CSCI3390-Lecture 6: Undecidability Proofs; Reductions

February 8, 2016

1 Summary

- We prove the claims that all the problems about Turing machines listed in the last lecture are undecidable.
- The main tool for proving this is to reduce the problem that we want to show is undecidable to a problem we already know to be undecidable.

2 A collection of undecidable problems about Turing machines

Recall our list of problems.

2.1 Halting Problem, version 1

Input: A Turing machine $M$ and a string $w \in \{0, 1\}^*$.
Output: Yes if $M$ eventually halts when started on input $w$, No otherwise.

2.2 Halting Problem, version 2

Input: A Turing machine $M$.
Output: Yes if $M$ eventually halts regardless of the input string (i.e., $M$ is free of infinite loops), No otherwise (i.e., if $M$ runs forever on some input).
2.3 Nonemptiness
Input: A Turing machine $M$.
Output: Yes if there is some $w \in \{0, 1\}^*$ accepted by $M$, no otherwise.

2.4 Finiteness.
Input: A Turing machine $M$.
Output: Yes if the set of strings accepted by $M$ is finite, No if $M$ accepts infinitely many strings.

2.5 Equivalence
Input: Two Turing machines $M, N$.
Output: Yes $M$ and $N$ accept exactly the same strings, no otherwise.

2.6 Minimality
Input: A Turing machine $M$.
Output: Yes if $M$ is the smallest Turing machine that recognizes the set of strings it accepts, No if there is a smaller machine that recognizes the same language.

Here you have to say what exactly you mean by the size of a Turing machine. One way to do this is to fix an encoding scheme, and use the length of the encoding of $M$ as the size of $M$.

3 Reductions
The main tool for proving a given problem $Y$ is undecidable is to reduce another problem $X$ that we already know is undecidable to $Y$. In essence, this means that we show that if we had an algorithm to decide $Y$, then we would have such an algorithm for $X$ (which we already know is not possible). We sometimes say that we prove $Y$ is undecidable by a ‘reduction from $X$’. Think of it this way—if we had a $Y$ subroutine, we could use it to write an $X$ program.

The actual form these reductions take is to provide an algorithm for transforming an instance $u$ of problem $X$ into an instance $u'$ of problem $Y$ such that $u'$ is a positive instance (a Yes instance) of $Y$ if and only if $u$ is a Yes instance for $X$. 

2
Then if we had an algorithm for \( Y \) we would obtain the following algorithm for \( X \): Take an input \( u \), transform it to \( u' \) and apply the algorithm for \( Y \) to \( u' \).

Let’s see how this works in practice with our collection of problems from the last section:

### 3.1 Halting Problem, version 1

We reduce \( L_{TM} \) to this problem. Given a Turing machine \( M \), we modify it to produce a new Turing machine \( M' \) as follows: We replace the rejecting state by a new state \( q^* \) such that if \( M' \) enters \( q^* \), it runs forever. The modification is shown in Figure 1.

The result is that \( M' \) halts on an input string \( w \) if and only if \( M \) accepts \( w \), because if \( M \) rejects \( w \), \( M' \) would run forever. So if we had an algorithm for the halting problem, we would get one for \( L_{TM} \). Given an input \( (M, w) \) for \( L_{TM} \), we transform it to \( (M', w) \), and test if \( M' \) halts on \( w \). If it does, then \( M \) accepts \( w \), and if \( M' \) does not halt on \( w \), then \( M \) does not accept \( w \).

### 3.2 Halting Problem, version 2

We reduce version 1 to this problem. Given \( M, w \), we will show how to construct a Turing machine \( M' \) such that \( M' \) halts on every input if and only if \( M \) halts on \( w \). Here is how \( M' \) will work: initially, it erases its input, and then writes the

---

1 Students often get which way the reduction works backwards, and for good reason. To show \( Y \) is undecidable, we reduce the known undecidable problem \( X \) to \( Y \). But by this means, according to a more common use of ‘reduce’, we are reducing the problem of proving \( Y \) undecidable to the problem of proving \( X \) undecidable. So, yes, it’s confusing. You just have to think it through.
Figure 2: Illustration of the reduction for the second version of the halting problem, as well as the finiteness problem. The new machine begins by erasing its input and then writing the string $w$ (in this case 01) on the tape, and then running $M$ on $w$.

Figure 3: Illustration of the reduction for the nonemptiness problem. The new machine first verifies that its input is identical to $w$ (in this case 01), rejecting otherwise. It then runs $M$ on $w$.

input string $w$ on its tape—this is hard-wired into the transitions of $M'$. After this initial phase, $M'$ returns to the start of the input and does what $M$ does. This is illustrated in Figure 2 for the case $w = 01$. Observe that in contrast to the previous problem, the construction of $M'$ depends on the string $w$. So we can test if $M$ halts on $w$ by testing whether $M'$ halts on every input.

### 3.3 Nonemptiness

We reduce $L_{TM}$ to this problem. Given $M, w$ we construct a TM $M'$ that accepts the string $w$ if $M$ accepts $w$, and accepts nothing at all if $M$ does not accept $w$. So to test if $M$ accepts $w$, we construct $M'$ and test it for nonemptiness. $M'$ works as follows: In an initial scan, it checks to see if its input is equal to $w$, and if it is not, $M'$ rejects the input. Otherwise, $M'$ returns to the left end of the tape, and simulates $M$. Thus $M'$ is a Yes instance of the nonemptiness problem if and only if $M, w$ is a Yes instance of $L_{TM}$. The construction is illustrated in Figure 3 for the case $w = 01$. 
3.4 Finiteness

Again, we reduce from $L_{TM}$. We use exactly the same construction as in version 2 of the halting problem: Given $M$ and $w$, we construct a new TM $M'$ that erases its input, writes $w$ on the tape, and then simulates $M$ on $w$. Thus $M'$ accepts an infinite set of strings (in fact every string) if $M$ accepts $w$, and a finite set of strings (the empty set) if $M$ does not accept $w$.

3.5 Equivalence

Reduction from nonemptiness (or, rather, from its complement emptiness). If we could test for equivalence of two Turing machines, we could test whether a given Turing machine $M$ is equivalent to the TM $M_\emptyset$ shown in Figure 4, which recognizes the empty language. That is, we transform an instance $M$ of the emptiness problem into the instance $(M, M_\emptyset)$ of the equivalence problem. Thus if we could decide whether two TMs are equivalent, we could decide whether a given TM recognizes the empty language.

4 Reductions formally

Remember that in a more formal setting, decision problems are languages over some input alphabet. So, formally, a reduction of a (known undecidable) problem $X$ to another problem $Y$ is a reduction of a an undecidable language $L \subseteq \Sigma^*_1$ to $L' \subseteq \Sigma^*_2$. The reduction itself is a function

$$f : \Sigma^*_1 \rightarrow \Sigma^*_2$$

with the property that $w \in L$ if and only if $f(w) \in L'$. One additional, crucial requirement is that $f$ be computable. There has to be an algorithm for computing
that is, there is some $TM \mathcal{N}$ such that $f = f_{\mathcal{N}}$. We adopt our usual practice of assuming the Church-Turing thesis, and simply describe the algorithm, rather than the Turing machine $\mathcal{N}$.

For instance, the reduction for version 2 of the halting problem is officially a function that takes a string $\langle M, w \rangle \in \{0, 1\}^*$ and produces the string $\langle M' \rangle \in \{0, 1\}^*$. 