Lab 4: Grammars and Pushdown Automata

CSCI 340: Computational Models

100 points

Submission will be through `autolab.millersville.edu`

Remember to use JFLAP (version 7.1) to complete these problems.

Steps to Follow

- Go to AutoLab and download the handout for Lab 4. This will give you a `.zip` file with six `.jff` files in it. The structure of the `.zip` file should be:

```
$ zip -sf handout.zip
Archive contains:
  handout/
  handout/1a.jflap.jff
  handout/1b.jflap.jff
  handout/2a.jflap.jff
  handout/2b.jflap.jff
  handout/3a.jflap.jff
  handout/3b.jflap.jff
```

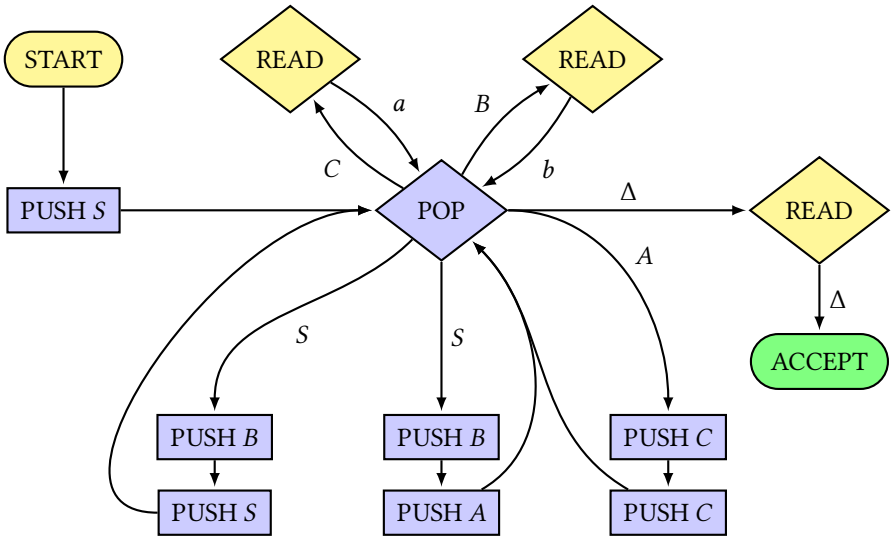
- **Do NOT change the file names.** Edit the files in JFLAP to create the appropriate automata. Do NOT create new files and overwrite.
- When you are ready to submit one or more automata for testing, create a `.zip` file of the directory (**it must still be called handout**). This should ***exactly match*** the structure of the handout `.zip` file.
- Submit the created `.zip` file to AutoLab.

JFLAP Representation of PDAs

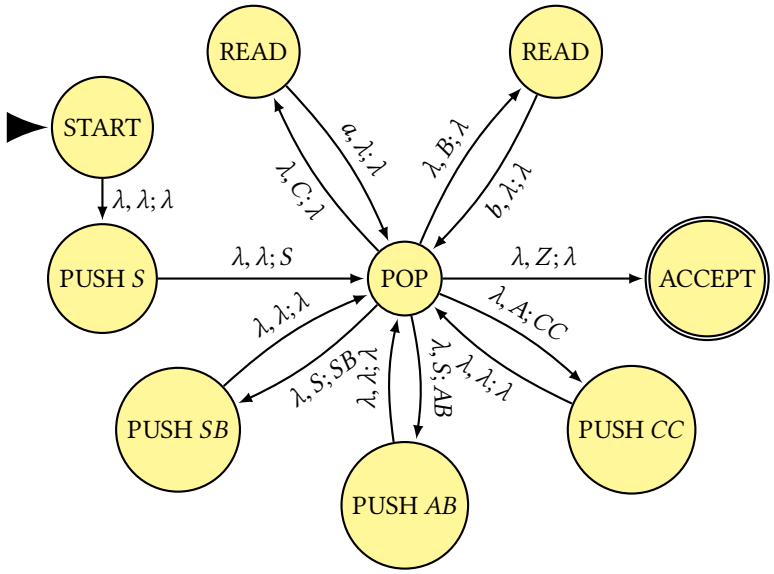
Our representation of PDAs in lecture are different from what JFLAP uses. Consider the following grammar:

$S \to SB$
$S \to AB$
$A \to CC$
$B \to b$
$C \to a$

The PDA for the language may look like:

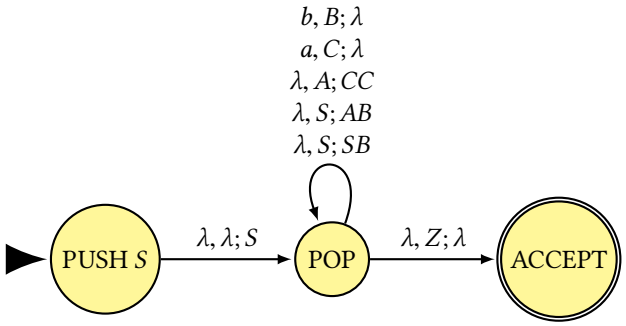


In JFLAP it may be represented as:



Note that on each transition, the first value represents what is **read**, the second value represents what is **popped** and the third value represents what is **pushed** onto the stack (be careful here, this might be in a different order than you'd expect). **Important: JFLAP begins by inserting a Z onto the stack. When we pop a Z, we know that stack is empty.**

In this PDA, we only take one action on each transition. If we combine some of the actions on the transitions that form loops between two states, we can create a PDA with fewer states, like this:



For this assignment, you can do whichever makes more sense to you.

Problems

1. [30pts] Let $\Sigma = \{a \;\; b \;\; c\}$ and L be the language of all words in which all a's come before the b's and there are the same number of a's and b's. There are arbitrarily many c's that can exist anywhere in the string. Some words in L are *abc*, *caabcb*, and *ccacaabccccbccbc*

   (a) Construct a PDA that accepts L

   (b) Construct a CFG that generates L

2. [30pts] Let L be the language of all words that have the same number of a's and b's and that – as we read them from left to right, never have more b's than a's. For example *abaaabbabb* is good but *abaabbbba* is no good because at a certain point, we have four b's and only three a's.

   (a) Construct a PDA that accepts L

   (b) Construct a CFG that generates L

3. [40pts] Let us consider the set of all regular expressions to be a language over the alphabet $\Sigma = \{ \;\; a \;\; b \;\; (\;\;) \;\; + \;\; * \;\; \hat{} \;\; \}$. Note that you should use the carat symbol ($\hat{}$) to represent λ for simplicity's sake. Let us call this language REGEX.

   (a) Construct a PDA that accepts *REGEX*

   (b) Construct a CFG that generates *REGEX*