CS 341: Foundations of Computer Science II
Prof. Marvin Nakayama

Project 2

(Updated on 10/26/2022 to add Step 1 to the Program Specifications on page 3)

Section 005: Due 11/3/2022, 2:30pm NJ local time

Section 007: Due 10/31/2022, 11:30am NJ local time

Be sure to read this entire document before starting the assignment.

Academic Integrity

Any student caught cheating on any assignment will be reported to the dean of students. Cheating includes, but is not limited to, getting solutions (including code) from or giving solutions to someone else. You may discuss the assignment with others, but ultimately you must do and turn in your own work.

1 Overview

We define the language A to be a particular set of strings (defined below) that represent valid arithmetic expressions operating on floating-point numbers, with the entire expression contained between specified delimiters `%`. For this assignment you are to draw a PDA that recognizes this language and write a program that implements your PDA.

2 The Language A

To precisely define the language A, we first define the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S, T, F, X, D\}$ is the set of variables; the alphabet is

$$\Sigma = \{ \texttt{.}, \texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{(}, \texttt{)}, \texttt{\%} \}, \tag{1}$$

which includes a dot for float-point numbers; the starting variable is S; and the rules R are

$$\begin{array}{rcl} S & \rightarrow & \texttt{\%}T\texttt{\%} \\ T & \rightarrow & T\texttt{+}T \mid T\texttt{-}T \mid T\texttt{*}T \mid T\texttt{/}T \mid \texttt{(}T\texttt{)} \mid F \\ F & \rightarrow & X\texttt{.}X \mid X\texttt{.} \mid \texttt{.}X \\ X & \rightarrow & DX \mid D \\ D & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \cdots \mid \texttt{9} \end{array}$$

Then we define the language $A = L(G)$, which contains strings that begin with `%` and end with `%`, and in between is an arithmetic expression over floating-point numbers. For example, the string "`%(15.-(6.312*.7))%`" belongs to A, which we can show by using the derivation

$$\begin{aligned}
S &\Rightarrow \%T\% \Rightarrow \%(T)\% \Rightarrow \%(T\texttt{-}T)\% \Rightarrow \%(T\texttt{-}(T))\% \Rightarrow \%(T\texttt{-}(T*T))\% \\
&\Rightarrow \%(F\texttt{-}(T*T))\% \Rightarrow \%(X\texttt{.-}(T*T))\% \Rightarrow \%(DX\texttt{.-}(T*T))\% \\
&\Rightarrow \%(DD\texttt{.-}(T*T))\% \Rightarrow \%(\texttt{1}D\texttt{.-}(T*T))\% \Rightarrow \%(\texttt{15.-}(T*T))\% \\
&\Rightarrow \%(\texttt{15.-}(F*T))\% \Rightarrow \%(\texttt{15.-}(X\texttt{.}X*T))\% \Rightarrow \%(\texttt{15.-}(D\texttt{.}X*T))\% \\
&\Rightarrow \%(\texttt{15.-(6.}X*T))\% \Rightarrow \%(\texttt{15.-(6.}DX*T))\% \Rightarrow \%(\texttt{15.-(6.}DDX*T))\% \\
&\Rightarrow \%(\texttt{15.-(6.}DDD*T))\% \Rightarrow \%(\texttt{15.-(6.3}DD*T))\% \Rightarrow \%(\texttt{15.-(6.31}D*T))\% \\
&\Rightarrow \%(\texttt{15.-(6.312}*T))\% \Rightarrow \%(\texttt{15.-(6.312*.}X))\% \Rightarrow \%(\texttt{15.-(6.312*.}D))\% \\
&\Rightarrow \%(\texttt{15.-(6.312*.7)})\%
\end{aligned}$$

The above derivation is given so that you can get a better understanding of the language A, but the derivation cannot be directly used to design a PDA for A. The grammar G does not include the rule $T \rightarrow \texttt{-}T$ nor the rule $T \rightarrow \varepsilon$, so the strings "`%-45.0%`" and "`%.8+-9.%`", do not belong to A. Also, note that operands must be floating-point numbers, so integers (numbers without a dot) are not allowed; e.g., "`%45+3%`" does not belong to A. The grammar G is ambiguous; e.g., the string `%1.+2.*3.%` $\in A$ has two different parse trees.

