Honors Theory, Spring 2002, Yap
Homework 2
Due: Wed Feb 13 in class

MOSTLY ABOUT CONTEXT FREE LANGUAGES.

1. No credit work, as this exercise is self-rewarding :-). Recall the example of translating "`The spirit is willing but the flesh is weak`" into Russian, and back again, producing "`The vodka is strong but the meat is rotten`". The sentence "`Out of sight, out of mind`" was translated into "`Blind idiot`". Try to outdo these machine translations produced in the early days of MT. [It only goes to prove that this problem is too hard for machines.]

2. (10+15 Points)
For $n \geq 1$, let $\Sigma_n = \{a_1, \ldots, a_n\}$ be an alphabet with n letters. Define $L_n \subseteq \Sigma_n^*$ to comprise those words w for which there is some $i = 1, \ldots, n$ such that a_i does not occur in w.
(i) Show that there is an nfa N_n with $n+1$ states that accept L_n.
(ii) Show that every dfa tha accepts L_n has at least 2^n states. HINT: why 2^n?

3. (15+10 Points)
Let $A, B \subseteq \Sigma^*$. The **right quotient** of A by B is defined to be
$$A/B := \{w \in \Sigma^* : (\exists u \in B)[wu \in A]\}.$$
(i) Show that if A is context free and B is regular, then A/B is context free.
(ii) Use part (i) to show that the language $\{0^p 1^n : p \text{ is prime}, n > p\}$ is not context free.

4. (10 Points Each) Prove or disprove that the following are context free:
(i) $L_1 = \{w \in \{a,b,c\}^* : \#_a(w) = \#_b(w) \text{ or } \#_b(w) = \#_c(w) \text{ or } \#_a(w) = \#_c(w)\}$.
(ii) $L_2 = \{w \in \{a,b,c\}^* : \#_a(w) \neq \#_b(w) \text{ or } \#_b(w) \neq \#_c(w) \text{ or } \#_a(w) \neq \#_c(w)\}$.
(iii) $L_3 = \{w \in \{a,b,c\}^* : \#_a(w) = \#_b(w) \text{ and } \#_b(w) = \#_c(w) \text{ and } \#_a(w) = \#_c(w)\}$.
NOTE: $\#_a(w)$ counts the number of occurences of a in w.

5. (15+15 Points) (i) Construct an efficient algorithm that, on input $\langle G, w\rangle$ where $G = (V, T, S, R)$ is a grammar in Chomsky Normal Form and w a string, decide whether $w \in L(G)$.
HINT: Use dynamic programming. For $1 \leq i \leq j \leq n$, let w_{ij} denote the substring $a_i \cdots a_j$ where $w = a_1, \ldots, a_n$. Define $V_{ij} = \{A \in V : A \Rightarrow^* w_{ij}\}$. How do you compute V_{ij} if you know the sets V_{ik}, V_{kj} for all $k = i, \ldots, j$?
(ii) What is the worst-case complexity of your algorithm, as a function of input sizes $m = |G|, n = |w|$? There is an interesting low-level issue here: $|w|$ and $|G|$ must be suitably interpreted. Note that V, T are arbitrary alphabets, but your algorithm must accept input with a fixed alphabet (say Σ). Hence use the following convention: assume Σ contains the special symbols $A, a, 0, 1$ (among others), and each symbol of V is encoded as a string of the form $A(0+1)^*$, and each symbol of T and w is encoded as a string of the form $a(0+1)^*$. The definition of $|G|$ can be taken to be the number of symbols in writing down all the rules of G plus $|V \cup T|$. Then each symbol in x has length equal to its encoding as a string in $L(a(0+1)^*)$! Similarly, you need specify your encoding G over the fixed alphabet Σ and tell us how to determine its length $|G|$.