CSE 431 Theory of Computation **Spring 2014**

Lecture 7: April 22

Lecturer: James R. Lee *Scribe: Eric Lei*

An interesting question about Turing machines is whether they can reproduce themselves. A Turing machine cannot be be defined in terms of itself, but can it still somehow print its own source code? The answer to this question is yes, as we will see in the recursion theorem. Afterward we will see some applications of this result.

7.1 Recursion Theorem

Our end goal is to have a Turing machine that prints its own source code and operates on it. Last lecture we proved the existence of a Turing machine called $SELF$ that ignores its input and prints its source code. We construct a similar proof for the recursion theorem. We will also need the following lemma proved last lecture.

Lemma 7.1 *There exists a computable function $q : \Sigma^* \to \Sigma^*$ such that $q(w) = \langle P_w \rangle$, where P_w is a Turing machine that prints w and hats.*

Theorem 7.2 (Recursion theorem) *Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. There exists a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every w,*

$$r(w) = t(\langle R \rangle, w).$$

The theorem says that for an arbitrary computable function t, there is a Turing machine R that computes t on $\langle R \rangle$ and some input.

Proof: We construct a Turing Machine R in three parts, A, B, and T, where T is given by the statement of the theorem.

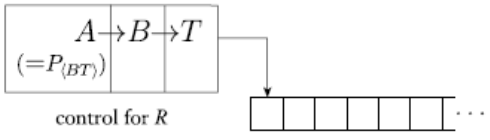

Figure 7.1: Schematic of R

Let w be the original input to R. Here A is the Turing machine $P_{\langle BT\#w \rangle}$, where # is a special character to separate the original input. Now A exists because of the lemma. After A runs, the tape contains $\langle BT \rangle \# w$.

Next B is a Turing machine that constructs the string $q(x)$ given input x, where q is from the lemma. Then it prints $\langle q(x), x \rangle$ and halts. When B runs after A, it constructs the string $\langle P_{\langle BT\#w \rangle} \rangle = \langle A \rangle$. It prints $\langle \langle A \rangle, \langle BT \rangle \# w \rangle = \langle ABT\#w \rangle = \langle R\#w \rangle$.

Finally T runs on its input, which is $\langle R\#w \rangle$ here. This completes the proof. ■

7.2 Applications

Recall that $A_{TM} = \{\langle M, w\rangle : M \text{ is a Turing machine that accepts } w\}$ is undecidable. We proved this before using Cantor diagonalization. The recursion theorem allows a simpler proof.

Theorem 7.3 *A_{TM} is undecidable.*

Proof: We prove the theorem by contradiction. Assume Turing machine A decides A_{TM}. Construct the following machine T.

$T =$ "On input w:

1. Obtain $\langle T\rangle$ using the recursion theorem.
2. Simulate A on input $\langle T, w\rangle$.
3. *Accept* if A rejects and *reject* if A accepts."

Running T on input w does the opposite of what A declares it does. Therefore, A cannot be deciding A_{TM}.

■

A second application is to show that the following language is not recognizable. If M is a Turing machine, say that M is minimal if $\langle M\rangle$ is the shortest code with M's functionality. Let $MIN_{TM} = \{\langle M\rangle : M \text{ is minimal}\}$.

Theorem 7.4 *MIN_{TM} is not recognizable.*

Proof: We prove the theorem by contradiction. Assume the language is recognizable. Then there exists an enumerator E for the language. Construct the following Turing machine T.

$T =$ "On input w:

1. Obtain $\langle T\rangle$ using the recursion theorem.
2. Simulate E until it prints $\langle D\rangle$ such that $\|\langle D\rangle\| > \|\langle T\rangle\|$.
3. Simulate D on w and output the same result as it."

Now Step 2 will always complete because the alphabet is finite and there are infinitely many Turing machines printed, so there is no maximum length of the machines printed. Then D is not minimal because T has the same functionality with shorter code. ■

7.3 Logic and Incompleteness

This section is completely unrelated to the previous sections. A logical statement is one that uses quantifiers $(\forall, \exists)$ and makes a claim about variables. Here are some examples.

Example 7.5 (infinitely many primes) $\forall q \quad \exists p \quad \forall x, y \quad (p > q \wedge x, y > 1) \implies xy \neq p$

Example 7.6 (twin primes conjecture) $\forall q \quad \exists p \quad \forall x, y \quad (p > q \wedge x, y > 1) \implies xy \neq p \wedge xy \neq p + 2$

Example 7.7 (Fermat's last theorem) $\forall a, b, c, n \in \mathbb{Z} \quad (a, b, c > 0 \wedge n > 2) \implies a^n + b^n \neq c^n$

We define the famous incompleteness theorem, which says something interesting about logical statements.

Theorem 7.8 (Godel's incompleteness theorem) *In any logical system with addition and multiplication, the system is either inconsistent, meaning it proves untrue statements, or incomplete, meaning some statements are unprovable.*