CSE 431: Introduction to Computational Theory
Lecture 6 - 4/17/14
# Reduction Proofs and the Recursion Theorem

## Reduction Proofs

We have already seen that the following language is undecidable:

$$A_{TM} = \{< M, w >: M \text{ is a TM that accepts } w\}$$

Now we will use this fact in a new method of proving undecidability in TMs, known as ***reduction*.**

**Example 1:**
Given the language,

$$HALT_{TM} = \{< M, w >: M \text{ is a TM that halts on } w\},$$

can we prove that it is undecidable?

Idea*:* We will assume that $HALT_{TM}$ is decidable then prove that $A_{TM}$ is decidable (since we know it is not, this provides the contradiction needed for the proof). This is the method of *proof by reduction*.

Proof: Assume that *N* is a TM that decides the language $HALT_{TM}$ as defined above.

*Claim: B is a TM that decides* $A_{TM}$

*B = "*On input <*M, w*>:
1. Simulate *N* on <*M, w*>
2. If *N* rejects, *REJECT*
3. Simulate *M* on *w*
4. If *M* accepts *w, ACCEPT*
5. If *M* rejects *w, REJECT"*

Here we have given a TM that decides the language $A_{TM}$, which is guaranteed to halt on all input and hence also deciding the $HALT_{TM}$ language. But we already proved that $A_{TM}$ is undecidable, so we have arrived at a contradiction and our assumption that *N* decides the $HALT_{TM}$ language is falsified, making $HALT_{TM}$ undecidable.

We will also see that TMs can create other TMs, as shown in the next example.

**Example 2:**

Given the language,

$$E_{TM} = \{< M >: M \text{ is a TM and } L(M) = \emptyset\}$$

show that it is undecidable.

Proof: Assume $Q$ is a TM that decides $E_{TM}$

*Claim: B is a TM that decides* $A_{TM}$

*B* = "On input <*M, w*>:

1. *R* = "If the input $x \neq w$, *REJECT*
   "If $x = w$, simulate *M* on *w* using the behavior of *M*"
2. Run *Q* on <*R*>
3. If *Q* accepts <*R*>, *REJECT*
4. If *Q* rejects <*R*>, *ACCEPT*"

Notes:

- *R* accepts *w* iff *M* accepts *w*
- If *M* accepts *w*, $L(R) = \{w\}$
- If *M* does not accept *w*, $L(R) = \emptyset$

As in the previous example, we have given a TM that decides the language $A_{TM}$ and so the TM *Q* will decide the language $E_{TM}$. Having already proved the undecidability of $A_{TM}$, the TM B cannot exist and we have arrived at a contradiction. The language $E_{TM}$ is then undecidable.

**Example 3:**

Given the language,

$$EQ_{TM} = \{< M, N >: M, N \text{ are TMs and } L(M) = L(N)\}$$

show that it is undecidable.

Proof: Assume A is a TM that decides $EQ_{TM}$

*Claim: B is a TM that decides* $A_{TM}$

*B* = "On input <*M*>:

1. *R* = "On input *x, REJECT*"
2. Run *A* on <*M,R*>
3. If *A* accepts, *ACCEPT*
4. If *A* rejects, *REJECT*"

Note that for step 2 here, we are deciding if $L(M) = L(R) = \emptyset$.

**Other Problems of Undecidability**

TM languages are not the only problems that can be undecidable! Consider the following example:

Given the polynomial equation $17x^3 + 256y^2 + 470z = 0$, is there some solution where $x, y, z \in \mathbb{Z}$? If we were talking about real numbers, then we could certainly find the root or at least get close enough to the root to confidently say it is the answer, but there is no way to decide if there is an integer answer!

## The Recursion Theorem

Consider the TM M which behaves as follows:

*M* = "…….

1. …….
2. …….
3. Obtain a copy of <*M*>
4. Print on the tape"

What does this mean? Ignoring whatever the rest of the TM does, in step 3 we obtain a copy of <*M*>, meaning we want to get a copy of the source code of *M*. Then on the input, we want to write that source code.

**Problem:** Design a TM *SELF* that when executed, prints <*SELF*> (its own source code).

We must be careful not to fall into an infinite loop of recursion; our initial naïve approach might be to add a print to the top of the code…but then this is changing the source code so we would have to add *another* print to the top, and so on forever!

**Definition:** A function $q: \Sigma^* \rightarrow \Sigma^*$ is <u>computable</u> if there is a TM *M* that on every input $w \in \Sigma^*$, it halts with $q(w)$ written on the tap

*Lemma:* There is a computable function $q$ such that for every input $w \in \Sigma^*$, $q(w) = < P_w >$ where $P_w$ is a TM that prints *w.*

Define a TM *Q* as follows:

*Q* = "On input *w*:

1. $P_w$ = "Erase the input
   Write $w$ on the tape
   *HALT*"
2. Erase the input
3. Write $< P_w >$ on the tape

Define <*SELF*> = <*AB*>: *A* is a TM that outputs <*B*>, and *B* outputs <*A*>

Giving the code for *A* is easy, it is simply $A = P_{<B>}$! But the code for *B* becomes a bit more complicated; we cannot refer to *A* in the code since *A* already refers to *B*.

*B* = "On input <*M*>:
1. Compute $q$(<*M*>)
2. Erase the input
3. Print <$P_{<M>}$, *M*>
4. *HALT*"

Notes:
- $q$(<*M*>) is a program that prints <*M*>
- At no point does *B* refer to *A*; this prevents the infinite recursion loop and allows us define *SELF*

The definition of *A* and *B* together make up the code of *SELF* , and when *SELF* is run, it will print its own source code using what *A* and *B* do together.