1 Harder than Halting

In class we proved that the Halting problem \( \text{HALT} = \{[P \leftarrow L] : P(L) \neq \bot \} \) (where \( P \) is a WHILE program and \( L \) a list) is undecidable. It is very tempting to believe that if we only could solve the halting problem, than we could algorithmically solve any other computational problem (without the danger of entering an infinite loop.) In this problem, you are asked to prove that this intuition is wrong: even if you could decide HALT, there are still plenty of problems that cannot be solved in an effective way.

Assume that you can write programs containing a special instruction \( X = \text{HALT}(Y) \) that on input a list \( Y \), determines if \( Y \in \text{HALT} \) and set \( X \) to 0 or 1 accordingly. (To be precise, you don’t need to assume that you can write such programs: these programs are just Turing reductions, so you can certainly write them. The assumption is that after you write such a program/reduction \( R \), you can run \( R^{\text{HALT}} \), i.e., execute \( R \) using HALT as an oracle to answer the query.) Prove that there exists a decision problem \( D \) (i.e., a set of lists) that is still algorithmically unsolvable, i.e., it is not decided by any program \( R^{\text{HALT}} \) written in this extended programming language. (Your solution should contain a formal description of the language \( D \), i.e., a description detailed enough to define a precise set of lists. E.g., if you define \( D \) as a set of WHILE programs or reductions, then you should specify how these WHILE programs/reductions are encoded as lists.)

Remark: Intuition aside, formally, what this problem is asking you to do is to prove that there exist a language \( D \) that is not Turing reducible to HALT. If you are not sure about the level of detail you should use in describing your solution, ask the TA or instructor. As a general guideline, your solution should be at very least as detailed as the proof (given in class) that Diag, Loop, and Halt are undecidable. (In fact, since you have much more writing space on paper than on a slide, we expect your solution to be more verbose than the instructor’s slides.)

2 Revisiting time complexity

As part of this problem, you are asked to re-evaluate our assumptions about program running times, and results about the time required to solve certain problems. You will have to run programs on your computer, and measure the time required to run them. You can choose any method to measure the run time as you deem appropriate, and any computer of your choice. Though, you should run all your experiments (involving real time measurements) on the same computer, and include a description of the computer (e.g., cpu and clock rate, and whether you used “runghc” or “ghc” to compile the while interpreter) and the method you used to measure time as part of your solutions.

[PART A] Write any WHILE program of your choice that can operate on lists of arbitrary length. E.g., the program to reverse a list presented in class might do, but other programs with higher running times might work better. The main requisites for this program are that:
• It should be able to accept inputs of arbitrary length.
• It should have a running time that is a fairly simple function of the input size.
• The program should be fairly simple and short, and easy to understand, though it is not required to be doing anything useful.
• The program should be written using only the core WHILE instructions (no extensions, macros, etc.), so that it can be easily encoded as a list and passed as input to our self-interpreter.

You should include the listing of your program as part of your solutions, as you are not required to submit the code as a separate file.

Experiment running the programs on inputs of increasing length, and measuring the running time of your program on each input in two different ways:

• Measure the actual clock time (in seconds) used by your computer to run the program.
• Measure the time as the number of basic instructions executed by the program during its run. (Here time is expressed just by an integer number.) You can count the number of instructions executed by the program by encoding it as a list and running it through a modified version of the self interpreter posted on the webpage.

The number of test runs and input sizes is left to you, but try to be reasonable, and come up with a sensible collection of tests. Plot your data and draw your conclusions.

• How reasonable is to take the number of basic WHILE instruction executed as a measure of the running time of a program?
• Can you come up with a formula that, given the number of instructions executed by a program, produces a realistic estimate of the actual running time (in seconds) the program will take to run on your system?

[PART B] Based on the results from PART A (and, if needed, additional experiments, e.g., involving the actual running time of the self interpreter), formulate and justify a concrete version of the time hierarchy theorem proved in class for WHILE programs. Remember, in class we proved that for any (nice, time constructible) function $T(n)$ there are problems that can be solved in time $T(n)$, but cannot be solved in time $T'(n) = T(n)/c$ (for some fixed but unspecified constant $c > 1$) where the time is measured using our abstract model of computation as the number of basic WHILE instructions executed. Here you are asked to make (and justify) a similar (but more concrete) statement concerning the actual running time of WHILE programs on your system, where, based on your data, the unspecified constant $c$ is replaced by an actual number. (By system we mean your computer, compiler, while interpreter, and execution environment to run WHILE programs.)

Your statement should be of the form “there are problems that can be solved in time $T(n)$, but cannot be solved in time $T'(n)$”, where $T, T'$ are functions expressing the actual running time of the programs on your system, measured in seconds.

While we do not expect you to provide a formal proof of your statement, you should give a precise and convincing justification of your claim and your specific choice of functions $T$ and $T'$.

---

1You can do this in a variety of ways, e.g., plotting each measure against the input size, or plotting the two time measure one against the other.