CS 342 Fall 2020 (Secs 003, 005)
Instructor: Ravi Varadarajan
Deterministic PDA implementation

Project Goal

The goal of this project is to implement a deterministic PDA to check if a string conforms to a grammar that generates statement blocks consisting of zero or more assignment statements followed by an arithmetic expression. The left side of an assignment statement is a variable name and the right side is an arithmetic expression. If it is valid, the PDA outputs the derivation starting from the start symbol else produce error if the string is not valid.
You can choose to partner with one person to work on this project.

Grammar

It is a CFG grammar G = (V, Σ, P, S), where V = {S, B, T, T1, T2, F1, F2, X, C, N, I},

$\sum =$ { a, b, c,…..z, A,B,C,……Z,0,1,2,…..9, +, -, *, /, (,), =, ;, \$} which includes assignment operator = and statement separator symbol ;.

The starting variable is S; and the rule set P is given as follows:

$$
\begin{aligned}
&\boldsymbol{S} \rightarrow \$\boldsymbol{B}\$ \\
&\boldsymbol{B} \rightarrow \boldsymbol{T} \mid \boldsymbol{X} = \boldsymbol{T}\,;\boldsymbol{B} \\
&\boldsymbol{T} \rightarrow \boldsymbol{T} + \boldsymbol{T1} \mid \boldsymbol{T1} \\
&\boldsymbol{T1} \rightarrow \boldsymbol{T1} - \boldsymbol{T2} \mid \boldsymbol{T2} \\
&\boldsymbol{T2} \rightarrow \boldsymbol{T2} * \boldsymbol{F1} \mid \boldsymbol{F1} \\
&\boldsymbol{F1} \rightarrow \boldsymbol{F1} \,/\, \boldsymbol{F2} \mid \boldsymbol{F2} \\
&\boldsymbol{F2} \rightarrow \boldsymbol{I} \mid \boldsymbol{X} \mid (\boldsymbol{T}) \\
&\boldsymbol{I} \rightarrow \boldsymbol{N}\,\boldsymbol{I} \mid \boldsymbol{N} \\
&\boldsymbol{N} \rightarrow \boldsymbol{0} \mid \boldsymbol{1} \mid \boldsymbol{2} \mid \boldsymbol{3} \mid \boldsymbol{4} \mid \boldsymbol{5} \mid \boldsymbol{6} \mid \boldsymbol{7} \mid \boldsymbol{8} \mid \boldsymbol{9} \\
&\boldsymbol{X} \rightarrow \boldsymbol{CX} \mid \boldsymbol{C} \\
&\boldsymbol{C} \rightarrow \boldsymbol{a} \mid \boldsymbol{b} \mid \boldsymbol{c} \mid \ldots.\boldsymbol{z} \mid \boldsymbol{A} \mid \boldsymbol{B} \mid \boldsymbol{C} \mid \ldots\ldots \mid \boldsymbol{Z}
\end{aligned}
$$

Note that the variable symbols B,S,T,I,N,X and C are different from the terminal uppercase characters.

Leftmost derivation of the string “\$**a = 2; b = (a+5)*(a/2); fg = a-b; fg\$**” is given follows:

$S \to \$B\$ \to \$X = T; B\$ \to \$C = T; B\$ \to \$a = T; B\$ \to \$a = T1; B\$$

$\to \$a = T2; B\$ \to \$a = F1; B\$ \to \$a = F2; B\$ \to \$a = I; B\$$

$\to \$a = N; B\$ \to \$a = 2; B\$ \to \$a = 2; X = T; B\$ \to \$a = 2; C = T; B\$$

$\to \$a = 2; b = T; B\$ \to \$a = 2; b = T1; B\$ \to \$a = 2; b = T2 * F1 ;$

$\to \$a = 2; b = F1 * F1; B\$ \to \$a = 2; b = F2 * F1; B\$$

$\to \$a = 2; b = (T) * F1; B\$ \to \$a = 2; b = (T + T1) * F1; B\$$

$\to \$a = 2; b = (T1 + T1) * F1; B\$ \to \$a = 2; b = (T2 + F1) * F1; B\$$

$\ldots\ldots\to \$a = 2; b = (a + F1) * F1; B\$ \ldots\to \$a = 2; b = (a + 5) * F1; B\$$

$\to \$a = 2; b = (a + 5) * F2; B\$ \to \$a = 2; b = (a + 5) * (T); B\$$

$\ldots\to \$a = 2; b = (a + 5) * (a/2); B\$ \ldots\to \$a = 2; b = (a + 5) * (a/2); X = T;$

$B\$ \ldots\ldots\to \$a = 2; b = (a + 5) * (a/2); fg = a\text{-}b; B\$$

$\ldots\to \$a = 2; b = (a + 5) * (a/2); fg = a\text{-}b; T\$$

$\ldots\ldots\to \$a = 2; b = (a + 5) * (a/2); fg = a\text{-}b; fg\$\ldots$

Rightmost derivation of the string “\$**a = 2; b = (a+5)*(a/2); fg = a-b; fg\$**” is given follows:

