Homework 9

CS 5315 (Theory of Computation)

Instructor: Dr. Vladik Kreinovich

1. Write the proof for Halting Problem.

Halting Problem Theorem:

There is no algorithm that will, given an arbitrary program p and input data d, check whether p halts on d or not.

PROOF By reduction to a contradiction.

Part 1 Let's assume there exists such algorithm and call it *"halt_checker"*. The *halt_checker* algorithm correctly checks if a given program p halts on a given data d, returning *true* if p halts on d, or returning *false* if p does not halt on d.

In the computer, the program p is represented as a sequence of 0's and 1's. We can interpret this sequence as an integer. This integer will be called a *code* of program p.

For every integer c that is a code of a syntactically correct program, we will define the result of applying this program to an integer n by $f_c(n)$.

Part 2 Let's define a function $f(n)$.

$$f(n) \equiv \begin{cases} f_n(n) + 1 & \text{if } n \text{ is a code of a syntactically correct program} \\ & \text{and this program halts on } n\text{, i.e.,} \\ & \textit{halt\_checker}(f_n, n) = \textit{true;} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Part 3 This function f is computable. Indeed, it can be computed as follows: first, we pass n as input to a Pascal compiler which checks whether the sequence of 0's and 1's that represents n is a syntactically correct code.

If the integer n does not represent a syntactically correct code, then, we return 0 as $f(n)$; otherwise, if n is a syntactically correct program, then we apply the *halt_checker* to this program n and to the same integer n serving as input data.

If *halt_checker*(n, n) = *false*, this means that the program f_n does not halt on the data n, so we return 0 as $f(n)$.

If *halt_checker*(n, n) = *true*, this means that the program f_n halts on n. In this case, we apply this program f_n to the input n, and then add 1 to the result $f_n(n)$ of this application.

Part 4 We have shown that the function $f(n)$ is computable. Moreover, we can easily write this function $f(n)$ as a Pascal program. Therefore, it has a code.

Let us denote this code by c. By definition, the function $f(n)$ always halts. The fact that c is a code of the function f means that for every n, $f(n)$ coincides with $f_c(n)$, or $f(n) = f_c(n)$.

Since it is true for all n, it must be true for $n = c$, so $f(c) = f_c(c)$.

But by definition (1) of function f, $f(c) = f_c(c) + 1$. Hence, $f_c(c) = f_c(c) + 1$, which is impossible. **Contradiction!**

Therefore our assumption that a *halt_checker* exists is false; there is no *halt_checker* algorithm that would check whether a given program p halts on a given data d.

2. A *cube_checker* is an algorithm that checks whether a given program p always computes n^3. Formally: a *cube_checker* is an algorithm that, given a program p that always halts, returns *true* if for all n, $p(n) = n^3$, and returns *false* if there exists an integer n such that $p(n) \neq n^3$.

Prove that the *"cube_checker"* is impossible.

PROOF By reduction to *"zero_checker"*.

Let's assume that such *cube_checker* program exists.

Let us use this *cube_checker* to build a *"zero_checker"* program that receives a program q as input, and returns *true* if for all n, $q(n) = 0$, or *false* if there exists n such that $q(n) \neq 0$.

Indeed, let us design the following algorithm U: it receives a program q as input, constructs a new program p that computes $q(n) + n^3$ for all n, and passes this program p as an input to the *cube_checker* program.

If *cube_checker* returns *true*, it means that for all n, $q(n) + n^3 = n^3$, which implies that $q(n) = 0$ for all n.

If the *cube_checker* returns *false*, it means that $q(n) + n^3 \neq n^3$ for some n, which implies that $q(n) \neq 0$ for this n.

Thus, the algorithm U returns *true* if and only if the given program q always returns 0. Thus, U is a *zero_checker.*

But it was already proved that a *zero_checker* is impossible; therefore a *cube_ checker* is also impossible. □