311 Quiz Section: December 8, 2011

Notation for Halting Problem and Reductions:

Capital letter: an executable program,
e.g. **G**: a program that asks the user for their name and prints out "Hello <name>!"

Capital letter enclosed in brackets: the source code for the executable
e.g. **<G>** :

```
name = raw_input("What's your name? ")
if name == "Kris":
     print "Hello, Kris!"
else:
     while True:
          print "Hello, %s" % name
```

Lower case letter: the input to an executable program
e.g. **x** : the name entered by the user

The halting problem is undecidable:

Suppose H is an executable program that takes as input:

- the source code of some program, <P>
- an input to that program, x

We can represent running H on this input as H(<P>, x)

And suppose that the output (return value) of H is:

- 1 (true) if P would halt if it were run on input x
- 0 (false) if P would not halt (loop forever) if it were run on input x

We have proven that H cannot exist, i.e. no program can decide (for arbitrary P and x) whether or not P halts on x.

Proof by Reduction (in general):

We know that it is impossible to do thing A.
We want to prove that it is also impossible to do thing B.
We prove this by showing that if we could do thing B, then we could also do thing A.
This is called a "reduction from A to B".

Examples:

- There is a board game that involves moving different colored tokens around on numbered squares. We know that blue tokens can never win the game. We want to prove that blue tokens can also never reach square 50. We can prove this by showing that if a token can reach square 50 then it can win the game.

- There is a role-playing game that involves different player types - magicians, fighters, thieves, etc. – and a treasure room guarded by monsters. We know that it is impossible for a thief to enter the treasure room. We want to prove that it is also impossible for a

thief to defeat a dragon in battle. We can prove this by showing that if a player can defeat a dragon, then the player can enter the treasure room.

- There is a game called, “Can some computer program solve this problem?” (Haven't you seen it on TV? It's better than Jeopardy!) We know that it is impossible for a computer program to solve the halting problem. We want to show that it is also impossible for a computer program to solve problem X. We can prove this by showing that if a program can solve problem X, then it can also solve the halting problem. (Unlike the previous examples, this game doesn't have any winners – no analogy to red tokens or magicians. Not even quantum computers can solve an undecidable problem!)

Example decidability proof: Use the undecidability of the halting problem to show that the “always halts” problem is also undecidable.

Definition of “always halts” problem: Recall that the halting problem is to take source code <P> and input x and decide if the program P would halt if run on input x. The “always halts” problem is slightly different. The always halts problem is to take source code <P> and decide if the program P always halts on any input.

Proof by reduction:
Known: Halting problem is undecidable: i.e. there is no program H that takes input (<P>, x) and returns 1 if P would halt on x and returns 0 if P would not halt on x.
Want to prove: Always halts problem is undecidable: i.e. there is no program B that takes input (<P>) and returns 1 if Q would always halt (regardless of input) and returns 0 if P would not always halt.
Need to show: if the “always halts” problem is decidable then the halting problem is decidable: i.e. if program B exists, I can use it to create program H.

How do I show this?
I can build H as follows:

1. H takes input <P> and x.
2. **H modifies source code <P> to remove any variable assignments based on the input and replace them with variable assignments based on x (“hard code” x into <P>). We call this modified source code, <P'>.**
3. H sends <P'> to B: i.e. H uses B as a function and calls B(<P'>).
4. **If B returns 1 (P' halts on all input), then H returns 1 (P halts on x). If B returns 0 (P' doesn't halt on all input), H returns 0 (P doesn't halt on x).**

Note that steps 2 and 4 are the parts that vary from reduction proof to reduction proof:

- How do you modify the input to H to make it the right kind of input to B?
- How does H interpret the output from B?

We have shown that if B exists then we can use it to build A. i.e. we have shown that if we can solve the “always halts” problem, then we can solve the halting problem. But we know the halting problem is undecidable. Thus, we have proven that the “always halts” problem is also undecidable.

Note: when you modify <P> to <P'> you must be sure not to make any changes that could affect whether the program halts on x. In other words, if P halts on input x, then P' must halt on input x, and if P loops forever on input x, then P' must loop forever on input x. (But notice that changes to what a program prints will never affect whether that program halts or not.)