

Homework 2 Solutions, CS 132, Winter 2007

January 25, 2007

1. (9.33 part a) *Suppose L is accepted by a TM T . Describe how you could construct a nondeterministic TM to accept the set of all prefixes of elements of L .*

Our new NTM will do the following:

1. Go to the end of the input.
 2. Choose nondeterministically:
 - (a) Write a symbol in Σ , go right, and repeat 2, OR
 - (b) Go to beginning of tape and execute T .
2. (9.40 part b) *For any $n \geq 0$ and any input of length n , T begins by making $n+1$ moves in which the tape head is moved right each time, and thereafter T does not move the tape head to the left of square $n+1$. Show that the language accepted by the TM T is regular.*

At a high level the idea is this: a TM has only two kinds of memory—the tape and its state. Since the TM has can't revisit the input, after looking at it once it will be in one of a finite number of states. Though the remaining tape is infinite, whether or not the TM will accept is completely determined by it's state at cell $n+1$ so there is no point in use for the rest of the tape. Since writing to the tape has no meaningful effect (nothing can be re-read), we can get rid of Γ , the tape alphabet. Additionally since we no longer move past cell $n+1$, we will always go right, and there is no need to specify a direction with each transition.

Specifically, we can transform any TM $T = (Q, \Sigma, \Gamma, q_0, \delta)$ that has this restriction to an NFA $M = (Q', \Sigma, q_0, \{h_a\}, \delta')$, such that $Q' = Q \cup \{h_a\}$. No change to Σ , or q_0 is required. $\delta' : Q' \times \Sigma \rightarrow Q'$ will be created from $\delta : Q \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{R, L, S\}$ by removing all transitions on input symbols in $\Gamma - \Sigma$, and dropping the output symbol and direction from δ .

3. (10.3) *Is the following statement true or false? If L_1, L_2, \dots are recursively enumerable subsets of Σ^* , then $\bigcup_{i=1}^{\infty} L_i$ is recursively enumerable. Give reasons for your answer.*

False. Let $L = \{x_1, x_2, \dots\}$ be a language that is not recursively enumerable. By itself, each word in L , is a recursively enumerable language, namely $\{x_i\}$ (indeed these languages are regular, they have only one string); but by assumption $\bigcup_{i=1}^{\infty} x_i = L$ is uncomputable.

4. (10.6) *Suppose $L \subseteq \Sigma^*$. Show that L is recursively enumerable if and only if there is a computable partial function from Σ^* to Σ^* that is defined precisely at the points of L .*

By definition a computable function has a TM that accepts the strings for which it is defined, therefor making these string recursively enumerable. (see page 328-9).

If T is a TM accepting L then we can create a T' that computes $f : \Sigma^* \rightarrow \{1\}$, a function that is defined only at the points of L . To do this we need only run T (within brackets), and then erase the used tape (within and including the brackets) and return the head to the first cell before entering the state h_a so that when T' accepts, it does so in the configuration $(h_a, \underline{\Delta}1)$.

Note that since Δ may be in the tape alphabet, erasing the tape is tricky. To erase the tape we will have to put bracket characters around our input so we know how much tape to erase, and modify T so that the right bracket is shifted right whenever it is encountered.

5. (10.34 extra credit) *Suppose L is recursively enumerable but not recursive. Show that if T is a TM that accepts L , there must be infinitely many input strings for which T loops forever.*

Suppose there is a recursively enumerable but not recursive language L accepted by T , and T infinitely loops only on the strings L_{loop} , a finite subset of L . Since L_{loop} is finite, it is regular and there is a DFA that accepts it. We can now construct a TM, T' that first simulates this DFA on the input. If the DFA accepts T' accepts; if not it runs T on the input, accepting if and only if T does. Clearly T' will never loop, and so L is recursive; but by assumption L is not recursive. This is a contradiction, so no such L is possible.