The First Prelim Exam will be held in class period on Tuesday, April 19. It will have the same rules as Prelim I. It will cover up to chapter 3, focusing mostly on sections 2.1 and 2.3 and the auxiliary notes on induction proofs. Conversion “all the way” to Chomsky normal form will not be asked on this or the final—just as far as eliminating the $\epsilon$-rules and the unit rules.

Reading: Tuesday’s lecture will finish example(s) Section 2.3, skip section 2.4, and move into Chapter 3. It is good to read all of Chapter 3 “in one gulp” the first time, then focus on certain parts when the lectures hit them. In particular, I will wish to mention Turing machines with more than one tape fairly early because a pushdown automaton (PDA, whose special-purpose formalism I avoided introducing in section 2.2) is equivalent to a restricted kind of 2-tape TM. To wit, a [nondeterministic] PDA is equivalent to a [nondeterministic] 2-tape TM whose input tape is read-only and makes no left (“L”) moves, and whenever the initially-empty second tape moves left it must write a blank character. The latter rule makes the second tape behave exactly like a stack.

(1) (24 pts.) Say that a context-free grammar $G = (V, \Sigma, R, S)$ is in “short form” if for each $A \in V$, the rules for the variable $A$ collectively have no more than two occurrences of variables total on all their right-hand sides. This form allows $A \rightarrow \epsilon$ freely and allows unit rules $A \rightarrow B$ (but at most two of them for each $A$) unlike Chomsky normal form. For argument’s sake we’ll fix $\Sigma = \{0, 1\}$ for this problem.

Prove that for every regular language $L$ there is a CFG $G$ in short form such that $L = L(G)$. Prove this by structural induction on the following grammar $G_{REC}$ that generates regular expressions $R$:

$$E \rightarrow \emptyset \mid \epsilon \mid 0 \mid 1 \mid (E \cup E) \mid (E \cdot E) \mid (E^*)$$

Put another way, describe and verify an algorithm to convert any regular expression $R$ into a short-form CFG $G$ such that $L(G) = L(R)$, so that the algorithm works by recursion on sub-expressions. That way your answer should follow the same structure as the conversion from regexps to equivalent NFAs shown in class.

(2) (21 pts. total) Consider the following context-free grammar $G$:

$$S \rightarrow AC \mid DC$$
$$A \rightarrow aS \mid BA$$
$$B \rightarrow \epsilon \mid SCS$$
$$C \rightarrow BD \mid AS$$
$$D \rightarrow BB \mid b$$

(a) Find the whole set $N$ of nullable variables. Then carry out the step that adds rules skipping any subset of occurrences of nullable variables to get a new grammar $G_1$. Note that if $S$ is nullable then you get $L(G_1) = L(G) \setminus \{\epsilon\}$, else you get $L(G_1) = L(G)$. (Do not do the text’s initial step of adding a new start variable $S_0$. 6 + 6 = 12 pts.)
(b) Your grammar $G_1$ will have several unit rules—but don’t include “self-rules” like $A \to A$. Draw a directed graph whose nodes are the five variables and which has an edge $(A, B)$ if $A \to B$ is a unit rule. Then take the transitive closure of the graph, which will tell you all pairs $(A, B)$ such that $A \Rightarrow^* B$. Here we still ignore self-loops; that is, we only consider $B \neq A$. (6 + 6 = 12 pts.)

(c) Show the grammar $G_2$ that you get upon making all right-hand sides of rules for $B$ become right-hand sides of rules for $A$ whenever $A \Rightarrow^* B$, then finally deleting all the unit rules. (6 pts.)

(d) Convert $G_2$ all the way into a grammar $G_3$ in Chomsky normal form, such that $L(G_3) = L(G)$. (−9 pts.; sorry for the answer being unspeakably ugly compared to the original $G$, really like the text’s example in that regard.)

As usual, if you skip any part of a problem, your score is figured “out of” the total points stated for the problem at the top, not just the points for the parts you attempted. We try to give partial points for scratchwork, but you can put a note asking that it not be counted.

3 (18 pts.) Let $\Sigma = \{a, b\}$, and let $L$ be the language of palindromes over $\Sigma$ that have twice as many $a$’s as $b$’s. That is,

$$L = \{x \in \Sigma^* : x = x^R \land \#a(x) = 2 \cdot \#b(x)\}.$$ 

Prove via the CFL Pumping Lemma that $L$ is not a context-free language. (Hint: Try $x$ of the form $a^p b^p a^p$. 18 pts., making the whole problem set “out of” 63 pts.)