$S \to \$B\$ \to \$X = T; B\$ \to \$X = T; X = T; B\$ \to \$X = T; X = T; X = T; B\,\$$

$\to \$X = T; X = T; X = T; T1\$ \to \$X = T; X = T; X = T; T1\$ \ldots$

$\to \$X = T; X = T; X = T; fg\$ \to \$X = T; X = T; X = T1; fg\$$

$\to \$X = T; X = T; X = T1 - T2; fg\$ \ldots\to \$X = T; X = T; X = T1 - b; fg\$ .. \to$

$\$X = T; X = T; X = a - b; fg\$ \to \$X = T; X = T; CX = a - b; fg\$$

$\to \$X = T; X = T; CX = a - b; fg\$ \to \$X = T; X = T; CC = a - b; fg\$$

$\to \$X = T; X = T; Cg = a - b; fg\$ \to \$X = T; X = T; fg = a - b; fg\$ \ldots$

$\to \$X = T; b = (a + 5) * (a/2); fg = a - b; fg\$$

....→ “\$a = 2; b = (a+5)*(a/2); fg = a-b; fg

If you look at the reverse of the rightmost derivation, then we use reduction from left to right. For example after seeing “\$a”, we first reduce “a” by C (i.e. pop “a” and push “C” to stack) which is again reduced to X to get \$X. Then when we see “=”, we push it to stack. After that when we see “2”, we will reduce it by I which is again reduced to X and then to T so we get \$X=T based on what we have seen so far in the string. We keep adding input to \$X=T until we reduce “**b = (a+5)*(a/2)**” by X=T so that we have \$X=T; X=T reducing the string **\$a = 2; b = (a+5)*(a/2).** Continuing this way we will reduce the string “\$**a = 2; b = (a+5)*(a/2); fg = a-b; fg**” by \$X=T; X=T; X=T;T. Reducing this we get \$X=T; X=T; X=T;B which will be reduced to \$X=T; X=T;B and so on until we get \$B. When we see the final \$, we will reduce B by S. Parsers work this way by using shift-reduce actions; **shift** action involves pushing symbol from input string to stack and **reduce** action involves replacing the string from the top of the stack that matches the right-hand side of a production rule by the non-terminal symbol on the left side of that rule. For example when the stack has "X = T; B" (first symbol is at top of the stack), reduce action can replace it by B using the production rule $\boldsymbol{B \rightarrow X = T ; B}$

Deterministic PDA

Note that since the grammar is unambiguous you can define a deterministic PDA to recognize the language. There will be unique left-most and right-most derivations as mentioned before. Your first task is to design a deterministic PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ that recognizes strings that can be generated by the above grammar.

The DPDA M must satisfy the following conditions:

(a) 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.

(b) The PDA must have exactly one accept state.

(c) The states in the PDA must be labeled $q_o, q_2, q_3, \ldots, q_{n-1}$, where q_0 is the start state and n is the number of states in the PDA.

(d) There is no restriction on pushing or popping ε on transitions but (g) gives restrictions on such operations.

(e) The first thing the PDA does is it reads a \$ and pushes a \$ on the stack.

(f)Any edge going into the accept state has label "\$, \$ → ε".

(g) For every state q and an input symbol "a" **exactly one** of the following action pairs can occur on a transition, apart from possible change of state :

(i) read "a", pop "x" from stack

(ii) read "a", do nothing ($\in pop$) to stack

(iii) do not read any input ($\in read$), pop "x' from stack

(iv) do not read any input or pop from stack (change state and/or push some symbol to stack)

(h) For convenience you can read, pop or push more than one symbol onto stack on a transition (otherwise you may have to introduce a lot of intermediate state to achieve that). But make sure no conflicts arise due to violation of the condition mentioned in (g).

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 "$ab+*bc$", 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 * in this case, so the PDA drawing will crash at this point.

Project deliverables

There are 2 parts to the project. You need a zip archive file that consists of your submissions for the 2 parts of the project.

(a) In the first part, you need to show the transition diagram for your PDA. Scan it to create a pdf document called "**PDADesign.pdf**". This should be included at the top level directory of the archive file.

(b) For the second part, you need to implement the PDA in Java or Python that can process an input string using its transition function.
(i) You need to have the .java files in **src** directory of the archive file and all the necessary class files in a Jar file located in **bin** directory of the zip archive file.
(ii) You also need to have a windows command file (called **PDATester.cmd**) which when run should call the main function to test your PDA. This file should be included at the top level directory of the archive file.

For testing your PDA, write a **main()** function in your java file that repeatedly prompts the user for an input string and processes it using your PDA. If the debug option "-debug" is turned on as a JVM option, you should output each step of the PDA transition step showing the PDA configurations that leads to an accepting or rejecting state. It should print a message "String is rejected" or print the list of steps in the derivation (leftmost or rightmost) of the string in CFG beginning from start symbol as given before.

Academic Integrity

Any student caught cheating on any assignment will receive an automatic fail grade. Cheating includes, but is not limited to, getting solutions (including code) from or giving solutions to someone else or plagiarizing it from internet or other sources. You may discuss the assignment with others, but ultimately you must do and turn in your own work.