \right] \\ & \wedge \forall x,y \left[R_1(x,y) \leftrightarrow R_1(y,x) \right] \\ & \wedge \forall x,y,z \left[(R_1(x,y) \wedge R_1(y,z)) \rightarrow R_1(x,z) \right].\end{aligned}$$

Step-by-step solution

Step 1 of 2

Consider the following sentence that is provided in the problem 6.10 in the textbook,

$$\begin{aligned}\phi_{eq} = & \forall x \, [R_1(x,x)] \\ & \wedge \forall x,y \, [R_1(x,y) \leftrightarrow R_1(y,x)] \\ & \wedge \forall x,y,z \, [(R_1(x,y) \wedge R_1(y,z)) \rightarrow R_1(x,z)]\end{aligned}$$

The above statement explains about the conditions of the equivalence relation. A model (A, R_1), where *A* is any universe and R_1 is the equivalence relation over the elements of *A*. The line 1 describes that for all *x*, *x* is equal to itself. The line 2 describes that for all *x* and *y*, if and only if *x* is equal to *y* then *y* is equal to *x*. The line 3 describes that for all *x*, *y*, and *z*, if *x* is equal to *y* and *y* is equal to *z* then *x* is equal to *z*.

Comment

Step 2 of 2

Consider the following sentence,

$$\begin{aligned}\phi_{lt} = & \phi_{eq} \\ & \wedge \forall x,y \, [R_1(x,y) \rightarrow \neg R_2(x,y)] \\ & \wedge \forall x,y \, [\neg R_1(x,y) \rightarrow (R_2(x,y) \oplus R_2(y,x))] \\ & \wedge \forall x,y,z \, [(R_2(x,y) \wedge R_2(y,z)) \rightarrow R_2(x,z)] \\ & \wedge \forall x \exists y \, [R_2(x,y)]\end{aligned}$$

The above statement explains about the conditions of the equivalence relation and less than relation. A model (A, R_1, R_2), where *A* is any universe, R_1 is the equivalence relation over the elements of *A* and R_2 is the less than relation over *A*.

• The line 1 describes the conditions of the equivalence relation.

• The line 2 describes that for all x and y, if x is equal to y then it is tends to complement of less than relation. This means if x is equal to y then it can be said that x is not less than y.

• The line 3 describes that for all x and y, if x is not equal to y then either x less than y or y less than x is tr

• The line 4 describes that for all x, y and z, if x is less than y and y is less than z then x is less than z.

• The line 5 describes that for all x there exist y, x is less than y.

Comment

Problem

Let $(\mathcal{N}, <)$ is decidable.

Step-by-step solution

Step 1 of 2

Consider theorem 6.12 given in the textbook where it is proved that Th(◊, +) is decidable where + refers to the relation. As decidability has been proved for Th(◊,+), reduce Th(◊,+) to Th(◊,<).

The decidability of Th(◊, <) can be proved by converting sentence S_1 of language Th(◊,<) into S_2 sentence preserving all the truth or falsity related to models.

Comment

Step 2 of 2

Now, replace the occurrences of S_1 in S_2 where $\wedge \text{and} \vee$ with formula:

$\exists k[(i+k=j)\wedge(k+k\neq k)]$

k refers to new variable which will be different for every case.

S_2 is equivalent to S_1 because the value of *i* is less than the value of *j*. So, the value of *j* can be obtained by adding non-zero value in *i*. "i is less than *j*" that means '*j*' can be obtained by adding non zero value to '*i*'.

To prove the decidability of Th(◊,+) S_2 is supposed to be put in prenex-normal form.

For bringing existential qualifiers to front of the sentence, quantifiers are supposed to pass through Boolean operations that are being appeared in the sentence.

When the quantifiers are brought, the operations ^ and V will not be changed. When a null is brought it changes to and vice-versa. So, $\neg\exists k\varphi$ becomes $\forall k\neg\varphi$ and $\neg\forall k\varphi$ becomes $\exists k\neg\varphi$.