3 PDA for A

First you are to construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q_1, F)$ that recognizes A, where Σ is defined in equation (1). The PDA M must satisfy the following conditions:

- The PDA must be defined with the alphabet Σ defined in equation (1). In other words the PDA must be able to handle any string of symbols from Σ. The PDA can handle certain strings not in A by crashing, i.e., the drawing does not have an edge leaving a state corresponding to particular symbols read and popped.
- The PDA must have exactly one accept state.
- The states in the PDA must be labeled $q_1, q_2, q_3, \ldots, q_n$, where q_1 is the start state and n is the number of states in the PDA. (It is also acceptable for the states to be labeled $q_0, q_1, \ldots, q_{n-1}$, with q_0 the start state.)
- Each edge in the PDA must correspond to reading a symbol from Σ; i.e., no edge can correspond to reading ε. There is no restriction on pushing or popping ε on transitions.
- Leaving the start state is exactly one edge with label "$\%, \varepsilon \rightarrow \%$". Thus, the first thing the PDA does is it reads a `%`, pops nothing, and pushes a `%` on the stack.

- Any edge going into the accept state has label "$\%, \% \to \varepsilon$".
- There cannot be two edges corresponding to reading the same symbol $a \in \Sigma$ and popping the same symbol $b \in \Gamma$ leaving a single state.

You will not be able to use the algorithm in Lemma 2.21 to convert the CFG G into a PDA for A since the resulting PDA will not satisfy the last four properties above. However, those properties ensure that the PDA M is essentially deterministic, so once you figure out M, it will be easy to implement as a program. (Implementing a nondeterministic machine is more difficult since the program would need to check every branch in the tree of computation.)

The drawing of your PDA must include all edges that are ever used in accepting a string in A. But to simplify your drawing, those edges that will always lead to a string being rejected should be omitted. Specifically, when processing a string on your PDA, it might become clear at some point that the string will be rejected before reaching the end of the input. For example, if the input string is "`%34.5+*6.29%`", then it is clear on reading the `*` that the string will not be accepted. Moreover, if an input string ever contains the substring `+*`, then the input string will be rejected. Thus, your drawing should omit the transition corresponding to reading the operator `*` in this case, so the PDA drawing will crash at this point.

In this project, the machine you design needs to only *recognize* the language A, not to *evaluate* the arithmetic expression. Because of this, your PDA does not need to distinguish between different Arabic numerals; e.g., there should be no difference in reading the symbol `3` or the symbol `6` in the input. Thus, you can define notation such as $\Sigma_D = \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}\}$ to denote the set of digits (Arabic numerals), and you can use this notation in labeling certain transitions. Also, all operators (`+`, `-`, `*`, `/`) can be handled similarly.

4 Program Specifications

You must write your program in C, C++, Java, or Python. All input/output must be through standard input/output, and your program is to work as follows:

1. Your program first prints:

   ```
   Project 2 for CS 341
   Section number:  the section number you are enrolled in
   Semester:  Fall 2022
   Written by:  your first and last name, your NJIT UCID
   Instructor:  Marvin Nakayama, marvin@njit.edu
   ```

2. Your program asks the user if s/he wants to enter a string. The user then enters "`y`" for "yes", or "`n`" for "no".

- If the user enters "`n`", then the program terminates.
- If the user enters "`y`", then the user is prompted to enter a string over Σ.

3. If the user chooses to input a string, your program then reads in the string. You may assume that the user will only enter a string over Σ. After reading in the string, your program prints it. Then your program processes the string on your PDA in the following manner.
   - Your program must begin in the start state of the PDA and print out the name of that state (q_1 or q_0).
   - After each transition of the PDA, your program must print out the symbol that was read, the symbol that was popped from the stack, the symbol that was pushed onto the stack, and the name of the current state of the PDA after completing the transition.
   - If the PDA crashes before reaching the end of the input string, your program should output this fact.
4. After completing the processing of the string on the PDA (possibly by crashing), your program must indicate if the string is accepted or rejected based on how the string was processed on the PDA. Your program then must return to step 2.

