6.080 / 6.089 Great Ideas in Theoretical Computer Science
Spring 2008

6.080/6.089 GITCS	21 February 2008
Lecture 5	
Lecturer: Scott Aaronson	*Scribe: Emilie Kim*

1 Administrivia

When it is your turn to do scribe notes, please have a rough draft prepared within one week. Then you have until the end of the course to get the notes in better shape, and you'll want to do that because your final grade depends on it. After you submit your rough draft, schedule a meeting with Yinmeng to discuss your notes.

2 Review Turing Machines

Turing machines can go backwards and forwards on the tape, as well as write to the tape and decide when to halt. Based on this idea of Turing machines comes the "Existence of the Software Industry Lemma", which states that there exist universal Turing machines that can simulate any other Turing machine by encoding a description of the machine on its tape.

The Church-Turing thesis says that Turing machines capture what we mean by the right notion of computability. Anything reasonably called a computer can be simulated by a Turing machine. However, there are limitations associated with Turing machines. For example, no Turing machine can solve the halting problem (Proof by poem). Also, the number of possible problems is far greater than the number of computer programs. Remember the infinity of real numbers versus the infinity of integers.

But who cares? That's why Turing brought up the halting problem; we actually care about that.

3 Oracles

Oracles are a concept that Turing invented in his PhD thesis in 1939. Shortly afterwards, Turing went on to try his hand at breaking German naval codes during World War II. The first electronic computers were used primarily for this purpose of cracking German codes, which was extremely difficult because the Germans would change their codes every day. The Germans never really suspected that the code had been broken, and instead thought that they had a spy among their ranks. At one point they did add settings to the code, which then set the code-breakers back about nine months trying to figure out how to break the new code.

3.1 Oracles

An *oracle* is a hypothetical device that would solve a computational program, free of charge. For example, say you create a subroutine to multiply two matrices. After you create the subroutine, you don't have to think about how to multiply two matrices again, you simply think about it as a

"black box" that you give two matrices and out comes their product.

However, we can also say that we have oracles for problems that are unsolvable or problems that we don't know how to solve. Assuming that the oracle can solve such problems, we can then create *hierarchies of unsolvability*. If given many hard problems, we can use hierarchies of unsolvability to tell us which problems are hard relative to which other problems. It can tell us things like "this problem can't be solvable, because if it were, it would allow us to solve this other unsolvable problem".

Given:

$$A : \{0,1\}^* \to \{0,1\}$$

where the input is any string of any length and the output is a 0 or 1 answering the problem, then we can write

$$M^A$$

for a Turing machine M with access to oracle A. Assume that M is a multi-tape Turing machine and one of its tapes is a special "oracle tape". M can then ask a question, some string x, for the oracle on the oracle tape, and in the next step, the oracle will write its answer, $A(x)$, on the oracle tape.

From this, we can say that given two problems A and B, A is *reducible* to B if there exists a Turing machine M such that M^B solves A. We write this as $A \leq_T B$.

3.2 Example 1: Diophantine equations

Given:

$$x^n + y^n = z^n$$

$$n \geq 3$$

$$xyz \neq 0$$

is there a solution in just integers?

In fact, there is no solution involving just integers, and this was an unsolved problem for 350 years. What if we had an oracle for the halting problem? Could we solve this problem?

To try to solve this problem, we might try every possible solution in order, (x, y, z, n) of integers:

$$\begin{aligned} x + y + z + n &= 1 \\ x + y + z + n &= 2 \\ &= 3 \\ &= ... \end{aligned}$$

and try them all, one by one in order, and then pause when we find the solution. However, if there is no solution, it would just go on forever. So our question to the oracle would be, "Does this program halt or not?", and if so, the Diophantine equation would be solvable.

If we solve the Diophantine problem, can we also then solve the halting problem?

This was an unanswered question for 70 years until 1970, when it was shown that there was no algorithm to solve Diophantine equations because if there were, then there would also be an algorithm for solving the halting problem. Therefore, the halting problem is reducible to this problem.

3.3 Example 2: Tiling the plane

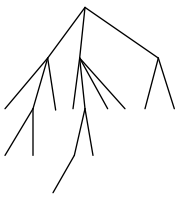
Given some finite collection of tiles of all different shapes, can we fill an entire plane just using these tiles? For simplicity, let us assume that all the tiles are 1x1 squares with different-shaped notches in the sides.

