CS 3719 (Theory of Computation and Algorithms) – Lecture 6

Antonina Kolokolova[*]

January 25, 2012

0.1 Examples of reductions

Recall that $A \leq_m B$ iff there exists a computable function f such that $\forall x \in \Sigma_A^*$ $x \in A$ iff $f(x) \in B$. The notation suggests that "A is at most as hard to solve as B". Often we use the reduction to prove hardness for problems of comparable complexity, but sometimes it is not the case: A can be a lot simpler than B.

Example 1. $\{uu | u \in \{0,1\}^*\} \leq_m A_{TM}$
We need to define $f(x) =< M, w >$. Take a Turing machine, say, which accepts an empty string and rejects everything else. So the description of M is simple: $Q = \{q_0, q_{accept}, q_{reject}\}$, $\Sigma = \{0,1\}$, $\Gamma = \{0, 1, \sqcup\}$ $\delta = \{(q_0, 0) \rightarrow (q_{reject}, 0, R), (q_0, 1) \rightarrow (q_{reject}, 0, R), (q_0, \sqcup) \rightarrow (q_{accept}, 0, R)$. Now, define $f(x) =< M, \epsilon >$ if $x = uu$ for some $u \in \{0,1\}^*$ and $f(x) =< M, 0 >$ otherwise. This is a computable function, and it has the desired property that $x \in \{uu | u \in \{0,1\}^*\} \iff f(x) \in A_{TM}$.

Similarly, for the rest of this lecture we will use reductions to show that certain problems are even harder than ones we encountered so far. Recall that semi-decidable languages are ones for which there is a Turing machine which halts (and accepts) on all strings in the language; for co-semi-decidable languages, there is a Turing machine halting on all inputs not in the language. However, there are some languages which are neither semi-decidable nor co-semi-decidable, but belong higher in arithmetic hierarchy, as it is called. The best way to think about them is using quantifiers: semi-decidable languages correspond to an existential quantifier (or several existential quantifiers) over a decidable predicate (e.g., "exists a string on which there exists an accepting computation" – here, the decidable predicate is the check that the existential quantifiers indeed guessed a string and a correct computation of this

[*]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 and SFU CS 710. Special thanks to Richard Bajona for scribing

Turing machine on this string ending in an accept state). Similarly, a co-semi-decidable language can be described using a universal quantifier, just by negating a formula describing the language (e.g. "for any string any computation (finite and correct) is not accepting") . The languages beyond semi-decidable and co-semi-decidable are, thus, described using a combination of quantifiers. The number of quantifier alternations corresponds to the levels of this (strict) hierarchy.

In this lecture we will only talk about languages described with just one quantifier alternation. But already in this case we cannot talk at all about a Turing machine corresponding to this language (or its complement).

Example 2. Let $L_{01} = \{\langle M \rangle \mid M \text{ loops on 0 and accepts 1}\}$. Note that the description of this language has both a universal quantifier ("loop"= "all finite computations are wrong") and an existential quantifier ("accept" = "exists correct accepting computation").

To prove that this language is neither semi-decidable nor co-semi-decidable we will use two reductions. Let us use A_{TM} as the "hard problem". To show that L_{01} is not co-semi-decidable, we will reduce a not co-semi-decidable A_{TM} to L_{01}. Then, to show that L_{01} is not semi-decidable we will reduce a non-semi-decidable $\overline{A_{TM}}$ to L_{01}.

First we will show that L_{01} is not co-semi-decidable by showing $A_{TM} \leq_m L_{01}$. For that, by definition of reduction, we will describe a computable function $f(\langle M, w \rangle) = \langle M' \rangle$ that for any pair M, w constructs M' such that M accepts w if and only if M' loops on 0 and accepts 1. Since we are reducing a semi-decidable language (existential quantifier), we will force M' to always loop on 0, and will make its behaviour on 1 depend on whether M accepts w.

```
M': on input $x$
    for $x \neq 0, x \neq 1$ it does not matter what $M'$ does. It could accept, or run $M$ on $w$, anything. Say $M'$
    if $x = 0$ then loop
    if $x = 1$ then run $M$ on $w$.
    if $M$ accepts, accept. If $M$ rejects, loop (here, reject would also be correct).
```

Thus, M' always loops on 0, and accepts 1 if and only if M accepts w, just as we wanted. This reduction shows that L_{01} is at least as hard as A_{TM}, in particular, since A_{TM} is not co-semi-decidable, then neither is L_{01}.

