Harvard University
Computer Science 121

Problem Set 4

Due Tuesday, October 8, 2013 at 11:59 PM.
Submit your solutions electronically to cs121+ps4@seas.harvard.edu with "ps4 submission" in the subject line. The solutions to each part should be attached as separate PDF files, called lastname+ps4a.pdf, lastname+ps4b.pdf.
Late problem sets may be turned in until Friday, October 11, 2013 at 11:59 PM with a 20% penalty.

Problem set by ** ENTER YOUR NAME HERE **

Collaboration Statement: **FILL IN YOUR COLLABORATION STATEMENT HERE (See the syllabus for information)**

See syllabus for collaboration policy.

PART A (Graded by Joy and Francisco)

PROBLEM 1 (2+4 points)

For this problem, you may assume that we are working over the alphabet $\Sigma = \{a, b\}$.
(A) Give a simple English description of the language generated by the following context-free grammar (you do not need to justify your description):

$$S \to \epsilon \mid aY \mid bX \mid XY$$

$$X \to \epsilon \mid bX \mid Xa$$

$$Y \to \epsilon \mid aY \mid Yb.$$

(B) Give a context-free grammar for the following: $L = \{w : w \text{ has twice as many } a\text{'s as } b\text{'s}\}$.

(A) The set of strings with at most one block of consecutive a's.

(B)

$$S \to SS \mid aaSb \mid bSaa \mid aSbSa \mid \epsilon$$

PROBLEM 2 (6 points)

Draw the state diagram for a PDA for the language in Problem 1B. Use the state diagram notation for PDAs given in Sipser.

When the PDA reads an a, it pushes A or pops B. When it reads a B, it pushes two B's or pops two A's or one of each. If the stack is empty, there are twice as many A's.

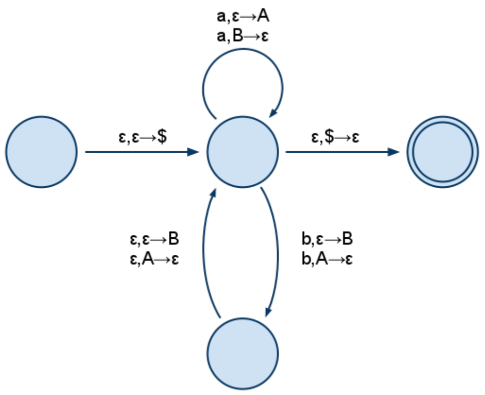

Figure 1: PDA for problem 2.

PROBLEM 3 (8+4 points)

A satisfied boolean expression is a string over the alphabet $\Sigma = \{0, 1, (,), \neg, \wedge, \vee\}$ representing a boolean expression that evaluates to 1, where $\neg$ is the *not* operator, $\wedge$ is the *and* operator, and $\vee$ is the *or* operator (see Sipser p.14). For instance, 1, $(1 \wedge (\neg 0))$, and $(\neg(\neg 1))$ are satisfied boolean expressions.

Meanwhile, $)(1$, $(0\neg)$, $\wedge(0 \vee 1)$, $(0 \wedge 1)$, and $(\neg(1 \vee 0))$ are examples of strings that are *not* satisfied boolean expressions: (The first three examples are strings that aren't valid expressions and simply don't make sense. The last two are well-formed expressions, but they evaluate to 0, rather than to 1.)

(A) Write a context-free grammar that generates the language of satisfied boolean expressions. You may assume that expressions must be completely parenthesized. For instance, your grammar need not generate $\neg(0 \wedge \neg 0 \wedge 1)$ but should generate its equivalent, $(\neg(0 \wedge ((\neg 0) \wedge 1)))$.

(B) Prove that the language of satisfied boolean expressions is not regular.

(A) We construct a grammar such that T generates boolean expressions that evaluate to 1, F generates boolean expressions that evaluate to 0, and A generates any boolean expressions. We let the start symbol be T.

$$T \to 1 \mid (\neg F) \mid (T \vee A) \mid (A \vee T) \mid (T \wedge T)$$

$$F \to 0 \mid (\neg T) \mid (F \wedge A) \mid (A \wedge F) \mid (F \vee F)$$

$$A \to T \mid F$$

(B) The language of satisfied boolean expressions is not regular, because it has to 'count' parenthesis in order to ensure that they match up.
Formally, assume that the language of satisfied boolean expressions was regular. Then, some DFA M recognizes it. We transform M into an NFA N by replacing every transition that does not read a left or a right parenthesis symbol with an ε transition. Then N recognizes the language of properly balanced parenthesis. But the language of properly balanced parenthesis is not regular, so this is impossible. Therefore, the language of satisfied boolean expressions must not be regular.