5 Test Cases

Test your program on each of the following input strings:

1. `%381.5886%`
2. `%.8+7.-8.00/90.765+%`
3. `%7956.+.492*341.2/060.10/52581.263-.9+.53/7.%`
4. `%382.89*34%`
5. `%4.239-.*7.29%`
6. `%6.88.6+32.208%`
7. `%(1.2+(3.4-.9)/39).3%`
8. `%(.3)64%`
9. `%((824.23+(9.22-00.0)*21.2))+((.2/7.))%`
10. `%(())%`
11. `%((14.252+(692.211*(.39+492.1))/49.235)%`
12. `%+6.5%`

13. `%26.0*(.87/((4.+.2)/(23.531)-2.9)+6.)/(((823.*.333-57.*8.0)/.33+.0))%`

14. `%.0*(32.922+.7-*9.))%`

15. `%(4.+(.8-9.))/2.)*3.4+(5.21/34.2%`

You must create an output file containing your program's output from each test case in the order above.

6 Deliverables

You must submit all of the following **through Canvas** by the due date/time given on the first page:

1. A Microsoft Word document stating, "I certify that I have not violated the University Policy on Academic Integrity", followed by your first and last name, NJIT student ID, and UCID. If you do not include this pledge, then you will receive a 0 on the assignment. Anyone caught violating the University Policy on Academic Integrity will be reported to the dean of students.

2. A drawing of the PDA for A that your program implements. This format of the file must be either Microsoft Word, pdf, or jpeg (e.g., a photo from your phone's camera, but make sure it's not blurry). The file must be smaller than 5MB in size.

3. A **single file** of your source code, of type `.c`, `.cpp`, `.java`, or `.py`. Only submit the source code; do not include any class files. You must name your file

   `p2_22f_ucid.ext`

   where `ucid` is replaced by your NJIT UCID (which is typically your initials followed by some digits) and `.ext` is the appropriate file extension for the programming language you used, e.g., `.java`. The first few lines of your source code must be comments that include your full name and UCID.

4. A **single file** containing the output from running your program on all of the test cases, in the order given in Section 5 of this document. The output file must be either .txt or in Microsoft Word.

The files **must not be compressed**. You will not receive any credit if you do not complete all of the above. **Late projects will be penalized as follows:**

Lateness (Hours)	Penalty
$0.0 <$ Lateness ≤ 24	10
$24 <$ Lateness ≤ 48	30
$48 <$ Lateness ≤ 72	60
$72 <$ Lateness	100

where "Hours" includes any partial hours, e.g., 0.0000001 hours late incurs a 10-point lateness penalty. A project is considered to be late if all of the above are not completed by the due date/time, and the lateness penalty will continue to accumulate until all of the above have been completed. Any submissions completed more than 72 hours after the due date/time will receive no credit.

7 Grading Criteria

The criteria for grading are as follows:

- the correctness of the drawing of your PDA for A (30 points; you will lose points if your PDA has significantly more states than necessary)
- your program works according to the specifications given in Section 4, matches your PDA for A, and follows the directions in Section 6 (15 points)
- your program is properly documented with comments (10 points)
- your output is correct for the test cases (45 points).

Your grade will mainly be determined by examining the source code, the drawing of the PDA, and the output that you turn in; the source code will only be tested if there are questions about your code.

To receive any credit for this assignment, you must turn in a drawing of the PDA your program implements and a *minimally working* program. For a program to be minimally working, it must

- compile without syntax errors;
- properly process all strings in A_0, where A_0 is the set of strings in A that do not have parentheses; and
- implement the drawing of your PDA for A.

If you do not hand in a minimally working program, then you will receive a 0 for the assignment *and* your grade in the course will be lowered by one step, e.g., from B to C+, or from C to D.

8 Hints

You should design your PDA incrementally. First consider a subset A_0' of A_0, where A_0' has arithmetic expressions with no parentheses (as in A_0) but also operands are only integers (i.e., not floating-point numbers). (Actually, A_0' and A_0 are regular languages, so you could start by designing a DFA for A_0', which can be converted into a PDA by popping and pushing ε on each transition.) Once you figure out a PDA for A_0', then modify it to have operands that are floating-point numbers rather than integers (it may help to look at HW 3, problem 6) so that your PDA recognizes A_0. Once you have figured out a PDA for A_0, then enhance your PDA to also accept the rest of the strings in strings in A.

To develop a PDA for A_0', start by partitioning Σ into subsets, where each subset contains one type of symbol. For example, one subset contains the symbols that are operators (`+`, `-`, `*`, `/`). Then draw a state for each subset, and also add a start state and an accepting state. If the PDA is currently in a state corresponding to one subset, then the last symbol read was a symbol from that subset. Then figure out what are the valid choices for the next type of symbol, and draw a transition from the current state to each of those states.