UVa - cs302: Theory of Computation Spring 2008

Problem Set 3

Corrected version, 16 February 2007. **Due: Tuesday, 19 February (2:02pm)**

This problem set covers material from Classes 1-9 (through February 14) and Sipser's book through the end of Chapter 2, focusing on context-free languages and the two ways we have seen to recognize them: context-free grammars and pushdown automata. For full credit, answers must be concise, clear, and convincing, not just correct. Please **staple** your answer sheets before class.

Honor Policy. For this assignment, we will use the "Tila Tequila" collaboration policy as described on Problem Set 2.

Problem 1: Nondeterminstic Pushdown Automata. Draw a nondeterministic pushdown automaton that recognizes the language $\{w0w^R | w \in \{0,1\}^*\}$. The fewer states you use, the better.

Problem 2: Defining Regular Expressions. Definition 1.52 provides a formal definition of a *regular expression*. Rewrite that definition as a context-free grammar. The set of terminals is implied by the definition: $\Sigma = \{a, \epsilon, \emptyset, \cup, \circ, ^*, (,)\}$ (where a represents any symbol in the alphabet, and ϵ means the epsilon symbol, not the empty string; if you need to represent the empty string in your grammar use λ to avoid confusion with the ϵ symbol that can appear in a regular expression). You may use as many variables are you need to be clear, but should give them sensible names. Your grammar should be able to generate all possible regular expressions, but no strings that are not regular expressions.

Problem 3: Context-Free Grammars. Consider the grammar G below, which describes the same language as Problem 1d from PS1: (S is the start symbol, and 0 and 1 are terminals)

$$
\begin{array}{lcl}
S & \rightarrow & \epsilon \\
S & \rightarrow & S\mathbf{00} \\
S & \rightarrow & \mathbf{11}S \\
S & \rightarrow & \mathbf{0}S\mathbf{1} \\
S & \rightarrow & \mathbf{10}S
\end{array}
$$

a. Show that the string 111000 can be produced by G by showing a derivation that produces it.

b. How many different derivations are there in G to produce 111000? (Support your answer with a clear argument.)

c. What is the fewest number of rules that can be added to G to produce a grammar that describes the language of all even-length strings in $\{0,1\}^*$? (Your answer should include the rules to add.)

Problem 4: Regular Grammars. As discussed in Lecture 8, a *regular grammar* is a replacement grammar in which all rules have the form $A \to aB$ or $A \to a$ where A and B represent any variable and a represents a terminal. Prove that all regular languages can be recognized by a regular grammar.

Problem 5: Pumping Lemma for Context-Free Languages. For each part, either argue that the language is context-free (ideally, by showing how a PDA could recognize it or a CFG could generate it) or use the pumping lemma to show it is not context free.

a. $\{0^i 1^i 0^i\}$

b. $\{1^a + 1^b = 1^c | a \geq 0, b \geq 0, c \geq 0, a + b = c\}$

c. $\{0^i 1^j 2^k | i < j < k\}$ (hint: compare to Example 2.37)

Problem 6: Parsing. Below is a slightly simplified excerpt from the actual Java grammar specification (from *http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html*, Chapter 18). I have changed the syntax to match the context-free grammar notation used in Sipser and the class.

Expression	→	*Expression1 OptAssignmentOperator*
OptAssignmentOperator	→	ϵ | *AssignmentOperator Expression1*
Expression1	→	*Expression2 OptExpression1Rest*
OptExpression1Rest	→	ϵ | *Expression1Rest*
Expression1Rest	→	**?** *Expression* **:** *Expression1*
AssignmentOperator	→	**=**
Expression2	→	*Expression3 OptExpression2Rest*
OptExpression2Rest	→	ϵ | *Expression2Rest*
Expression2Rest	→	*InfixExpressionList*
InfixExpressionList	→	ϵ | *InfixExpression InfixExpressionList*
InfixExpression	→	*InfixOp Expression3*
InfixOp	→	|| | **&&** | **==** | **+**
Expression3	→	*Primary SelectorList*
Primary	→	**(** *Expression* **)** | **Identifier** | **Literal**
SelectorList	→	ϵ | *Selector SelectorList*
Selector	→	**[** *Expression* **]** | **. Identifier**

We use **Identifier** to mean any valid Java identifier (see Section 3.8 of the Java Language Specification for the grammar for Identifiers) and **Literal** to mean any numeric literal. For the examples, assume any single alphabet letter is an **Identifier** and that all variables are declared with type `boolean`, and any number, `true`, and `false` are **Literal**s. (The conditional expression, $Expression_{pred}$ **?** $Expression_{consequent}$ **:** $Expression_{alternate}$, is evaluated by first evaluating $Expression_{pred}$, which must evaluate to a boolean. If it evaluates to true, then the value of the conditional expression is the value obtained by

evaluating $Expression_{consequent}$ (and $Expression_{alternate}$ is not evaluated). If it evaluates to false, then the value of the conditional expression is the value obtained by evaluating $Expression_{alternate}$ (and $Expression_{consequent}$ is not evaluated).)

a. Show a derivation for the expression: `a . f`

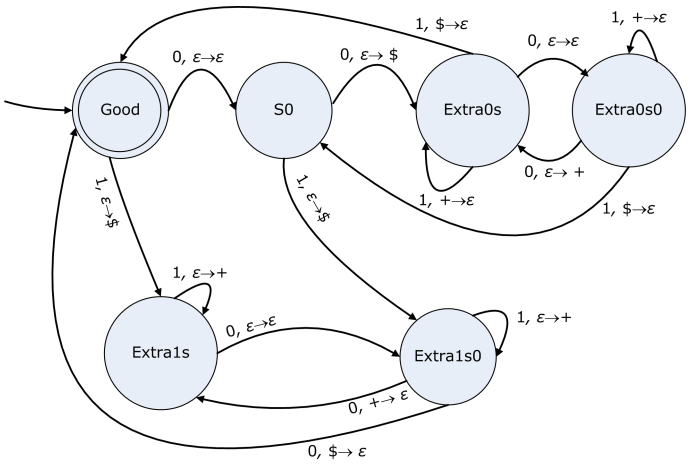
b. Consider the following Java expression:

```
true ? false ? true == true : false : false == false
```

which evaluates to `false`. By adding only parentheses, transform it into a grammatical Java expression that evaluates to `true`.

c. Explain how you would change the grammar rules so all $Expression$s that can be produced by the above grammar are still valid expressions, but the original expression in the previous part evaluates to `true` in the modified grammar. It is acceptable if your answer leads to an ambiguous grammar, as long as one possible parse of the expression in your grammar evaluates to `true`. (It is better, of course, if your grammar is unambiguous and the only possible parse of the expression evaluates to `true`.)

Problem 7: Deterministic Pushdown Automata. Precisely describe the language recognized by the deterministic pushdown automata shown below. (The state names are intended to be somewhat helpful, but not completely revealing. If your answer is correct, you should be able to find a simple way to describe the language.) (Hint: try comparing this machine to the machine from Class 6 from class and Example 3 of the notes.)



Challenge Bonus. (up to 20 points) Draw a deterministic pushdown automaton that recognizes the same language, but uses fewer states. (Make sure that your PDA is still deterministic.) A good answer must include a clear explanation of how your DPDA works.