Grammar Analyzer Writeup - CS311 - Leblanc

Ben Reichert

December 4, 2014

Encoding the Grammars: The languages given were converted into grammars following the two rules given. There's not a whole lot to talk about here, since the grammars were created based on the given rules and that's about it. The final encodings are as follows:

```
1.txt
S->aSb|#

2.txt
S->0S0|1S1|#

3.txt
S->aAb#cC
A->aAb|#
C->cC|#

4.txt
#{w | w is either 0 or an the same number of 1's on either side of a 0}
S->1S1|0
```

Reading the Grammar: The grammars were read in from their respective file and formats, as seen above. The file format resembles normal grammar notation with arrows between the variables and pipes between the rules. The file is read into a list of lists, where every element of the list is a list containing the start variable followed by the rules as indexes of the list. As a design decision, if the user inputs any character as part of the input string that is a variable in the grammar, the program will get confused and infinitely loop due to the indefinite pushing and popping of the rules. There is not a lot of input checking for the program, as I saw it was largely unecessary. If the user gives the application bad input, it will run poorly as you might think. It does not handle every case since it wasn't designed to be bulletproof.

```
File format: 1.txt S->aSb|#
Data structure in python by [['S','aSb','#']]
```

Parsing the String: The users input is taken in on the command line like './grammanalyzer.py filename string'. The string is put into a "buffer" which

is just another string as part of the Grammanalyzer class that it can peek and pop characters from. The Stack is implemented as a string, in the StringStack class. The stack has push, pop, and peek methods available. The push method takes input of any length and reverses it so 'food' would be 'doof' since the stack "top" is the rightmost character of the string. The file IO method constructs the previously described list of lists which the program uses to determine which rules to apply. Each time through a while loop a stack and buffer character are peeked at to see what the next step is. According to the rules in the requirements document, it makes a decision based on if both or either are empty, or if the stackpeek variable is a variable or terminal and deals with it accordingly. There were times where I had to rewrite code since I wasn't properly handling the # symbols, or when I would accidentally pop variables off the stack when I should have been only peeking at them. I resolved all of this over the course of an afternoon and everything seems to be runnign smoothly now. The method that does all of this logic is the sim method of the Grammanalyzer class.

Testing: I wrote a short shell script, test.sh, that tests each grammar with acceptable and rejecting input based on their rulesets. I tried to do my best to cover all cases of possible execution when testing the grammars.

Writeup other questions:

1. Why were those particular restrictions placed on the grammars?

The first restriction is to ensure we always product a string with at least one terminal, and cannot have an empty language. The second is to prevent having to "choose" which path to take, say you had aSb and aSaa, you would have to have more complicated logic to deal with that by looking farther ahead. In this case our application only has to look ahead into the stack and buffer one or two characters, but with multiples of the same terminal we would have to choose, or execute both branches at once, and therefore might have non-deterministic outcomes.

2. If those restrictions werent there, how would your program need to change?

I would have to take into account the empty string, and how to represent that in the application but not in the final output. I would also have to deal with branching if multiple terminals were allowed in the same rule, and possibly have something like a parse tree.

3. What would happen if the grammar were ambigious?

If the grammar was ambigious, there could be multiple right answers depending on how the program executed and what path it took to construct the grammar. To deal with this, the program would have to traverse a parse tree to get all possible right answers based on some input and then check if any one of them were right, which just sounds painful.

4. What other approaches might be used to tell if a string can be generated by a given grammar?

I briefly looked into parse trees before reading the assignment fully, which made me think this was going to be way harder than it turned out to be. I am glad I didn't have to implement a parse tree and the complicated parsing algorithms that go with them. So there is another way to determine if a string

was generate with a given grammar, but not the most efficient or easy way for these languages.

Test output follows from test.sh:

```
Language 1 ACCEPT
Grammar templates/1.txt is the following:
S->aSb|#
String a#b in language

Grammar templates/1.txt is the following:
S->aSb|#
String aa#bb in language

Grammar templates/1.txt is the following:
S->aSb|#
String aaa#bbb in language

Language 1 REJECT
Grammar templates/1.txt is the following:
S->aSb|#
String aaa#bb is NOT in language

Grammar templates/1.txt is the following:
S->aSb|#
String aa# is NOT in language

Grammar templates/1.txt is the following:
S->aSb|#
String a# is NOT in language

Grammar templates/1.txt is the following:
S->aSb|#
String #b is NOT in language

Grammar templates/1.txt is the following:
S->aSb|#
String b# is NOT in language

Grammar templates/1.txt is the following:
S->aSb|#
String aa## is NOT in language

Grammar templates/1.txt is the following:
S->aSb|#
String jaldfkjlajdsfljalsdfasdljfaljdflkja is NOT in language

Language 2 ACCEPT
Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 0#0 in language
```

```
Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 1#1 in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 00#00 in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 11#11 in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String # in language

Language 2 REJECT
Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 11#10 is NOT in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 10#11 is NOT in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 1#0 is NOT in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 0#1 is NOT in language

Grammar templates/2.txt is the following:
S->0S0|1S1|#
String 10101010101010101010101001010101001010100101001001010 is NOT in
    language

Language 3 ACCEPT
Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String a#b#cc# in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
```

```
String aaa#bbb#cc# in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aaa#bbb#c# in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aaaaa#bbbbb#c# in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aaa#bbb#ccc# in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aa#bb#cc# in language

Language 3 REJECT
Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String abc is NOT in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aaa#ccc# is NOT in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String a#bb#cc# is NOT in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aa#b#cc# is NOT in language
```

```
Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String aa#b#c# is NOT in language

Grammar templates/3.txt is the following:
S->aAb#cC
A->aAb|#
C->cC|#
String # is NOT in language

Language 4 ACCEPT
Grammar templates/4.txt is the following:
S->1S1|0
String 101 in language

Grammar templates/4.txt is the following:
S->1S1|0
String 11011 in language

Grammar templates/4.txt is the following:
S->1S1|0
String 0 in language

Grammar templates/4.txt is the following:
S->1S1|0
String 1110111 in language

Language 4 REJECT
Grammar templates/4.txt is the following:
S->1S1|0
String 1 is NOT in language

Grammar templates/4.txt is the following:
S->1S1|0
String 1101 is NOT in language

Grammar templates/4.txt is the following:
S->1S1|0
String 1011 is NOT in language

Grammar templates/4.txt is the following:
S->1S1|0
String 10 is NOT in language

Grammar templates/4.txt is the following:
S->1S1|0
String 01 is NOT in language
```

```
Grammar templates/4.txt is the following:
S->1S1|0
String 2323242424 is NOT in language
```
