CS 70 Discrete Mathematics and Probability Theory
Fall 2020 DIS 8B

1 Countability and the Halting Problem

Prove the Halting Problem using the set of all programs and inputs.

a) What is a reasonable representation for a computer program? Using this definition, show that the set of all programs are countable. *(Hint: Python Code)*

b) We consider only finite-length inputs. Show that the set of all inputs are countable.

c) Assume that you have a program that tells you whether or not a given program halts on a specific input. Since the set of all programs and the set of all inputs are countable, we can enumerate them and construct the following table.

	x_1	x_2	x_3	x_4	$\dots$
p_1	H	L	H	L	$\dots$
p_2	L	L	L	H	$\dots$
p_3	H	L	H	L	$\dots$
p_4	L	H	L	L	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

An H (resp. L) in the ith row and jth column means that program p_i halts (resp. loops) on input x_j. Now write a program that is not within the set of programs in the table above.

d) Find a contradiction in part a and part c to show that the halting problem can't be solved.

Solution:

a) As in discussion and lecture, we represent a computer programs with a set of finite-length strings (which, int turn, can be represented by a set of finite length binary strings). The set of finite length binary strings are countably infinite. Therefore the set of all programs is countable.

b) Notice that all inputs can also be represented by a set of finite length binary strings. The set of finite length binary strings are countably infinite, as proved in Note 11. Therefore the set of all inputs is countable.

c) For the sake of deriving a contradiction in part (d), we will use the following program:

procedure P'(x_j)
  if $P_j(x_j)$ halts **then**
    loop

```
    else
        halt
    end if
end procedure
```

d) If the program you wrote in part c) exists, it must occur somewhere in our complete list of programs, P_n. This cannot be. Say that P_n has source code x_j (i.e. its source code corresponds to column j). What is the (i, j)th entry of the table? If it's H, then $P_n(x_j)$ should loop forever, by construction; if it's L, then $P_n(x_j)$ should halt. In either case, we have a contradiction.

2 Fixed Points

Consider the problem of determining if a function F has any fixed points. That is, given a function F that takes inputs from some (possibly infinite) set $\mathscr{X}$, we want to know if there is any input $x \in \mathscr{X}$ such that $F(x)$ outputs x. Prove that this problem is undecidable.

Solution:

We can prove this by reducing from the Halting Problem. Suppose we had some function `FixedPoint`(F) that solved the fixed-point problem. That is, we supply a `FixedPoint` a function F, and it outputs `true` if it can find some $x \in \mathscr{X}$ such that $F(x)$ outputs x, and `false` if no such x exists. We can define `TestHalt`(F,x) as follows:

```
def TestHalt(F, x):
  def F_prime(y):
    F(x)
    return y
  return FixedPoint(F_prime)
```

If $F(x)$ halts, we have that $F'(y)$ will always just return y, so every input is a fixed point. On the other hand, if $F(x)$ does not halt, F' won't return anything for any input y, so there can't be any fixed points. Thus, our definition of `TestHalt` must always work, which is a contradiction; this tells us that `FixedPoint` cannot exist.

3 Computability

Decide whether the following statements are true or false. Please justify your answers.

(a) The problem of determining whether a program halts in time 2^{n^2} on an input of size n is undecidable.

(b) There is no computer program `Line` which takes a program P, an input x, and a line number L, and determines whether the L^{th} line of code is executed when the program P is run on the input x.

Solution:

(a) False. You can simulate a program for 2^{n^2} steps and see if it halts.

Generally, we can always run a program for any fixed *finite* amount of time to see what it does. The problem of undecidability arises when no bounds on time are available.

(b) True.

We implement `Halt` which takes a program P, an input x and decides whether $P(x)$ halts, using `Line` as follows. We take the input P and modify it so that each exit or return statement jumps to a particular new line. Call the resulting program P'. We then hand that program to `Line` along with the input x and the number of the new line. If the original program halts than `Line` would return true, and if not `Line` would return false.

This contradicts the fact that the program `Halt` does not exist, so `Line` does not exist either.

At a high level, you can show the undecidability of a problem by using your program which solves the problem as a subroutine to solve a different problem that we know is undecidable. Alternatively, you can do a diagonalization proof like we did for `Halt`. The first approach is natural for computer programmers and flows from the fact that you are given P as text! Therefore you can look at it and modify it. This is what the solution above does.