CS 70 Fall 2006 — Discrete Mathematics for CS — Papadimitriou & Vazirani — Lecture 42

Self-Reference and Computability

The Liar Paradox

Propositions are statements that are either true or false. We saw before that some statements are not well defined or too imprecise to be called propositions. But here is a statement that is not a proposition for more subtle reasons: "All Cretans are liars." So said a Cretan in antiquity, thus giving rise to the liar paradox which has amused and confounded people over the centuries. A more precise restatement of this paradox is the following statement: "this statement is false." Is the statement true? If the statement is true, then what it asserts must be true; namely that it is false. But if it is false, then it must be true. So it really is a paradox. Around a century ago, this paradox found itself at the center of foundational questions about mathematics and computation.

In this lecture, we will study how this paradox relates to computation. Before doing so, let us consider another manifestation of this paradox, created by the great logician Bertrand Russell. In a village with just one barber, every man keeps himself clean-shaven. Some of the men shave themselves, while others go to the barber. The barber proclaims: "I shave all and only those men who do not shave themselves." It seems reasonable then to ask the question: Does the barber shave himself? Thinking more carefully about the question though, we see that we are presented with a logically impossible scenario. If the barber does not shave himself, then according to what he announced, he shaves himself. If the barber does shave himself, then according to his statement he does not shave himself.

Halting Problem

Are there tasks that a computer cannot perform? For example, we would like to ask the basic question when compiling a program: does it go into an infinite loop? In 1936, Alan Turing showed that there is no program to perform this task. The proof of this remarkable fact is very elegant and combines two ingredients: self reference (as in the liar paradox), and the fact that we cannot separate programs from data! In computers, a program is represented by a string of bits just as integers, characters, and other data are. What matters is how the string of bits is interpreted.

We will now examine the halting problem. Given the description of a program and its input, we would like to know if the program ever halts when it is executed on the given input. In other words, we would like to write a program Halt that behaves as follows:

$$\text{Halt(code, input)} = \begin{cases} \text{“yes”}, & \text{if code(input) halts} \\ \text{“no”}, & \text{if code(input) loops} \end{cases}$$

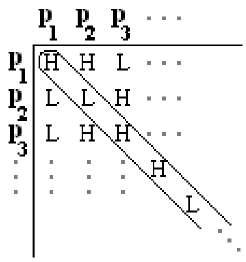
Why cannot such a program exist? First, let us use the fact that a program is just a bit string, so it can be input as data: consider Halt(code, code), which will output "yes" if code(code) halts and "no" if code(code) loops forever. Here is proof that no such program can exist.

Proof: Define the program

```
Turing(code)
  if Halt(code, code) = Yes, then loop forever
  else halt
```

So if the program 'code' when given as input 'code' halts, then Turing loops forever; otherwise, Turing halts. Let us look at what Turing(Turing) does. There are two cases: either it halts, or it does not. If Turing(Turing) halts, then it must be the case that Halt(Turing, Turing) returned false. But that would mean that Turing(Turing) should not have halted. In the second case, if Turing(Turing) does not halt, then it must be the case that Halt(Turing, Turing) returned true, which would mean that Turing(Turing) should have halted. In both cases, we arrive at a contradiction which must mean our initial assumption, that the program Halt exists, was wrong. Thus, it is impossible for a program to check if any general program ever halts.

What proof technique did we use? This was actually a proof by diagonalization. Why? Since the number of computer programs is countable, we can enumerate all programs as follows (where p_i represents the i^{th} program):



The i,j^{th} entry is H if program p_i halts on input p_j and L if it does not halt. Now if the program Turing exists it must occur somewhere on our list of programs, say as p_n. But this cannot be, since if the n^{th} entry in the diagonal Halt(p_n, p_n) claims to halt, then Turing loops. If the entry asserts that it loops, then Turing should have halted. Thus this contradicts the fact that we listed all possible programs p, and therefore the halting problem cannot be solved.

In fact, there are many more cases of questions we would like to answer about a program, but cannot. For example, we cannot know if a program ever outputs anything or if it ever executes a specific line. We cannot even check to see if the program is a virus.