Annie Lane and Michaela Kerem

COMP2022: Formal Languages and Logic

Semester 1, 2014

Assignment 2: Report

**Part I:**

The grammar G is not LL(1) because of left factoring and left recursion.

Left factoring occurs in:

L → I **;** | I

It is resolved by adding another variable M and modifying the rule for L:

L → I M

M → **;** L | ε

Left factoring also occurs in:

E2 → T Op2 E2 | T

It is resolved by adding variable K and modifying the rule for E2:

E2 → T K

K → Op2 E2 | ε

Left recursion occurs in:

E → E Op1 E2 | E2

It is resolved by adding variable R and modifying the rule for E:  
 E → E2 R

R → Op1 E2 R | ε

The new grammar G’ is:  
P →L

L →I M

M → **;** L | 

I →A | C | W

A →**id :=** E

C →**if** E **then** L O **endif**

O →**else** L | 

W →**while** E **do** L **end**

E → E2 R

R →Op1 E2 R | 

E2 →T K

K →Op2 E2 |  

T →**c** | **id**

Op1  →**<** |**=** | **!=**

Op2  →**+** | **-**

**Part II:**

Build the FIRST and FOLLOW sets for the grammar G’.

To build FIRST set for a variable, the left hand side of the production rules are considered. The FIRST set of each terminal is itself. If the FIRST set of a variable contains ε, then in order to build the parse table, the FOLLOW set must also be calculated for that variable. This entails examining the occurrences of the variable on the right hand side of the production rules and determining which terminals can occur after the variable. Often, in order to build FOLLOW sets for a variable, other FOLLOW sets must be built.

The grammar is LL(1) because in the parse table (see Part III) no entry has multiple rules.

Below are the FIRST and FOLLOW sets for all of the variables in the grammar.

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| P | { id, if, while } | { $, endif, end } |
| L | { id, if, while } | { $, endif, end } |
| M | { ε, ; } | { $, endif, end } |
| I | { id, if, while } | { ;, endif } |
| A | { id } | { ;, endif } |
| C | { if } | { ;, endif } |
| O | { ε, else } | { $, endif, end } |
| W | { while } | { ;, endif } |
| E | { c, id } | { ;, endif, do, then } |
| R | { ε, <, =, != } | { ;, endif, do, then } |
| E2 | { c, id } | { <, =, !=, ;, endif, do, then } |
| K | { ε, +, - } | { <, =, !=, ;, endif, do, then } |
| T | { c, id } | { +, -, <, =, !=, ;, endif, do } |
| Op1 | { <, =, != } | { c, id } |
| Op2 | { +, - } | { c, id } |

**Part III:**The parse table is built by using the FIRST and FOLLOW sets of the grammar. The table contains all of the variables in the left most columns and all of the terminals plus ‘$’ in the top most row. For each variable, the FIRST set is always used and the FOLLOW set is used if the FIRST set contains ε. The rule in the grammar yielding that terminal in the set is entered into the table.

*Example 1:*

Building the Parse Table along the Variable P.

Given:

FIRST(P) = { id, if, while }

To reach these terminals, the following rule must be used:

P →L

So in (P, id) (P, if) and (P, while) enter: P →L

*Example 2:*

Building the Parse table along Variable M.

Given:

FIRST(M) = { ε, ; }

Because ε is in the FIRST set, we will also need to look at the FOLLOW set:

FOLLOW(M) = { $, endif, end }

To reach ‘;’ use the following rule and enter it into (M, ;):

M->;L

To reach ‘$’, ‘endif’, and ‘end’ enter the rule into (M, $), (M, endif) and (M,end):

M->ε

**Part IV:**

See Appendix for the Python 3 implementation of the LL(1) table-driven parser.

This includes an error recovery feature. After stating the error and what the parser expected, a warning is given. The warning indicates that the parser will attempt to continue to parse the input. In order to continue, the parser discards the current token, the source of the error. Based on the current variable on the top of the stack, a list of “acceptable” terminals for the grammar is created. The first token in the input is then compared to the “acceptable” terminals. If there is a match, then the parser proceeds. If the token does not match any of the “acceptable” terminals in the generated list, then the token is discarded from the input. This comparison process between input tokens and “acceptable” terminals continues until there is a match.

**Part V:**

If a rule is to be added or modified, then the appropriate “Table\_Entry” objects must be modified in the “Table.” A “Table\_Entry” is a custom class object which contains a variable, a terminal, and a list of variables and/or terminals that are put on the stack if the top of the stack is the “Table\_Entry” variable and the current input symbol is the terminal of the entry. For example, if we had an entry into the parse table of (S, ) with the rule S->T J, then it we would add the following line:

Table.append(Table\_Entry(‘S’,’\*’,[‘T’,’J’])

Also, the list “variables” and “terminals” must be updated to include any newly introduced or removed variables or terminals.

The Table is built in lines approximately 125 to 190 of the Python File.

**Part VI:**

The parser is tested with various examples.

See Appendix for file contents.

*accept.txt*

This input is accepted by the LL(1) Parser. It utilizes 12 of the 16 terminals available.

accept\_other\_terminals.txt:  
This input is accepted by the LL(1) Parser. It includes the 3 of the 4 terminals not included in accept.txt

*error\_missing\_semicolon.txt:*

This input is rejected by the LL(1) Parser.

**Appendix:**

LL\_1\_Parser.py

accept.txt

if c < id then

id := c

else

while c < c + id do

id := id

end

endif

error\_missing\_semicolon.txt

id := id + c

c := c + id