Programming Assignment 2
Simple Grammar Analyzer
Due: December 5, 2014

CS311 Fall 2014

Overview

In this project you will be writing a program to read in a grammar and a string and determine if the grammar can generate that string. Unlike with project 1 where any DFA was a possible input, you will only need to be able to deal with specific grammars. You may use any programming language you choose.

Projects should be sent to *dleblanc@pdx.edu* with the following subject line:

CS311 Fall 2014 Project2 *your name*

Your e-mail should contain as attachments your completed working code and a 1-2 page, not including sample inputs/outputs, writeup describing your work. The writeup must be submitted as a **PDF**. If your code is comprised of several files please zip the files before sending. As an alternative, you can send a link to a repository containing your code.

1 Encoding a Grammar [25 pts]

For this part of the project you need to find a way to represent a Context-Free Grammar as a file so that your program can read that file in part 2. For this project we will be restricting ourselves to a very small subset of the possible grammars that can exist. There are two main criteria the grammars need to have.

1. All rules in the grammar must have a terminal as the first symbol on the right hand side of the rule.
2. No variable in the grammar can have two rules with the same terminal as the first symbol on the right hand side.

For full credit you must encode the following languages and at least one additional language of your choosing.

$$\{a^n\#b^n \mid n > 0\}$$

$$\{w\#w^R \mid w \in \{0,1\}^*\}$$

$$\{a^i\#b^j\#c^k\# \mid i = j \text{ and } i,j,k > 0\}$$

If you are having trouble creating grammars that obey the described criteria feel free to contact me or talk with your classmates. The encoding must be your own work, but I won't be as strict on the grammars.

2 Reading the Grammar [10 pts]

Once you have created the grammars you need to read them into some data structure of your choice. Read the next section to see how the grammars will be used to parse a string. The process will be similar to the pushdown automatons we discussed in class. You will also need to implement a way for the user to enter a string to be parsed. A simple command-line interface is what I recommend, but you are welcome to make another choice if you have other ideas.

3 Parsing the String [25 pts]

To parse a string you will need to maintain a stack and an input buffer. Begin by pushing the start variable onto the stack. Then repeat the following until you empty the input buffer.

1. Pop an element from the stack.
2. If the popped element is a variable, look at the next character in the input and use that to determine which rule to apply. Then push the right hand side of that rule onto the stack. If no rule matches the next input reject the string.
3. If the popped element is a terminal, make sure it matches the next character in the input and remove both. If it does not match reject the string.
4. If the stack is empty and the input buffer is not, reject the string.
5. If both the stack and the input buffer are empty, accept the string.

4 Testing [20 pts]

Once you have finished the program you will need to perform some thorough tests. You may assume that the grammar files are in whatever format you specify. Be sure to test a wide variety of input strings for each grammar. At minimum you need to test each possible accepting or rejecting path, as well as any corner cases that may exist for you language. A good strategy is to manually generate a small set of test cases for each grammar and then randomly generate some additional string that can be used. You may submit the tests in their own file or at the end of your writeup.

5 Writeup [20 pts]

Your writeup should walk through each of the above steps and explain why you took the approach you did. I would also like answers to the following questions.

1. Why were those particular restrictions placed on the grammars?
2. If those restrictions weren't there, how would your program need to change?
3. What would happen if the grammar were ambigious?
4. What other approaches might be used to tell if a string can be generated by a given grammar?