CS 70 Discrete Mathematics and Probability Theory
Spring 2021 DIS 7A

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	...
p_1	H	L	H	L	...
p_2	L	L	L	H	...
p_3	H	L	H	L	...
p_4	L	H	L	L	...
⋮	⋮	⋮	⋮	⋮	⋱

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 Hello World!

Determine the computability of the following tasks. If it's not computable, write a reduction or self-reference proof. If it is, write the program.

(a) You want to determine whether a program P on input x prints "Hello World!". Is there a computer program that can perform this task? Justify your answer.

(b) You want to determine whether a program P prints "Hello World!" before running the kth line in the program. Is there a computer program that can perform this task? Justify your answer.

(c) You want to determine whether a program P prints "Hello World!" in the first k steps of its execution. Is there a computer program that can perform this task? Justify your answer.

Solution:

(a) Uncomputable. We will reduce `TestHalt` to `PrintsHW`(P,x).

```
TestHalt(P, x):
  P'(x):
    run P(x) while suppressing print statements
    print("Hello World!")
```

```
if PrintsHW(P', x):
  return true
else:
  return false
```

If `PrintsHW` exists, `TestHalt` must also exist by this reduction. Since `TestHalt` cannot exist, `PrintsHW` cannot exist.

(b) Uncomputable. Reduce `PrintsHW`(P,x) from part (a) to this program `PrintsHWByK`(P,x,k).

```
PrintsHW(P, x):
  for i in range(len(P)):
    if PrintsHWByK(P, x, i):
      return true
  return false
```

(c) Computable. You can simply run the program until k steps are executed. If P has printed "Hello World!" by then, return true. Else, return false.

The reason that part (b) is uncomputable while part (c) is computable is that it's not possible to determine if we ever execute a specific line because this depends on the logic of the program, but the number of computer instructions can be counted.