So, Th(◊,<) is decidable.

Comment

Problem

For each

$$m > 1 \text{ let } \mathcal{Z}_m = \{0, 1, 2, \ldots, m-1\}, \text{ and let } \mathcal{F}_m = (\mathcal{Z}_m, +, \times)$$

is decidable.

Step-by-step solution

Step 1 of 1

Given that

- $z_m = \{0,1,2,...,m-1\}$ for $m > 1$ is a universe.
- $F_m = (z_m, +, \times)$ be the model whose universe is z_m.
- $+$ and $\times$ are relations over modulo *m*.

Now we have to show that $Th(F_m)$ is decidable.

Clearly z_m is finite.

So we can simply enumerate all the possible values into the formula and see if the formula is true.

If it is true then it is belong to $Th(F_m)$

If it is false then it does not belong to $Th(F_m)$

In particular, the following recursive procedure would work:

• Let $\exists x_i$ such that $\phi_i(x_1, x_2, ...x_i)$ be the formula

• Substitute $x_i = 0,1,...,m-1$ into the formula.

• If the formula $\phi_i(x_1, x_2, ...x_i)$ is true for any value of x_i then the original formula is true

• That means, if we have $\forall x_i$ instead of $\exists x_i$ then the original formula will be true.

→ A theory of a model is said to be decidable if we say that the formulae which are true are belong to that model and which are false that does not belong to that model.

So by the above recursive procedure we say that $Th(F_m)$ is decidable.

Comment

Problem

Show that for any two languages A and B, a language J exists, where $A \leq_T J$ and $B \leq_T J$.

Step-by-step solution

Step 1 of 1

For any two languages A and B, a language J exists, where $A \leq_T J$ and $B \leq_T J$. It can be proved as follows:

Two languages A and B are given and one more language J exists to find Turing reducibility between A and J and between B and J. it can be proved by using the concept of mapping reducibility.

$A \leq_T J$ means that A is Turing reducible to language J. $B \leq_T J$ means that B is Turing reducible to J. Now, use the concept of decidability. Here J is an intermediate language that is being used to prove the Turing reducibility of A and B. So prove first of all decidability of J must be proved.

Decidability of J can be proved by the given relation $B \leq_T J$ and $A \leq_T J$ if A and B are decidable then J will also be decidable. Replace the oracle Turing machine that decides A by an ordinary Turing Machine that decides A

Turing reducibility is the generalized form of mapping reducibility. Mapping reducibility holds true only if decidability holds true because the mapping reducibility may be used to give an oracle Turing Machine that decide Turing reducibility of A and B on the Basis of J.

Comments (8)

Problem

Show that for any language A, a language B exists, where $A \leq_{\mathrm{T}} B$ and $B \not\leq_{\mathrm{T}} A.$

Step-by-step solution

Step 1 of 3

Given that

A and *B* are two Languages.

We have to show that

$A \leq_T B$ and $B \not\leq_T A$.

That means

• *A* is Turing reducible to *B*.

But *B* is not Turing reducible to *A*.

Let $A = E_{TM}$ (Empty Turing machine)

$= \{\langle M \rangle | \text{ M is a TM and } \mathrm{L}(\mathrm{M}) = \phi\}$

And $B = A_{TM}$ (Oracle Turing machine)

$= \{\langle M, w \rangle, \text{ machine } M \text{ accepts } w\}$

Comment

Step 2 of 3

(i) $A \leq_T B$:

To show $A \leq_T B$, we have to show that $E_{TM} \leq_T A_{TM}$.

We know that E_{TM} is decidable relative to A_{TM} .

Since $E_{TM} \leq_T A_{TM}$, then *A* is decidable relative to *B* then $A \leq_T B$.

