Quiz 6

CSCI 334: Spring 2024

Your name: _____

The `Equiv` function is defined as follows.

$$\texttt{Equiv(p1, p2)} = \begin{cases} \texttt{true} & \text{if } \texttt{p1} \text{ and } \texttt{p2} \text{ compute the same output on all possible inputs.} \\ \texttt{false} & \text{otherwise} \end{cases}$$

where `p1` and `p2` are functions of the form

```
def MyFunc(x):
  ...
```

We are going to prove by reduction that `Equiv` is not computable. I need your help checking my proof.

1. What's the one fact we should assume in this proof? (hint: what function can we assume we have in our standard library)

   That the `Equiv` function exists and is computable.

2. Name a function already known not to be computable.

   The `Halt` function.

3. Here's my reduction function.

```
def Halt(p,x):
  p1 = "def MyFunc1(y):\n" +
       "  " + p + "\n" +
       "  MyFunc(" + x + ")\n" +
       "  return 1"
  p2 = "def MyFunc2(y):\n" +
       "  return 1"
  return Equiv(p1, p2)
```

(a) Suppose the `p` we call `Halt` with is:

```
def MyFunc(x):
  x = 0
  while (true):
    x += 1
  return x
```

and suppose `x` is `2`. What is the value of `p1` in `Halt`?

```
def MyFunc1(y):
  def MyFunc(x):
    x = 0
    while (true):
      x += 1
    return x
  MyFunc(2)
  return 1
```

You can think of `p1` as a "gadget" that makes `p` (whatever it is) return the same thing as `MyFunc2` (i.e., `1`) if and only if `p` halts.

(b) Is the reduction correct? Why or why not? Explicitly consider what happens when a given `p` halts and when it does not.

Yes, the reduction is correct.

If `p` halts, then `p1` will also halt and return `1`. Observe that `p1` will always return `1`, no matter the input `y`, because we ignore `y` and always run it on `x`. Since `p2` always returns `1` for every input (it is a constant function), `Equiv(p1,p2)` returns `true` whenever `p` halts.

If `p` does not halt, then `p1` will also not halt. In fact, `p1` does not halt, no matter the input `y`. In this case, `Equiv(p1,p2)` returns `false` whenever `p` does not halt, because `p2` always halts and returns `1`.

Since `Equiv(p1,p2)` returns `true` whenever `p` halts and `Equiv(p1,p2)` returns `false` whenever `p` does not halt, then clearly we are able to write a `Halt` function. However, we already know that we cannot write a `Halt` function; it is undecidable. Therefore, we've derived a contradiction and must conclude that our assumption that `Equiv` could exist is false. `Equiv` is not computable.