Matthew O'Brien
CS311 Project 2
6/5/2015

I actually found this project to be a lot more difficult then the last one. I was really expecting something else from this, and considering how crazy this week has been, I'm actually quite surprised I was able to get this done at all. Hopefully you had fun at the play last night. Now onto more serious topics.

To start off with, for the encoding of the grammar, the first portion you said that all of the grammars must have a terminal as the first position of the right hand side of the rule. I figured the reasoning for that comes from having to look at both the stack and the input buffer. If they match originally, the terminal is popped off the stack, and the next input is considered. This would allow for a much easier look a head method, I think. I didn't really try it the other way around, mostly because the instructions said so. I struggled enough with the encoding of this project, I didn't attempt to deviant from the instructions all too much. The next portion made more sense to me though. Making sure each variable's rules did not have the same terminal. As this would break the look a head method of the entire parser. I actually sort of ran away from the look ahead method, and just sort of took the input off the buffer and popped/added to the stack right away. I feel like I made a lot more work for myself with that method, but I was really struggling with the encoding, and when I finally got it to work for the first language, I didn't really look back. Parsing the string was mostly followed by the instructions laid out in the assignment. I gave the stack a dummy element of sorts, to show the difference between an empty stack, and the correct stack for error checking. I immediately popped off the variable, and looked at the input buffer. This would then push onto the stack, assuming a correct string. This would continue until a terminal is reached, if it was the correct one, both the stack and input buffer would remove that terminal, and again, assuming a correct string. This would continue until the string finished, and the stack only had my dummy element inside it. If any other sort of deviation from a correct string would happen, the stack would reach an empty point, and reject the string or the stack would have more then just the final dummy element. I'm pretty sure I had made a possible mistake with the way I set up possible variables and inputs. In some cases, if the string was incorrect I would toss useless terminals onto the stack, and nothing would happen by the end of the parse. This would leave a nonempty stack, and the string would be rejected. I just feel like this was sort of redundant, and a better method for parsing the string could have been implemented.

Like I stated already, I ran into a lot of troubles with the encoding of this program. I used the framework of my DFA to work my way through this project, and originally that was a terrible idea. I made a lot of mistakes with my rules for variables. After a lot of frustration and a few beers later, I was able to work my way through most of the languages. I found a corner case for the third language, if it ends with two or more #'s, it will accept the string, with or without, c's. This is because of how I handled the input for #. I actually did that grammar last, so I wasn't coding around the idea of a # being the end of the string. I was able to fix the grammar for any number of c's, but any string with the appropriate number of a's and b's with the right #, ending in a ## will be accepted. I figured I would just own up to my mistake rather then let you find it. On the upside, I learned a lot more about python thanks to this project.

The restrictions that were placed on the grammars in order in ensure a working non deterministic look a head PDA. Without that set up this project would have been much more difficult then it already was for me. The program would need to be able to branch and build upon any combination of strings, having multiple stacks, and if any of them were accepted, the string would be accepted. And I have honestly, no idea how I could go about that. It sounds terrifying. The grammars themselves were simple number restrictions. I thought about trying a palindrome for my forth grammar,

and I very quickly changed my mind. Using a simple set of numbers for it was a lot easier to restrict and code around, then some specific pattern. I also thought of the #'s as a epsilon. Since I had no idea how to encode a rule to go to nothing, and to still change the stack, I treated the # as if it was epsilon, and would move to the next input along the buffer. I'm unsure if that is the correct way, but it worked for me in every case but the one corner case of grammar three. Another approaches might be to actually use the old DFA program, and modify it have another set of transitions to push and pop on the stack. Then you wouldn't be pushing the rules themselves onto the stack, but anything and you would just move from state to state. If the stack was empty and an ending state was the final state, accept.

For this project, I included four different JSON files for the different grammars. I noted the portion of the code for where to change the file name to try the other grammars. For the first grammar,

Input	Output
aa#bb	Acceptable String!
a#bb	Bad String!!
aaa#bbb	Acceptable String!
aa#b	Bad String!!
a#	Bad String!!
#b	Bad String!!
#	Bad String!!

The second grammar,

Input	Output
1010#0101	Acceptable String!
111#111	Acceptable String!
000#000	Bad String!!
1#0	Bad String!!
0#1	Bad String!!
#	Bad String!!
11#00	Bad String!!

The third grammar,

Input	Output
aa#bb#cc#	Acceptable String!
aa#bb#c#	Acceptable String!
a#bb#c#	Bad String!!
aa#b#ccc#	Bad String!!
#	Bad String!!
aaa#bbb##	Bad String!!(This is the bad one)
aa#bb#	Bad String!!

The final grammar,

Input	Output
aa#b	Acceptable String!
aaaa#bb	Acceptable String!
#b	Bad String!!
a#	Bad String!!
aaa#b	Bad String!!
a#bb	Bad String!!(This is the bad one)
aaaa##bb	Bad String!!

In conclusion, my project works in almost all of the possible cases that I tested. Excluding that one corner case that I found, that I just can't seem to rid myself of, the other possible strings tested came out as expected. As to not go over two pages. I will end my explanation here. Hopefully it is enough.