Lecture Notes for CS 509
Foundations of Computer Science
Lecture 4-Second Half

Lecturer: Joe Kilian
Scribe: Darakhshan J. Mir
Department of Computer Science
Rutgers, The State University of New Jersey

1 Many-one reducibility

Among *Recursively Enumerable(RE)* sets, the *Halting problem* is as hard as it gets. How do we assign a 'hardness' to this problem? What does it mean to be 'as hard as'? One way of dealing with these issues is the notion of reducibility.

Definition. Language L_1 is *many-one-reducible* to Language L_2, denoted by $L_1 \leq_m L_2$ if:

1) $x \in L_1$ iff $y \in L_2$, and
2) y can be computed from x, that is $y = f(x)$, where $f()$ is a computable function.

Theorem 1.1. Halt *is RE-complete, that is,* $\forall$ L *in* RE , $L \leq_m Halt$.

Proof. Suppose, $\exists$ M, such that if $x \in L$, $M(x)$ accepts. Consider M', which halts iff the input is x.

On input y, $M'(y)$ does the following:
Interpret M on x and
if $M(x)$ accepts, halt.
if $M(x)$ rejects, loop forever.
This would imply that $M'(< M' >)$ halts iff $M(x)$ accepts.
Therefore, $M' \in Halt$ iff $x \in L$.

□

Theorem 1.2. *If L_1 is* RE Complete *and $L_1 \leq_m L_2$ then:*
1) L_2 is RE.
2) L_2 is RE Complete.

If one wants to prove that any problem is as hard as a known problem, then one must reduce the known problem to the given problem. Notice, that is you prove the other way round, then all you have proved is that the known problem is reducible to this other problem, which doesn't lead you to anywhere useful in commenting on its 'hardness'.

Corollary 1.3. *If $L \leq_m L_1$ and $L_1 \leq_m L_2$, then $L \leq_m L_2$.*

Fact 1. Take any language L, if you can reduce any RE Complete language to L, then L is also RE Complete.

1.1 Complement of a RE

Theorem 1.4. *If L is RE and $\overline{L}$ is also RE, then L is decidable.*

Proof. To prove that L is decidable, we need to be able to say that given x , is $x \in L$ or if $x \notin L$.

Start enumerating everything in L, we can do this because L is RE. Also, start enumerating everything in $\overline{L}$. x will eventually appear in one of these lists. This means I am guranteed to terminate. □

Corollary 1.5. *If L is undecidable and $\overline{L}$ is RE, then L is not RE.*

1.2 Recognizability and provability

The notion of recognizability(or recursive-enumerability) is equivalent to provability. Note that I can write a CFG that accepts every string. However, there exists no proof for it!

If a statement is true, one might think it is always possible to prove it. However there's more than what meets the eye here. If a statement is true, it might not be provable. It was Gödel, who first proposed that truth is stronger than provability.

2 The *Busy Beaver* function

Consider all descriptions of Turing Machines, $< M >$, of length $\leq \mathbb{N}$. We see which TM halts last.Some of these TMs will go on for ever, some will halt. Of those that halt, the longest running TM is the winner and the number of steps it runs is $BB(N)$, the *Busy Beaver* function. $BB(N)$ is a fast growing function, it grows faster than any function that can be computed by a computer. Why?

2.1 If the *Busy Beaver* problem is computable, the *halting problem* is decidable.

Suppose, $BB(N)$ is computable. Suppose we are given a TM, M(its description is of length N) and someone asks us whether M will go on forever or whether it'll halt at some point in time(*The Halting Problem*). We run M for $BB(N)+1$ steps. If it halts, within this number of steps, we have an answer.If at this point M has not halted, it has crossed the Busy Beaver bound and will never halt(by definition of $BB(N)$) and we have an answer that M will never halt. Hence we have provided a decision for the halting problem. A contradiction.Hence $BB(N)$ is not computable.

One can see that $BB(N)$ is this gigantic, fast growing function. Are there functions that aren't this gigantic, but still aren't computable? See next section.

3 Another hard-to-compute function

Let $\#Halt(N)$ be the number of $< M >$(descriptions of TMs) of length N bits that halt. We claim that $\#Halt(N)$ is not computable.

We can think of this as the number of all N-bit programs; some of these go on forever, some halt. We see that the number of all N-bits programs is Σ^N(if Σ is the alphabet). It is straightforward to see that $\#Halt(N)$ can be described in N bits.

3.1 If the *#Halt(N)* function is computable, the *halting problem* is decidable.

Assume that the $\#Halt(N)$ function is computable.We list all TMs of description length N bits and, and compute $\#Halt(N)$ by noting which of these halt; let this number be x. Suppose, someone gives us a TM, M, whose description length is N-bits. We look up in our almanac and see if M is one of the x machines that halted. If not, it will never halt. Hence we always have an answer for the halting problem, which basically arises from the fact that $\#Halt(N)$ number of TMs of description length N-bits halt and the others go on forever. Hence, $\#Halt(N)$ cannot be computable.

This function actually asks Σ^N hard questions. For each of the Σ^N TMs you are actually asking whether the TM halts or not. Clearly, this is a monster function!