Comments (1)

Step 3 of 3

(ii) $B \not\leq_T A$

To shown $B \not\leq_T A$ we have to show that

$A_{TM} \not\leq_T E_{TM}$.

Let us assume the contradiction $A_{TM} \leq_T E_{TM}$.

By mapping reducibility rules for any two languages $x \leq_m y \Leftrightarrow \overline{x} \leq_m \overline{y}$

Language x is mapping reducible to language y, written $x \leq_m y$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every *w, f* called reduction of *x* to *y*.

$w \in x \Leftrightarrow f(w) \in y$

According to given mapping reducibility we have to derive $A_{TM} \leq_T E_{TM} \Leftrightarrow \overline{A_{TM}} \leq_m \overline{E}_{TM}$

However $\overline{E}_{TM}$ is Turing recognizable, but $\overline{A}_{TM}$ is not Turing recognizable

Accor ¨ ıg mapping reducibility rules

$x \leq_m y$ and *y* is Turing – recognizable then *x* must be Turing recognizable.

Therefore this gives a contradiction.

Therefore our assumption that $A_{TM} \leq_T E_{TM}$ is wrong.

Hence $A_{TM} \not\leq_T E_{TM}$ i.e., $B \not\leq_T A$.

From (i) and (ii) we have showed that there exist two languages *A* and *B* such that

$A \leq_T B$ But $B \not\leq_T A$.

Comments (1)

Problem

Prove that there exist two languages A and B that are Turing-incomparable—that is, where $A \not\leq_T B$ and $B \not\leq_T A$.

Step-by-step solution

Step 1 of 1

Given:

In this problem two languages A and B are present.

Proof:

Two languages A and B are Turing incomparable if the language is not Turing comparable with language B and it's vice versa.

Languages Incomparability can be proved as follows:

• In the Previous question there was one more language J was given for proving Turing reducibility of 2 languages A and B.

• If it comes to previous situation there was one more language J exists and it is being used to decide the Turing reducibility of two languages A and B.

• But if it comes to discuss about current situation then no other language or can say oracle support with Turing machine or language is given to detect Turing Comparability.

• Here no Turing machine is given that will work as an oracle Turing machine to find decidability and Turing reducibility.

• So none language from A and B can be replaced by any intermediate language and further no oracle Turing machine can be produced.

• For producing any Turing machine at least one intermediate language is required.

• So both languages can be compared through intermediate languages and Turing reducibility can be found.

• Oracle Turing machine set is countable that is why it is required to find the Turing reducibility and on the other hand set of all languages are uncountable.

• Most important thing is still missing that Turing reducibility cannot be established until decidability is proved as Turing reducibility is generalized concept of decidability.

• If discuss about further concept that how to find Turing comparability then machine must be added up with oracle support.

Conclusion:

No solution or method is defined here to prove the decidability of 2 languages so languages A and B are not Turing reducible.

Hence it is clear that language A and B is Turing incomparable.

Comment

Problem

Let A and B be two disjoint languages. Say that language C *separates* A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Describe two disjoint Turing-recognizable languages that aren't separable by any decidable language.

Step-by-step solution

Step 1 of 2

Let A and B are two disjoint languages. Consider that language C, that separates A and B if $A \subseteq CandB \subseteq \overline{C}$. It can be proved as follows:

• It is quite easy to understand this statement. Read the statement carefully things are pretty obvious all you need to understand is $A \subseteq CandB \subseteq \overline{C}$.

• Here, A **is a set of** C **and** B **is a set of Complement of** C. So, it is quite obvious now A and B both are not related to each other anyhow.

• Here, no use of C because even if we use C it won't be able to prove decidability of A on the basis of B and Turing reducibility of A on the basis of B.If it comes to languages those are fully different and belong to different sets then no separators are required. So, here it is worthless to use C as separator.

