German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail

CSEN1002 Compilers Lab, Spring Term 2020
Task 7: $LL(1)$ **Parsing**

Due: 17.04.2020 by 23:59

1 Objective

For this task you will implement an $LL(1)$ parser using pushdown automata (PDA) and predictive parsing tables. Given an input context-free grammar $G = (V, \Sigma, R, S)$, you need to (i) construct the predictive parsing table for G, (ii) construct the PDA equivalent to G, and (iii) implement an $LL(1)$ parser for G which makes use of the table and the PDA to direct its decisions. Given an input string w, the parser should signal an error if $w \notin L(G)$ and produce a derivation of w from S if $w \in L(G)$.

2 Requirements

- *Only Java may be used for this task.*
- We make the following assumptions about input CFGs for simplicity.
  a) The set V of variables consists of upper-case English symbols.
  b) The start variable is the symbol S.
  c) The set Σ of terminals consists of lower-case English symbols other than "`e`".
  d) The letter "`e`" represents ε.
- You should implement three methods: `CFG`, `Table`, and `Parse`. What follows is a description of these methods.
- `CFG` takes an input string encoding a CFG and returns a CFG object. A string encoding a CFG is a semi-colon-separated sequence of items. Each item represents a largest set of rules with the same left-hand side and is a comma-separated sequence of strings. The first string of each item is a member of V, representing the common left-hand side. The first string of the first item is S.
- For example, consider the CFG $(\{S, T\}, \{\texttt{a}, \texttt{c}, \texttt{i}\}, R, S)$, where R is given by the following productions.

$$\begin{array}{lcl} S & \longrightarrow & \texttt{i}\ S\ T \mid \varepsilon \\ T & \longrightarrow & \texttt{c}\ S \mid \texttt{a} \end{array}$$

  This CFG will have the following string encoding.

$$\texttt{S, iST, e; T, cS, a}$$

- `Table` constructs an object representing the predictive parsing table for a CFG, and returns a string encoding of the parsing table. To construct the table, you may use your implementation of *First* and *Follow* from Task 6. You may also use a third-party implementation of *First* and *Follow* provided that you properly cite your source.

- A string encoding of a parsing table is a semi-colon-separated sequence of items. Each item represents a table entry; empty entries are not included. An entry is represented by a coma-separated sequence of items. The first item is a variable and the second is a terminal or $. These two items respectively identify a row and a column of the table. The third and next items represent the rules included in the table entry; each rule is represented by its right-hand side which is a string of variables and terminals. For convenience, entries of the same row should appear consecutively. For example, for the above CFG, the string representation of the corresponding predictive parsing table may be as follows.

  `S,a,e;S,c,e;S,i,iST;S,$,e;T,a,a;T,c,cS`

- `Parse` takes a string w as input and returns a string encoding a left-most derivation of w in G; in case $w \notin L(G)$, this derivation ends with "`ERROR`." The method should construct a PDA equivalent to G and use the PDA together with the table (constructed by `Table`) to reach its decision. Note that we will be testing `Parse` using only $LL(1)$ grammars. Hence, you do not need to include a search algorithm in your implementation; w either has no derivation in G or has exactly one.

- A string encoding a derivation is a comma-separated sequence of items. Each item is a sentential form representing a step in the derivation. The first item is `S`. If $w \in L(G)$ the last item is w; otherwise, it is `ERROR`. For example, given the above CFG, on input string `iiac`, `Parse` should print the following string.

  `S,iST,iiSTT,iiTT,iiaT,iiacS,iiac`

  On the other hand, on input string `iia`, `Parse` should print the following.

  `S,iST,iiSTT,iiTT,iiaT,ERROR`

3 Evaluation

- Your implementation will be tested using two grammars and five input strings for each.
- You get one point for each correct output; hence, a maximum of ten points.

4 Submission Instructions

- The deadline is April 17, 2020 by 23:59. Late submissions will receive zero credit; thus, make sure that you do not submit at the last moment to avoid delays due to any unpleasant technical surprises.
- Only Java may be used for this task.

- Use the default Java package.
- Submit *one* Java source file; use the associated template (`template.java`).
- The file name should have the format *TutorialNo_ID_ Name* (e.g., `T07_37_0000_John_Smith`). Also include the file name, in this format, as a comment in your source file. Submissions which are not in the indicated format will not be considered.
- You will receive an e-mail containing the submission link.