Figure by MIT OpenCourseWare.

The halting problem is reducible to this problem, which means if you could solve this problem, you could then also solve the halting problem. Why is this? Assume you could create a set of tiles that would only link together in some way that encodes the possible actions of a Turing machine. If the Turing machine halts, then you wouldn't be able to add any more tiles. Then the plane would only be tiled if the machine runs forever.

Is this problem reducible to the halting problem? In other words, if you could solve the halting problem, then can you decide whether a set of tiles will tile the plane or not? Can you tile a 100x100 grid? Can you tile a 1000x1000 grid? These questions can be answered by a Turing machine. But suppose every finite region can be filled, why does it follow that the whole infinite plane can be tiled?

In comes **König's Lemma**. Let's say that you have a tree (a computer science tree, with its root in the sky and grows towards the ground) with two assumptions:



1) Every node has at most a finite number of children, including 0.
2) It is possible to find a path in this tree going down in any finite length you want.

Claim: The tree has to have a path of infinite length.

Why is this? The tree has to be infinite because if it were finite, then there would exist a longest path. We don't know how many subtrees there are, but there are a finite number of subtrees at the top level. From that, we can conclude that one of the subtrees has to be infinite because the sum of the number of vertices in all the subtrees is infinite and there is only a finite number of subtrees

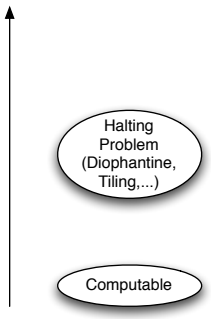
so one of the subtrees has to have infinitely many vertices.

We'll start at the top and move to whichever node has infinitely many descendants. We know there has to be at least one of them, as we just stated previously. If we repeat this, now the next subtree, by assumption, has infinitely many nodes in it with a finite number of children, and each of those children is the top of a subtree and one of its subtrees must be infinite, and we can keep going on forever. The end result is an infinite path.

So how does König's Lemma apply to the tiling problem? Imagine that the tree is the tree of possible choices to make in the process of tiling the plane. Assuming you're only ever placing a tile adjacent to existing tiles, then at each step you have a finite number of possible choices. Further, we've assumed that you can tile any finite region of the plane, which means the tree contains arbitrarily long finite paths. Therefore, König's Lemma tells us that the tree has to have an infinite path, which means that we can tile the infinite plane. Therefore, the tiling problem is reducible to the halting problem.

3.4 Turing degrees

A *Turing degree* is a maximal set of all problems that are reducible to each other. For example, we have so far seen two examples of Turing degrees: computable problems and problems equivalent to the halting problem.

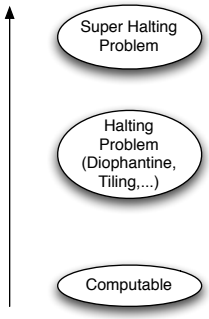


Is there any degree above the halting problem? In other words, if we were given an oracle for the halting problem, is there any problem that would still be unsolvable?

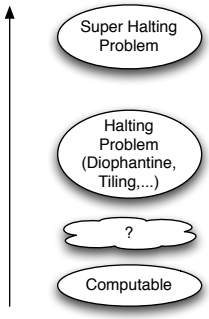
How about if we took a Turing machine with an oracle for the halting problem and asked, "Do you halt?" This can be called the "Super Halting Problem"! To prove this, we can repeat Turing's original halting problem proof, but just one level higher, where all the machines in the proof have this oracle for the halting problem. As long as all of the machines in the proof have the same oracle, then nothing changes and this problem is still the same as the original halting problem. We can follow each machine step by step and ask the oracle, but the oracle won't have the answer for the Super Halting Problem.

If there were the Super Turing Machine that could solve the Super Halting Problem, then you could feed that Super Turing Machine itself as input and cause it to do the opposite of whatever it does, just like an ordinary Turing Machine.

From this, it is clear that we could just keep going up and up with harder and harder problems, the Super Duper Halting Problem, the Super Duper Duper Halting Problem...



Is there any problem that is in an "intermediate" state between computable and the halting problem? In other words, is there a problem that is 1) not computable, 2) reducible to the halting problem, and 3) not equivalent to the halting problem?