Comment

Step 2 of 2

Now, consider a Turing machine T that will work as a decider for the language C that separates A and B.**Consider both languages as Regular Expressions that will be decided by** M_1 **and** M_2**.**

1. $S = <M, w>$ Where M is a Turing machine.

2. Now, run $< M_1, W >$ and $< M_2, W >$

3. If M_1 accepts then M rejects and if M_2 accepts M rejects.

• **Remember** M **will always halt in each situation. Where C decides A or B. Now it is pretty easy to understand the situation**.

• Therefore, it can be said that only the first statement of question is enough to prove the concept" **No decidable languages can be used to separate two disjoint Turing recognizable languages**".

Comment

Problem

Show that $\overline{EQ_{\mathrm{TM}}}$ is recognizable by a Turing machine with an oracle for A_{TM}.

Step-by-step solution

Step 1 of 1

Given:

A language that is recognizable by an oracle Turing machine.

Construction:

Consider Turing Machine T with an oracle for A_{TM} and consider 2 more Turing machines P and Q that will work as decider for T and then T will work as a decider for A_{TM} and further A_{TM} will work as a Turing recognizable for $\overline{EQ_{TM}}$. Use the following approach:

$(P,w) \in A_{TM} \longleftrightarrow T(< M, w >) \in EQ_{TM}$

Above statement can be applied by using following algorithm:

T = on input $\langle P,w \rangle$

Where w is an input string and running on P

• Construct 2 Machines P and Q

P = On any input: accept.

Q = On any input: run P on w, if it accepts, accept

• Output $\langle P,Q \rangle$

Conclusion:

Here, it is quite obvious that T is working as a decider for A_{TM} and then A_{TM} for $\overline{EQ_{TM}}$. This way $\overline{EQ_{TM}}$ is recognizable by a Turing Machine with an oracle for A_{TM}.

Comment

Problem

In Corollary 4.18, we showed that the set of all languages is uncountable. Use this result to prove that languages exist that are not recognizable by an oracle Turing machine with an oracle for A_{TM}.

COROLLARY **4.18**

Some languages are not Turing-recognizable.

Step-by-step solution

Step 1 of 1

Given:

Group or set of all languages are not countable.

Proof:

From the corollary 4.18 of text book it is clear that the group of legal encodings of a Turing machine is countable with A_{TM} accessed by oracle. Use same argument for proving that the set of all Turing machines is countable.

In other words it can be said that Turing machine with oracle is countable but on the other hands decidability, reducibility and recognizability cannot be proved without oracle support.

But the set of all languages are not countable. Any language D can be mapped to binary vector corresponding of their characteristic vector. This binary vector is infinite in nature and composed of two sequences which are 0's and 1's. The characteristic vector is represented as χ_D .

So group of all languages cannot be placed in correspondence to group of all Turing machines.

Conclusion:

Sometimes some set of all languages cannot be put into a correspondence with the set of all Turing Machines with oracle support. As the set of all languages cannot be kept into set of all Turing machines, hence some languages are not recognizable by using Turing machine oracle access to A_{TM} .

Comments (1)

Problem

Recall the Post Correspondence Problem that we defined in Section 5.2 and its associated language *PCP*. Show that *PCP* is decidable relative to A_{TM}.

Step-by-step solution

Step 1 of 1

Given:

Here, A_{TM} is with associated with oracle concept.

Proof:

PCP (Post Correspondence Problem) is decidable relative to A_{TM} . Here, Post Correspondence Problem can be considered as an example of un-decidability problem concerning with manipulation of strings to find a match. A match can be found, if String made by combining all symbols of upper side and string made by combing all symbols of lower side, both is same.

Consider a contrary that PCP is un-decidable relative to A_{TM} but it is not associated with oracle concept.

The concept of oracle associated with A_{TM} provides flexibility or countable property that is not provided by other Turing machines that are not associated with oracle.

