CS 70 Discrete Mathematics and Probability Theory
Spring 2015 Vazirani Discussion 8W

1. Sanity check!

Prove that the halting problem is undecidable by writing a self-referential program. You can assume you have some subroutine TESTHALT(P,x) that returns true if P halts on input x and false otherwise. Try not to look at the notes while you do this!

2. Self Reference

Here we learn about self-reference through a paradox dubbed the "Grelling-Nelson Paradox". We define two types of adjectives:

1. An adjective is called **autological** if it describes itself. For example "short" is autological, since the word "short" is a short word. "English", "unhyphenated" and "pentasyllabic" are also autological.
2. An adjective is called **heterological** if it does not describe itself. Hence "long" is a heterological word (because it is not a long word), as are "unwritten" and "monosyllabic".

Now, it seems that all adjectives must either be autological or heterological, because either it describes itself, or it doesn't. Let's see if this is truly the case.

a. Is "heterological" a heterological word?

Answer: NO $\implies$ "heterological" is autological $\implies$ "heterological" describes itself $\implies$ "heterological" is heterological, which is a contradiction.

YES $\implies$ "heterological" is heterological $\implies$ "heterological" does not describe itself $\implies$ "heterological" is not heterological, which is a contradiction.

Here we see an instance of a self-reference paradox! Woohoo

b. Is "autological" an autological word?

Answer: By definition, "autological" is autological if and only if "autological" describes itself. In other words, "autological" is autological if and only if "autological" is autological. But this statement is trivially always true. (In logic, such a statement is called a **tautology**.)

In particular, this means that it does not matter whether we take "autological" to be autological or heterological, because either way, our tautological definition is always guaranteed to be true. Indeed we can check that neither choice gives rise to an inconsistency:

1. Suppose we say "autological" is autological. If we now ask if "autological" describes itself, then yes, it does, and is thus indeed autological.
2. Suppose we say "autological" is heterological. If we now ask if "autological" describes itself, then no, it does not, and is thus indeed heterological.

Hence we can choose "autological" to be either autological or heterological. (But of course, we cannot choose it to be both at the same time.)

3. Dead-Code Problem

Here we try to show the dead-code problem is undecidable. The problem is as follows: given a program P, an input x, and a line n in P, does P on input x ever execute line n?

Answer: Here we do a reduction of the halting problem to the dead-code problem. In other words, we show that if the dead-code problem were decidable, then the halting problem would be decidable as well.

Given any program P, we can replace any instruction corresponding to `halt` with `goto line k`, where line k is the end of the program. Note that P, modified this way, would still have the same functionality as before, but now it has the property that it halts if and only if it gets to line k. Therefore, if we can solve the dead-code problem, we would also be able to solve the halting problem.

Hence the dead-code problem must be undecidable.

4. Undecidability

Here we use reductions to prove that certain problems are undecidable.

a. The totality problem is defined as follows: A program F is said to be **total** if $F(x)$ is defined for all x. Assume there exists a procedure TOTAL that takes an input program P and outputs 'Yes' if P halts on all inputs and 'No' otherwise. Argue that this means we could solve the halting problem.

Answer: We can decide whether P halts on input x by constructing another program Q that ignores its own input and always emulates P on input x. Note that Q halts on all inputs if and only if P halts on input x. We now use TOTAL to decide whether Q halts on all inputs, and in doing so we also decide whether P halts on input x.

b. What does this mean about our assumption that TOTAL exists? Is the totality problem decidable?

Answer: This means that our assumption was wrong, because we know that the halting problem is undecidable (proof in the lecture notes). I.e. the totality problem is also undecidable.

c. The equivalence problem is defined as follows: Given two programs P and Q, do they compute the same function? (is $P(x) = Q(x)$ for all x?). Prove that the equivalence problem is undecidable by reducing the totality problem to it. In other words, show that you can solve the totality problem using the equivalence problem.

Answer: We show that if the equivalence problem is decidable, then the totality problem is decidable as well.

To decide whether P is total, we construct another program P' which emulates P except that it ignores any output by P and always prints `YES`. Also, let Q be the program that just prints `YES` and halts. Complete the reduction by noting that

1. If P is total, P' and Q are equivalent because $P'(x) = Q(x) =$ `YES` for all x.
2. If P is not total, P' and Q are not equivalent because there exists some x for which $P'(x)$ is undefined, but $Q(x) =$ `YES`.

Since the totality problem is undecidable, the equivalence problem must also be undecidable.