It remains to show that L_{01} is not semi-decidable. We will do it by reduction $\overline{A_{TM}} \leq_m L_{01}$. That is, we will construct a computable function f which on input $\langle M, w \rangle$ produces M' which now will loop on 0 and accept 1 if and only if M does not accept w. Another technicality is that since we are talking about complement of A_{TM}, the language we are reducing <u>from</u> contains all the "garbage" – strings that do not encode Turing machines. That is, $\overline{A_{TM}} = \{s | s \neq \langle M, w \rangle or s = \langle M, w \rangle$ and M does not accept $w\}$. So $f(s)$, for $s \neq \langle M, w \rangle$ would output something in L_{01}: for example a description of a Turing machine with transitions $(q_0, 1) \rightarrow (q_a ccept)$ and $(q_0, 0) \rightarrow (q_{loop}, 0, R)$ where all transitions from q_{loop} go to q_{loop}. This machine accepts 1 and loops on 0.

```
M': on input $x$
    for $x \neq 0, x \neq 1$ it does not matter what $M'$ does. It could accept, or run $M$ on $w$, anything. Say $M'$
    if $x = 1$ then accept
    if $x = 0$ then run $M$ on $w$.
    if $M$ accepts, accept (reject is OK, just don't loop). If $M$ rejects, loop (here, it has to be loop).
```

Example 3. Let T be the language of "total" machines, that is, of machines that halt on every input. $\mathrm{T} = \{\langle M \rangle \mid M$ is a Turing Machine that halts on every input$\}$.

Lemma 19. *Neither T nor $\overline{T}$ is semi-decidable.*

Proof. We first show that $\mathrm{HaltB} \leq_m \mathrm{T}$, implying that $\overline{\mathrm{HaltB}} \leq_m \overline{\mathrm{T}}$, implying that $\overline{\mathrm{T}}$ is not semi-decidable.

Let $f(\langle M \rangle) = \langle M' \rangle$ (and if input to f is not a proper encoding of a Turing machine, it is an M' that just loops on every input). We will do an "all-or-nothing" reduction again:

```
$M'$ : on input $x$
    erase input and run $M$ on the blank tape.
    if $M$ accepts, accept. If $M$ rejects, reject.
```

We have M halts on the blank tape $\Leftrightarrow$ M' halts on every input, so we are done.

We next show that $\mathrm{HaltB} \leq_m \overline{\mathrm{T}}$, implying that $\overline{\mathrm{HaltB}} \leq_m \mathrm{T}$, implying that T is not semi-decidable. This reduction is a bit tricky.
Assume that the input for HaltB, and assume is well-formed: $\langle M \rangle$.
Let $f(\langle M \rangle) = \langle M' \rangle$ where M' is as follows.

```
$M'$ :On input $x$
    Simulate $M$ on the blank tape for $|x|$ steps;
    if $M$ halts within $|x|$ steps, then $M'$ goes into an infinite loop;
    if $M$ doesn't halt within $|x|$ steps, then $M'$ halts (and, say, accepts).
```

Clearly M halts on the blank tape $\Leftrightarrow$ M' is not a total machine. □

Example 4. Let $\mathrm{All} = \{\langle M \rangle \mid M$ is a Turing Machine that accepts every input$\}$.

We will show in one shot that *All* is neither semi-decidable nor co-semi-decidable by reducing T to it: $T \leq All$. Now, $f(\langle M \rangle) = \langle M' \rangle$ where M' is the same as M except every occurrence of q_{reject} in M is changed to q_{accept}.