Now, it is clear that first found a match to find PCP. It is considered that PCP is un-decidable relative to A_{TM} but it is not associated with oracle concept. Here, it is quite obvious if terminology of Turing machine is not associated with concept of oracle then it will be un-decidable rather PCP (Post Correspondence Problem) is decidable relative to A_{TM} . Here A_{TM} is with associated with oracle concept

Conclusion:

Hence, PCP is decidable with respect to A_{TM} .

Comment

Problem

Show how to compute the descriptive complexity of strings K(x) with an oracle for A_{TM}.

Step-by-step solution

Step 1 of 2

Descriptive complexity of strings:-

If *x* be binary string, then the minimal description and descriptive complexity of *x*'s are $d(x)$ and $\mathrm{K}(x)$ respectively. Turing machine *M* and small string *w* we get minimal description is $\langle M, w\rangle$. From several of such shorter strings we select lexicographically among them then we can get descriptive complexity of such strings $\mathrm{K}(x)=|d(x)|$.

Comment

Step 2 of 2

Now we have to show how to compute $\mathrm{K}(x)$ with an oracle for A_{TM}.

• For the given string *x*, start testing all the strings '*S*' up to the length $|x|+c$

Where $c=$ length of *TM* (Turing machine) that halts immediately upon starting.

• All the strings up to the length $|x|+c$ are potential description of *x*.

• If *S* is well formed as $\langle M, w\rangle$ from all binary strings in lexicographic order, then we simulate *M* with input *w* and see if it halts with *x* on the tape.

• Here we do not know whether *M* will halt on input *w* or not.

• An oracle for A_{TM} can determine this.

$$A_{TM}=\{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

• An oracle for A_{TM} will take $\langle M, w\rangle$ as input and determine whether *M* accepts *w* or not.

• If *M* doesn't halt we move on to the next string *S*, and so on.

• After that we will find lexicographically first string *S* among them.

• In this way shortest string will be determined and it is represented as minimal description $d(x)$.

• From $d(x)$, we find $\mathrm{K}(x)$ as

$$\mathrm{K}(x)=|d(x)|$$

• By this procedure, we will compute $\mathrm{K}(x)$ with an oracle for A_{TM}.

Comment

Problem

Use the result of Problem 6.21 to give a function f that is computable with an oracle for A_{TM}, where for each n, f(n) is an incompressible string of length n.

Step-by-step solution

Step 1 of 2

Compute descriptive complexity of strings $K(x)$ with an oracle for A_{TM}.

• For the given string *x*, start testing all the strings '*S*' up to the length $|x|+c$

Where $c =$ length of *TM* (Turing machine) that halts immediately upon starting.

• All the strings up to the length $|x|+c$ are potential description of *x*.

• If *S* is well formed as $\langle M, w\rangle$ from all binary strings in lexicographic order, then we simulate *M* with input *w* and see if it halts with *x* on the tape.

• Here we do not know whether *M* will halt on input *w* or not.

• An oracle for A_{TM} can determine this.

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

• An oracle for A_{TM} will take $\langle M, w\rangle$ as input and determine whether *M* accepts *w* or not.

• If *M* doesn't halt we move on to the next string *S*, and so on.

• After that we will find lexicographically first string *S* among them.

• In this way shortest string will be determined and it is represented as minimal description $d(x)$.

• From $d(x)$, we find $K(x)$ as

$$K(x) = |d(x)|$$

• By this procedure, we will calculate $K(x)$ with an oracle for A_{TM}.

Comment

Step 2 of 2

From the above procedure, $K(x)$ is the descriptive complexity of strings and computed with an oracle A_{TM}.

Now, we have to know the definition of incompressible strings.

Incompressible strings:

Let *x* be a string. If *x* doesn't have any description shorter than itself then *x* is incompressible.

Let *f* be the function that is computable with an oracle for A_{TM}, for each *n*, calculate *f*(*n*) i.e., incompressible length of string *n*.