This was an open problem called "Post's Problem". In 1956, it was solved by using a technique called the "priority method", which considers all possible Turing machines that might solve a problem A, and all possible Turing machines that might reduce the halting problem to A, and then constructs A in a complicated way that makes all of these machines fail. The problem A that you end up with is not something ever likely to occur in practice! But it does exist.

4 Gödel's Incompleteness Theorem

Gödel's Theorem is a contender (along with quantum mechanics) for the scientific topic about which *the most crap has been written*. Remember systems of logic, containing axioms and rules of inference. You might hope to have a single system of logic that would encompass all of mathematics. In 1930, five years before Turing invented Turing machines, Gödel showed that this was impossible. Gödel's theorems were a direct inspiration to Turing.

Gödel's Incompleteness Theorem says two things about the limits of any system of logic.

First Incompleteness Theorem: Given any system of logic that is consistent (can't prove a contradiction) and computable (the application of the rules is just mechanical), there are going to be true statements about integers that can't be proved or disproved within that system. It doesn't mean these statements are unprovable, but if you want to prove them, you need a more powerful system, and then there will be statements within *that* system that can't be proved, and so on.

Second Incompleteness Theorem: No consistent, computable system of logic can prove its own consistency. It can only prove its own consistency if it is inconsistent. Kind of like how people who brag all the time tend to have nothing to brag about.

(Technical note: Gödel's original proof only worked for a subset of consistent and computable systems of logic, including those that are *sound* and computable. Here "sound" means "unable to prove a falsehood." Soundness is a stronger requirement than consistency. On the other hand, it's also what we usually care about in practice. A later improvement by Rosser extended Gödel's Theorem to all consistent and computable systems, including those that are not sound.)

How did Gödel prove these theorems? He started out with the paradox of the liar: "This sentence is not true." It can't be either true or false! So if we're trying to find an unprovable statement, this seems like a promising place to start. The trouble is that, if we try to express this sentence in purely mathematical language, we run into severe problems. In particular, how do we define the word "true" mathematically?

Gödel's solution was to replace "This sentence is not true" with a subtly different sentence: "This sentence is not *provable*." If the sentence is false, it means that the sentence is provable, and is therefore a provable falsehood! That can't happen if we're working within a sound system of logic. So the sentence has to be true, but that means that it isn't provable.

Gödel showed that as long as the system of logic is powerful enough, you *can* define provability within the system. For, in contrast to truth, whether or not a sentence is provable is a purely "mechanical" question. You just have to ask: starting from the axioms, can you derive this sentence by applying certain fixed rules, or not? Unfortunately for Gödel, the idea of the computer hadn't been invented yet, so he had to carry out his proof using a complicated number theory construction. Each sentence is represented by a positive integer, and provability is just a function of integers. Furthermore, by a trick similar to what we used to show the halting problem was unsolvable, we can define sentences that talk about their *own* provability. The end result is that "This sentence is not provable" gets "compiled" into a sentence purely about integers.

What about the Second Incompleteness Theorem? Given any reasonable logical system S, let

$$G(S) = \text{"This sentence is not provable in } S\text{"}$$

$$Con(S) = \text{"}S \text{ is consistent"}$$

where again, "consistent" means that you cannot prove both a statement and the negation of the statement. Consistency is also a purely mechanical notion because you could just keep turning a crank, listing more and more consequences of the axioms, until you found a statement and its negation both proved. If you never succeed, then your system is consistent. Gödel shows that no sound logical system can prove its own consistency.

The key claim is that $Con(S) \Rightarrow G(S)$. In other words, if S could prove its own consistency, then it actually *could* prove the "unprovable" Gödel sentence, "This sentence is not provable".

Why? Well, suppose $G(S)$ were false. Then $G(S)$ would be provable. But then S would be inconsistent because it would prove a falsehood! So taking the contrapositive, if S is consistent then $G(S)$ must be true. Furthermore, all of this reasoning can easily be formalized within S itself.

So if $Con(S)$ were provable in S, then $G(S)$ would also be provable, and therefore S would prove a falsehood! Assuming S is sound, the only possible conclusion is that $Con(S)$ is *not* provable in S. The system will never be able to prove its own consistency.