CS 6743 **Lecture 5** **Fall 2007**

1 Examples for the assignment

Example 1 *(Turing machine with stay-put).* Consider a variant of Turing machines that has, in addition to "L" and "R" directions, also a "S" ("stay in place"). That is, it can have transitions of the form $(q_i, a) \to (q_j, b, S)$, corresponding to the Turing machine overwriting a with b, switching the state to q_j and leaving the head at the same cell.

We want to show that this kind of a Turing machine can be simulated by a usual TM. That is, we need to show how to imitate the transitions with "S" with just "L" and "R". For that we will show how to, starting with an arbitrary stay-put Turing machine M, build a usual Turing machine M' accepting the same language as M.

The idea is to imitate staying at the same place with a move right followed by a move left (keeping the cell to the right intact). The first attempt would be to designate a "moving state" q_s, such that from q_s on any input the Turing machine moves left. However, this would lose the information about the state in which the machine has to be when it finished the transition. Therefore, we have to introduce such q_s for every state q_j in the transition table (at least for every occurring on the right side of a transition with "S"). Therefore, set $Q' = Q \cup \{q'_j | q_j \in Q, q_j \neq q_a, q_j \neq q_r\}$. The transition table has to be modified as follows: each transition in δ of the form $(q_i, a) \to (q_j, b, S)$ changes into transitions in δ': $(q_i, a) \to (q'_j, b, R)$ and $(q'_j, c) \to (q_j, c, L)$ for every $c \in \Gamma$. So $M' = \{\Sigma, \Gamma, Q', \delta'\}$, where Σ and Γ are the same as for M.

Example 2. Let $L = \{< M >| \ M accepts some string consisting of all 0s\}$. To see that L is semi-decidable write it in the following quantified form: $\exists w \exists y R(w, y, M)$, where w is a string, y is an encoding of a computation, and R is a decidable relation consisting of conjunction of the following formulae 1) a formula checking that all bits of w are 0s 2) a formula checking that y is a valid computation of M starting with w and ending in an accept state. Checking the first condition takes $O(|w|)$ steps, checking the second condition takes $O(|y| \cdot |M|)$ steps (since for every step encoded in y the machine needs to find that transition in the description of M). Therefore, R is decidable.

To prove formally that L is semi-decidable we will first show, by reduction from A_{TM}, that L is undecidable, and then show how to semi-decide it.

First, we will show that $A_{TM} \leq_m L$. Consider a pair (M, w). We want to construct a machine M'_w such that M'_w accepts a string consisting of all 0s iff M accepts w. Here is one possible description of M'_w:

```
$M'_w$: if input $x \neq$ "000", then reject
        write $w$ on the tape
        simulate $M$ on $w$, if $M$ accepted, accept, otherwise, reject.
```

If M accepts w, then the language of M' consists of one string "000", and therefore M' is in L. If M does not accept w, then the language of M' is empty and so M' is not in L.

Now it remains to show an algorithm that accepts all M in L; that algorithm can either reject or never halt on $M \notin L$.

```
$M_L$: for $i = 1$ to $\infty$
    Run $M$ for $i$ steps on each of strings consisting of $\leq i$ 0s.
    If any accepted, accept.
```

2 NP and NP-Completeness

NP is a class of languages that contains all of **P**, but which most people think also contains many languages that aren't in **P**. Informally, a language L is in **NP** if there is a "guess-and-check" algorithm for L. That is, there has to be an efficient verification algorithm with the property that any $x \in L$ can be verified to be in L by presenting the verification algorithm with an appropriate, short "certificate" string y.

Remark: **NP** stands for "nondeterministic polynomial time", because an alternative way of defining the class uses a notion of a "nondeterministic" Turing machine. It does *not* stand for "not polynomial", and as we said, **NP** includes as a subset all of **P**. That is, many languages in **NP** are very simple.

We now give the formal definition. For convenience, from now on we will assume that all our languages are over the fixed alphabet Σ, and we will assume $0, 1 \in \Sigma$.

Definition 1. *Let $L \subseteq \Sigma^*$. We say $L \in$**NP** if there is a two-place predicate $R \subseteq \Sigma^* \times \Sigma^*$ such that R is computable in polynomial time, and such that for some $c, d \in \mathbb{N}$ we have for all $x \in \Sigma^*$*
$x \in L \Leftrightarrow$ there exists $y \in \Sigma^*, |y| \leq c|x|^d$ and $R(x, y)$.

We have to say what it means for a two-place predicate $R \subseteq \Sigma^* \times \Sigma^*$ to be computable in polynomial time. One way is to say that
$\{x; y \mid (x, y) \in R\} \in$**P**, where ";" is a symbol not in Σ.
An equivalent way is to say that
$\{\langle x, y\rangle \mid (x, y) \in R\} \in$**P**, where $\langle x, y\rangle$ is our standard encoding of the pair x, y.
Another equivalent way is to say that there is a Turing machine M which, if given $x\not{b}y$ on its input tape ($x, y \in \Sigma^*$), halts in time polynomial in $|x| + |y|$, and accepts if and only if $(x, y) \in R$.

Most languages that are in **NP** are easily shown to be in **NP**, since this fact usually follows immediately from their definition,

Example 3. A *clique* in a graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that there is an edge between every pair of vertices in S. That is, $\forall u, v \in V(u \in S \wedge v \in S \rightarrow E(u, v))$. The language $CLIQUE = \{< G, k > | G$ is a graph, $k \in N$, G has a clique of size $k\}$. Here, we take $n = |V|$, so the size of the input is $O(n^2)$.

We will see later that this problem is NP-complete. Now we will show that it is in NP.

Suppose that $< G, k > \in CLIQUE$. That is, G has a set S of vertices, $|S| = k$, such that for any pair $u, v \in S$, $E(u, v)$. Guess this S using an existential quantifier. It can be represented as a binary string of length n, so its length is polynomial in the size of the input. Now, it takes k^2 checks to verify that every pair of vertices in S is connected by an edge. If the algorithm is scanning E every time, it takes $O(n^2)$ steps to check that a given pair has an edge between them. Therefore, the total time for the check is $k^2 \cdot n^2$, which is quadratic in the length of the input (since E is of size n^2, the input is of size $O(n^2)$ as well).