Create an oracle TM (Turing machine) that does the following :

1. For a given number *n*, begin

2. Enumerate all strings *x* of length *n*.

3. For each *x*, calculate $K(x)$ by using oracle for A_{TM}.

4. As soon as we find $x = f(n)$ with $K(x) >= |x|$ output *x*.

[$K(x) >= |x|$ means getting a string which is greater than its description.]

5. Halt.

Comment

Problem

Show that the function K(x) is not a computable function.

Step-by-step solution

Step 1 of 4

Definition of complexity strings $\mathrm{K}(x)$:-

Comment

Step 2 of 4

If *x* be binary string, then the minimal description and descriptive complexity of *x*'s are $d(x)\text{ and }\mathrm{K}(x)$respectively. Turing machine *M* and small string *w* we get minimal description is $\langle M,w\rangle$. From several of such shorter strings we select lexicographically among them then we can get descriptive complexity of such strings $\mathrm{K}(x)=|d(x)|$.

Comment

Step 3 of 4

Now we must prove that $\mathrm{K}(x)$is not a computable function.

Let $y_n$be the lexicographical first-string s that satisfies $n<K(y)$. Let M be the Turing machine such that if the input is n, the binary strings that are generated are $x_0,x_1,x_2,x_3\cdots$.

The Turing machine M computes $\mathrm{K}(x_i)$ and if $\mathrm{K}(x)>n$, then the Turing Machine will generate x_i as the output and the machine will halt.

If the machine does not halt, the machine will examine the next lexicographical string x_{i+1}.

The Turing machine M will come across a string x which satisfies $\mathrm{K}(x)>n$because K is unbounded.

Comment

Step 4 of 4

For input n, the output generated is $y_n$but the length of the input n is $\log_2(n)$. So, $K(n)<\log_2(n)+c$, where c is the constant.

So, for all n, $n<K(y_n)$, so it can be said that $n<\log_2(n)+c\ \ n<\log_2(n)+c$ but this is false, if n is large.

This is a contradiction because M cannot compute K(x) so, it is not a computable function.

Comment

Problem

Show that the set of incompressible strings is undecidable.

Step-by-step solution

Step 1 of 2

Incompressible strings:

Let w_i be a string. If w_i doesn't have any description shorter than itself then w_i is incompressible.

Now we have to show that set of incompressible strings is un-decidable.

Let *A* be the set of incompressible strings and assume the contradiction *A* is decidable.

We construct a machine *M* which enumerates *A*.

Enumeration: $f : A \to N$ such that $f(w1) = 1, f(w2) = 2, f(w3) = 3...$ where first, second, and third shortest strings are respectively *w*1, *w*2, & *w*3.

Since *A* reaches infinite there is a string $w_i \in A$.

Comment

Step 2 of 2

Define a Turing machine *T* which computes w_i incompressible string of length *n*

$T =$ " on input *n*

1. Returns the first string w_i that *M* enumerates of length *n*.

2. If $K(\langle T, n\rangle) = c + \log(n)$. For any constant *c*

Then we find *n* such that

$|w_i| = n > c + \log(n)$

The string w_i is shorter description on $\langle M', f(w_i)\rangle$. Where M' is a machine, $f(w_i)$ is input and output as w_i.

Run machine M' each string in lexicographic order from and output the same from *M*.

It contradicts that w_i is compressible. Therefore our assumption that "*A* is decidable" is wrong. So for *A* set of incompressible strings *A* is un-decidable.

Comment

Problem

Show that the set of incompressible strings contains no infinite subset that is Turing-recognizable.

Step-by-step solution

Step 1 of 2

Incompressible strings:

Let w_i be a string. If w_i doesn't have any description shorter than itself then w_i is incompressible.

Now we have to show that set of incompressible strings contains no infinite subset that is Turing recognizable.

Let *A* be the set of incompressible strings and assume the contradiction *A* contains infinite subset and Turing recognizable.

