CS 3719 (Theory of Computation and Algorithms) – Lecture 5

Antonina Kolokolova[*]

January 23, 2012

1 Reductions

Now we will proceed to show that many problems are undecidable (and some of them are not semi-decidable, not co-semi-decidable or even neither semi- nor co-semi-decidabl). Rather than adapting the proof of undecidability of A_{TM} to other problems, we will use a concept which is going to be used a lot for the rest of this course: the notion of a *reduction.* A reduction is a method of "disguising" one problem as another, so if we can solve the disguised one it can give us the solution to the original. This method is very useful for proving that problems are hard: if you can disguise a hard problem as one in hand, then solving this problem is at least as hard as solving the hard one.

Definition 14. *A function $f : \Sigma^* \to \Sigma^*$ is* computable *if there is a Turing machine M that halts on every input x with $f(x)$ as its output on the tape.*

Definition 15. *Let $L_1, L_2 \subseteq \Sigma^*$. We say that $L_1 \leq_m L_2$ if there is a computable function $f : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$,*
$x \in L_1 \Leftrightarrow f(x) \in L_2$.

Here, we need f to be computable so that it always gives us an answer. The notation $\leq_m$ stands for "many-one reduction" or "mapping reduction". It is many-one since f may map many different instances of a problem to a single output.

Theorem 16. *Let $L_1, L_2 \subseteq \Sigma^*$ such that $L_1 \leq_m L_2$. Then*

1) $\overline{L_1} \leq_m \overline{L_2}$

[*]The material in this set of notes came from many sources, in particular "Introduction to Theory of Computation" by Sipser and course notes of U. of Toronto CS 364. Many thanks to Richard Bajona for taking notes!

2) If L_2 is decidable then L_1 is decidable.
(And hence, if L_1 is not decidable then L_2 is not decidable either).

3) If L_2 is semi-decidable then L_1 is semi-decidable.
(And hence, if L_1 is not semi-decidable then neither is L_2.)

Proof. **1)** Say that $L_1 \leq_m L_2$ via the computable function f. Then we also have $\overline{L_1} \leq_m \overline{L_2}$ via f, since $x \in L_1 \Leftrightarrow f(x) \in L_2$ implies that $x \in \overline{L_1} \Leftrightarrow f(x) \in \overline{L_2}$.

2) Say that $L_2 = \mathcal{L}(M_2)$ where M_2 is a Turing machine that halts on every input. Let M be a Turing machine that computes f. We now define Turing machine M_1 as follows.
On input x, M_1 runs M on x to get $f(x)$, and then runs M_2 on $f(x)$, accepting or rejecting as M_2 does. Clearly M_1 halts on every input, and $L_1 = \mathcal{L}(M_1)$, so L_1 is decidable.

3) Say that $L_2 = \mathcal{L}(M_2)$ where M_2 is a Turing machine. Let M be a Turing machine that computes f. We now define Turing machine M_1 as follows.
On input x, M_1 runs M on x to get $f(x)$, and then runs M_2 on $f(x)$, accepting or rejecting as M_2 does if and when M_2 halts. Clearly $L_1 = \mathcal{L}(M_1)$, so L_1 is semi-decidable. □

Question: is it true that $A_{TM} \leq_m \overline{A_{TM}}$? The answer is No: if it were true, then by the first property above we would have $\overline{A_{TM}} \leq_m A_{TM}$. But by 3) above, that would mean that $\overline{A_{TM}}$ is semi-decidable. But if both a language and its complement are semi-decidable, then, as we saw in the last class, the language would have to be decidable – which is a contradiction, since A_{TM} is undecidable. Thus, A_{TM} is not reducible to A_{TM}, and in fact, it is not reducible to any co-semi-decidable language. So please don't make a mistake of assuming that you always prove undecidability by reducing A_{TM} to a problem in hand – sometimes you have to use $\overline{A_{TM}}$, if the problem you are working with is co-semi-decidable. Or, conceptually easier, if the problem in hand is co-semi-decidable, then work with its complement all the way.

Now we can use this notion of reduction to prove that some languages are undecidable by reducing languages for which we already know that (such as A_{TM}) to them.

Example 1. Let $HaltB = \{\langle M\rangle |$ TM M halts on blank input $\}$. We will show that $A_{TM} \leq_m$ HaltB
Let $x \in \Sigma^*$, and assume that $x = \langle M, w\rangle$ where M is a Turing Machine.
(If x is not of this form, then we can let $f(x)$ be anything not in HaltB. In general we will assume that the input is "well-formed" since this will always be easy to test for.)

We will let $f(x) = \langle M'\rangle$ where M' works as follows on a blank tape (we don't care what M' does on a non-blank tape).

```
$M'$ :on input $x'$
    write $w$ on the tape
    simulate $M$ running on input $w$;
    if and when $M$ halts and accepts, $M'$ halts and accepts;
    if and when $M$ halts and rejects, $M'$ goes into an infinite loop.
```

Clearly f is computable. It is also easy to see that $x \in A_{TM} \Leftrightarrow f(x) \in \text{HaltB}$, since M accepts $w \Leftrightarrow M'$ halts on blank tape.

Corollary 17. *$\overline{HaltB}$ is not semi-decidable.*

Proof. We know HaltB is semi-decidable but not decidable, so $\overline{\text{HaltB}}$ is not semi-decidable. □

Example 2. Let NE be the language consisting of Turing Machines that accept a nonempty language. That is, $\text{NE} = \{\langle M \rangle \mid M \text{ is a Turing Machine and } \mathcal{L}(M) \neq \emptyset\}$.

Lemma 18. *NE is semi-decidable, but not decidable. (Hence, $\overline{NE}$ is not semi-decidable.)*

Proof. We will design a Turing Machine M_{NE} such that $\text{NE} = \mathcal{L}(M_{NE})$ M_{NE} behaves as follows:

```
$M_{NE}$: on input $x$
      if $x$ is not of the form $\langle M \rangle$, reject
      for $i = 1$ to $\infty$
          run $M$ on all inputs of length $\leq i$ for $i$ steps;
          if and when it is discovered that $M$ accepts some input, $M_{NE}$ halts and accepts.
```

That is, M_{NE} does the following (let $\Sigma = \{0, 1\}$): run M on all inputs of length ≤ 1 for 1 steps; if M accepted ϵ or 0 or 1 in one transition then accept; otherwise
run M on all inputs of length ≤ 2 for 2 steps; if M accepted one of $\epsilon, 0, 1, 00, 01, 10, 11$ then accept; otherwise
run M on all inputs of length ≤ 3 for 3 steps; etc.

Clearly $\text{NE} = \mathcal{L}(M_{NE})$, so NE is semi-decidable.

To show that NE is not decidable we will prove that $\text{HaltB} \leq_m \text{NE}$.
Let x be an input for HaltB, and assume that x is well-formed, that is, $x = \langle M \rangle$.
Define $f(x) = \langle M' \rangle$ where M' works as follows.

```
$M'$ on input $x$
    erase $x$ and run $M$ on the blank tape;
    if and when $M$ halts, $M'$ halts and accepts.
```

Clearly $\langle M \rangle \in \text{HaltB} \Leftrightarrow \langle M' \rangle \in \text{NE}$, so $\text{HaltB} \leq_m \text{NE}$, so NE is not decidable.

This type of reduction, doing the same on all inputs can help with a surprising number of problems. I like to call them "All-or-nothing" reductions. The resulting language is either Σ^* (and thus includes every subset one might be interested in) or $\emptyset$. $\square$