# CS 3719 (Theory of Computation and Algorithms) – Lecture 19

Antonina Kolokolova*

February 18, 2011

## 1 Closure properties of semi-decidable languages

Recall that the class of regular languages is closed under union, intersection, complementation, concatenation and so on, but CFLs are only closed under some of these operations (union and concatenation, but not intersection or complementation). What would be the case for semi-definite languages? Here we will show that they are closed under union; moreover they are also closed under intersection, however complementation may create a non-semi-decidable language.

**Theorem 16.** *The class of semi-decidable languages is closed under union and intersection operations.*

*Proof.* Let $L_1$ and $L_2$ be two semi-decidable languages, and let $M_1, M_2$ be Turing machines such that $\mathcal{L}(M_1) = L_1$ and $\mathcal{L}(M_2) = L2$. We will construct Turing machines $M_{L_1 \cup L_2}$ and $M_{L_1 \cap L_2}$ accepting union and intersection of $L_1$ and $L_2$, respectively.

Consider the union operation first; intersection will be similar. Let $x$ be the input for which we are trying to decide whether it is in $L_1 \cup L_2$. The first idea could be to try to run $M_1$ on $x$, and if it does not accept, then run $M_2$ on $x$. But $M_1$ is not guaranteed to stop on $x$, and we would still like to accept $x$ if $M_2$ accepts it. So the solution is to run $M_1$ and $M_2$ in parallel, switching between executing one or the other. If at some point in the computation either $M_1$ or $M_2$ accepts, we accept; if neither accepts, can run forever – but this is OK, because if neither $M_1$ nor $M_2$ accepts $x$ then $x \notin L_1 \cup L_2$. So we define $M_{L_1 \cup L_2}$ as follows:

$M_{L_1 \cup L_2}$ : On input $x$
For$i = 1$ to $\infty$
Run $M_1$ on $x$ for $i$ steps. If $M_1$ accepts, accept.

*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.

Run $M_2$ on $x$ for $i$ steps. If $M_2$ accepts, accept.

The intersection, in this case, is very similar. The only difference is that we accept at stage $i$ if not just one, but both $M_1$ and $M_2$ accepted in $i$ steps.

$\square$

**Corollary 17.** *$\overline{A_{TM}}$ is not semi-decidable. Moreover, complement of any semi-decidable, but undecidable language is not semi-decidable.*

*Proof.* Otherwise, running Turing machines $M_{A_{TM}}$ and $M_{\overline{A_{TM}}}$ simultaneously, as in the proof above, we could decide $A_{TM}$. Same holds for any semi-decidable, but undecidable language. $\square$

This shows that the class of semi-decidable languages is different (incomparable) from the class of co-semi-decidable ones. Also, there are languages that are neither semi-decidable nor co-semi-decidable. For example, consider a simple language $0-1A_{tm} = \{< M, w > \mid \text{TM } M \text{ accepts } 01 \text{ and loops on } 1w\}$.

Intuitively, testing if $< M, w >$ is in the language requires solving an $A_{TM}$ problem and a $\overline{A_{TM}}$ problem. The first one makes it not co-semi-decidable, the second not semi-decidable.

To make this intuition formal, we need 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 18.** *Let $L_1, L_2 \subseteq \Sigma^*$ such that $L_1 \leq_m L_2$. Then*

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

*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. □

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 $Regular_{TM} = \{< M > | M \text{ is a Turing machine and } \mathcal{L}(M) \text{ is regular}\}$. We can show that this language is undecidable using a reduction $A_{TM} \leq_m Regular_{TM}$. Assume, for simplicity, that $\Sigma = \{0, 1\}$.

We define the reduction function $f$, $f(< M, w >=< M' >$ such that $< M, w >\in A_{TM}$ if and only if the language of $M'$ is regular. where $M'$ is:

$M'$ : On input $x$
  If $x$ is of the form $0^n 1^n$, accept
  Otherwise, run $M$ on $w$, if $M$ accepts $w$, accept. If $M$ rejects $w$, reject.

Now, suppose $M$ accepts $w$. Then $\mathcal{L}(M') = \Sigma^*$: all strings are accepted. This language is definitely regular. Suppose now that $M$ does not accept $w$. Then the only strings accepted by $M'$ are of the form $0^n 1^n$, so $\mathcal{L}(M') = \{0^n 1^n | n \in \mathbb{N}\}$ which is not regular.