We construct an enumeration function $f : N \to A$, *A* is recognized by machine *M.*

Enumeration: $f : N \to A$ such that $f(1) = w1, f(2) = w2, f(3) = w3...$ where first, second, and third enumerated strings are respectively *w*1, *w*2, *w*3, etc.

Since *A* reaches infinite there is a string $w_i \in A$.

Comment

Step 2 of 2

Turing machine *N* which computes an incompressible string of length at least *n*.

$N =$" On input *n* (an integer in binary notation)

1. Call *M* to enumerate incompressible strings w_i.

2. If $|w_i| >= n$, output w_i and halt"

Now the length of $\langle N \rangle n$ is

$$|\langle N \rangle n| = |\langle N \rangle| + \log(n)$$

$= \log(n) + c$ where *c* is a constant

Now $|w_i| = n$

$n > \log n + c$ for large value of *n.*

$n > |\langle N \rangle n| \quad [\text{As} |\langle N \rangle| n = \log n + c]$

Therefore $|w_i| > n$

This contradicts the incompressibility of string w_i, therefore our assumption that *M* enumerates an infinite subset of incompressible strings w_i is wrong.

Hence set of incompressible strings contains no infinite subset is Turing – recognizable.

Comment

Problem

Show that for any c, some strings x and y exist, where $\mathrm{K}(xy) > \mathrm{K}(x) + \mathrm{K}(y) + c.$

Step-by-step solution

Step 1 of 1

Given:

Consider that *x* and *y* are strings whereas *c* is constant.

Proof:

Consider a Turing machine *TM* having input *w*. Assume that *x* is a binary string in *w*. The minimal description $d(x)$ is the shortest string in *w*. The descriptive complexity of *x* is

$$K(x) = |d(x)|$$

Consider a string *xy* that has been broken in 2 parts *x* and *y* and *c* can be considered as a default value that is used to match left hand side with right hand side.

Now use a Turing machine *T* that breaks its input *w* in description of 2 separate parts $d(x)$ and $d(y)$. The bits of first part and bits of second part are combined together or concatenate.

Once both of the part are obtained than both parts can be run to obtain the string *x* and *y* and can perform concatenate operation to achieve desired output of *xy*.

The length of description of part *xy* is clearly greater than individual parts description of *x* and *y* as some extra part *c* can be seen in the statement. The constant *c* can be termed as a constant for balancing between *xy* and individual parts.

Conclusion:

Hence, the Description complexity of *xy* is greater than sum of description of *x, y* and constant *c*.

$$K(xy) = K(x) + K(y) + c$$

Comments (2)

Problem

Let $S = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \{\langle M\rangle\}\}.$ is Turing-recognizable.

Step-by-step solution

Step 1 of 1

S and $\overline{S}$ are not recognizable can be proof by following method:

Proof:

This can be proved by reduction that is by reducing $\overline{A_{TM}}$ to S and $\overline{A_{TM}}$ to $\overline{S}$. At first it is necessary to show that $A_{TM} \leq_m S$.

Consider a function *f* having input $\langle M, w\rangle$ and output M' which works as follow on input *x*.

1. When $x == \langle M'\rangle$

ACCEPT

2. When $x == 0$ then execute M on input *w*.

If *w* is accept the **ACCEPT** else **REJECT**

3. **REJECT.**

It is a many-one reduction from $\overline{A_{TM}}$ to S. When $\langle M, w\rangle \in A_{TM}$ then it shows that *w* is accepted by *M*. According to M' definition it can be states that $L(M) = \{\langle M'\rangle, 0\} \neq \{\langle M'\rangle\}$. When $\langle M, w\rangle \notin A_{TM}$ then it shows that *w* is not accepted by *M*. As 0 is not acceptable by M' therefore $L(M') = \{\langle M'\rangle\}$

Now show $\overline{A_{TM}}$ to $\overline{S}$. Consider another function *f* having input $\langle M, w\rangle$ and output M'' which works as follow on input *x*.

