Problem 1.

You are employed as a programmer, and you are asked to write the given program. For each case, write "Y" or "N" if the program can or cannot be written. Do not concern yourself with memory limitations, that is, assume that the computer used to run your program has a large, effectively unbounded, amount of memory.

Solution

Part (a)

receives in input a generic C program x, and counts the number of statements in x.

Y

Part (b)

receives in input a generic C program x, and an input string w, and counts the number statements executed at least once when x runs on input w.

N

Part (c)

receives in input a generic C program x and an input string w, and counts the number of statements never executed when x runs on input w.

N

Part (d)

receives an input a generic C program x and decides whether x is syntactically correct.

Y

Part (e)

receives in input two natural numbers and computes a specific function $f : \mathbb{N}^2 \to \mathbb{N}$.

N

Part (f)

receives in input a generic arithmetic expression e composed of integers and the four arithmetic operators, and computes its value.

Y

Part (g)

halts on the empty string

Y

Part (h)

receives in input a generic C program x and decides whether x halts only on the empty string.

N

Part (i)

receives in input two generic regular expressions and decides whether they are equivalent.

Y

Part (j)

receives in input a generic C program x and the name of one of its functions, f, and decides whether x can ever call f.

N

Part (k)

receives in input a generic C program x, an input string w, and the name of one of its functions, f, and decides whether x calls f when running on input w.

N

Part (l)

receives in input two generic C program x_1 and x_2 and an input string w, and decides whether x_1 and x_2 produce the same output when running on input w.

N

Part (m)

receives in input two generic C programs x_1 and x_2 and decides whether x_1 and x_2 produce the same output when running on every possible input.

N

Part (n)

receives in input two generic C programs x_1 and x_2, and decides whether x_1 and x_2 produce the same output when running on at least one input.

N

Part (o)

receives in input a generic C program x, an input string w, and a natural number n, and decides whether x uses less than n bytes of memory when running on w.

Y

Part (p)

receives in input a generic C program x, an input string w, and decides whether there is an $n \in \mathbb{N}$ such that x uses less than n bytes of memory running on input w.

N

Problem 2.

Use reduction to prove that the language

$$L = \{p(M)p(w) : \text{ the TM } M \text{ never enters its initial state again when running on } w\}$$

is undecidable.

Solution

Proof. via contradiction

Assume that L is decidable $\Rightarrow \exists$ a Turing Machine K which generates L. Define a new Turing machine K' which takes as input a Turing machine M and string w encoded as $p(M)p(w)$. K' will perform a series of modifications to the input machine M, we will call this modified machine M'. Denote the initial state of M as q_0, K' will create a new state q_0', set it as the new initial state for M' and add an ϵ-transition from $q_0' \to q_0$. Thus M halts $\Leftrightarrow M'$ halts. Note that M' will never enter state q_0' again when running.

Next take the accepting and rejecting states of M, call them a and r, these will no longer be the halting states of M', but otherwise exist the same. Instead introduce new states a' and r' which are the respective accepting and rejecting states, however these include no edges. Next add ϵ-transitions from a and r to q_0'. Then M halts $\Leftrightarrow M$ enters either state a or $r \Leftrightarrow M'$ enters state q_0' again[1]. Further, if M does not halt then M does not enter states a or r. We know the only transitions into q_0' are through a and r, we therefore can conclude that M' does not enter state q_0' again[2].

Now that our machine K' has set up this machine M' based on its input, K' will then run the machine K recognizing L on M' and output the result. Note that M' is guaranteed to not halt, but that is not important or relevant to machine K. Construction of our M' is easily computable, further K is guaranteed to return a result by assumption. Thus K' is decidable.

If K' accepts then q_0' is entered again $\Rightarrow M$ halts[1].
If K' rejects then q_0' is not entered again $\Rightarrow M$ hangs[2].

Thus K' solves the halting problem $\Rightarrow K'$ is undecidable. This contradicts our statement that K' is decidable. Therefore our assumption is false and we conclude that L is undecidable. □