1. When $x == \langle M''\rangle$

2. Execute M on input *w*. If w is accept the **ACCEPT** else **REJECT**

3. **REJECT.**

It is also a many-one reduction from $\overline{A_{TM}}$ to $\overline{S}$. When $\langle M, w\rangle \notin A_{TM}$ then it shows that *w* is not accepted by *M*. According to M'' definition it can state that M'' is not acceptable by M'' therefore $M'' \notin S$. When $\langle M, w\rangle \in A_{TM}$ then it shows that *w* is accepted by *M* then according to M'' definition M'' is acceptable by M''.

Comment

Problem

Let $R \subseteq \mathcal{N}^k$

A **a.** $R_0 = \{0\}$

b. $R_1 = \{1\}$

c. $R_= = \{(a,a) | \ a \in \mathcal{N}\}$

d. $R_< = \{(a,b) | \ a,b \in \mathcal{N} \text{ and } a < b\}$

Step-by-step solution

Step 1 of 5

The k-ary relation $R \subseteq N^k$ is ***definable*** in $Th(N,+)$, if a formula ϕ can be given with k free variables $x_1,...,x_k$, such that for all $a_1,...,a_k \in N$, the formula $\phi(a_1,...,a_k)$ is true only when $a_1,...,a_k \in R$.

• The theory $Th(M)$ of a model M is the collection of true sentences in the language of M.

• The theory for this problem is $Th(N,+)$, so the model will be $(N,+)$. Thus, only need to find formulas that are true for the given relations over the $(N,+)$ model.

Comment

Step 2 of 5

a)

$R_0 = \{0\}$

• There is only one value 0 to be considered. Adding this value to the variable $y \in N$ will produce no change in the value of y as $y + 0 = y$.

• So, in the formula ϕ_0 to make R_0 definable in $Th(N,+)$ for all y, it has; $\forall x \in R_0 \forall y [x + y \to 0 + y]$.

From this, the formula can be expressed as:

$\phi_0(x) = \forall y [x + y = y]$

Comment

Step 3 of 5

b)

$R_1 = \{1\}$

• Similarly, in this case only one value is to be considered. So adding any value from $R_1 = \{1\}$ to the variable $y \in N$ will increment the value of y by one as:

$\forall x \in R_1 \forall y [x + y \to 1 + y]$

Therefore, the formula ϕ_1 to make R_1 definable in $Th(N,+)$ will be:

$\phi_1(x) = \forall y[x + y = y + 1]$

Comment

Step 4 of 5

c)

$R_{=} = \{(a,a) \mid a \in N\}$

• The $R_{=}$ relation is a 2-ary relation. So, the formula to make it definable will be a binary function $\phi_{=}(x,y)$. This function has to be true if and only if $x = y$.

• For it be defined for $\forall z \in N$, the only possible value of z is 0. This is true when $\phi_0(z)$ is true, which can be written as $\forall z[\phi_0(z) \to x + z = y]$.

Hence, the function is given by:

$\phi_{=}(x,y) = \forall z[\phi_0(z) \to z + x = y]$

Comment

Step 5 of 5

d)

$R_{<} = \{(a,b) \mid a,b \in N \text{ and } a < b\}$

• This relation is also 2-ary, so the formula will be a binary function $\phi_{<}(x,y)$. It needs to be true when $x < y$ and for all the values of the variable $\forall z \in N$.

• The only possible value of z is 0. Since all the other non-zero values will affect the magnitude of the quantity $x + z$, which will make it difficult to compare it against the value of y.

• As $\phi_0(z)$ is true when $z = 0$, it can be incorporated into $\phi_{<}$.

$\forall z[\phi_0(z) \to x + z = y]$.

This is the same as the definition of the function:

$\phi_{<}(x,y) = \forall z[\phi_0(z) \to z + x < y]$

Comment