Homework #4: CMPT-379

Distributed on Thu, Oct 7; Due on Thu, Oct 21

Anoop Sarkar – anoop@cs.sfu.ca

(1) **LR Parsing**: Implement the LR parsing algorithm (see Section 4.7 of the Dragon book). Your program should read in a text file containing a context-free grammar (the textual format of CFGs was described in a previous homework). In addition the LR parser should read in two text files, one containing the *action* table, the other containing the *goto* table. The output of the LR parser should be the parse tree corresponding to the input string. Examples of the action and goto tables as text files are provided in the usual location.

For this homework, instead of starting with the full **Decaf** grammar, your LR parser should be tested with the example grammar shown in Example 4.33 of the Dragon book. For this assignment, *do not create the parsing table using a program*: you can directly use the parsing table for the grammar provided in the Dragon book in Fig 4.31.

Your program should run as follows:

```
cat tokens | parser exprGrammar.txt action.txt goto.txt
```

For the following token input:

```
ID x
PLUS +
ID y
TIMES *
LPAREN (
ID z
RPAREN )
```

and given the expression grammar from Example 4.33 in the Dragon book, written in the usual text format below:

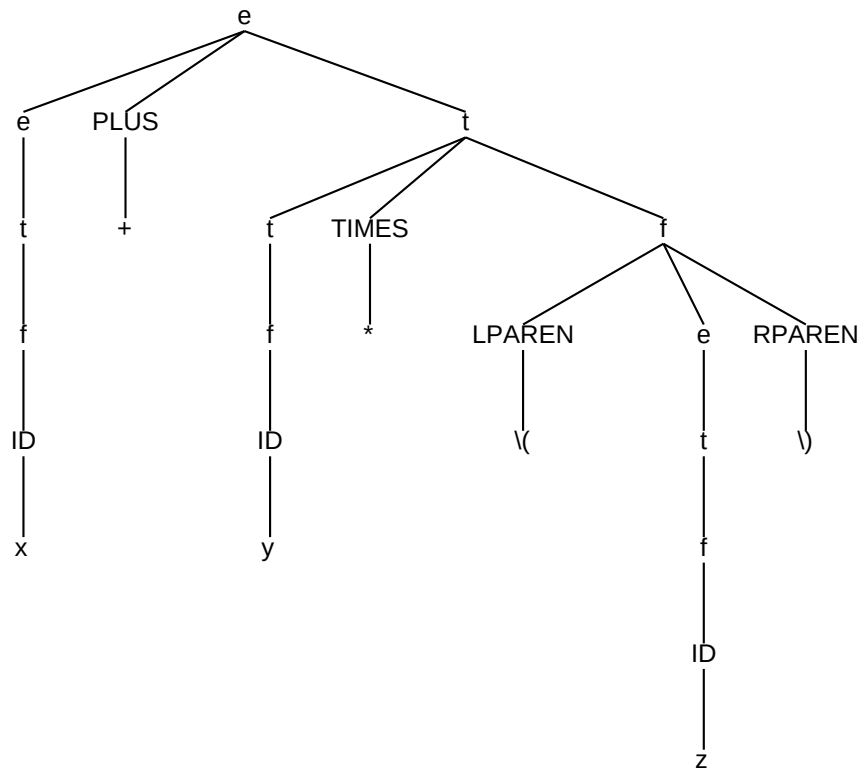
```
e e PLUS t
e t
t t TIMES f
t f
f LPAREN e RPAREN
f ID
```

the LR parser output should be the parse tree in the format shown below. Note the backslash preceding each instance of a literal parenthesis to avoid confusion with the parentheses used to denote the tree structure.

```
(e (e (t (f (ID x))))
   (PLUS +)
   (t (t (f (ID y)))
```

```
(TIMES *)
(f (LPAREN \()
   (e (t (f (ID z))))
   (RPAREN \)))))
```

For simplicity, your output parse tree can be printed out as one parse tree per line, rather than the indented form shown above. In a graphical view, the above parse tree will be as shown below:



Hints on code design:

- Your first step should be to implement a data structure for CFGs. You will need to read in files containing CFGs in the text format.
- When designing the data structure for CFGs you should pay attention to the future use of this data structure. In particular, consider the efficient implementation of the *closure* operation described in Figure 4.33 in the Dragon book.
- When designing the data structure for the *goto* table, notice that the *goto* table is identical to the definition of DFAs.
- When designing the data structure for the *action* table, notice that the *action* table is only a slight variation from the definition of DFAs. Assume a standard indexing scheme for the CFG rules for the *reduce* action when defining the text file for the *action* table.

(2) **Grammar Conversion**: Consider the following fragment of a **Decaf** program:

```
class foo {
  int bar
```

Note that we could continue the above fragment with a field declaration, *or* a method declaration. This issue will not be a problem for a LR parser if the CFG for **Decaf** can be written as an LR(0) or SLR(1) grammar.

Convert the following CFG, which represents a small fragment of **Decaf** syntax, into an LR(0) or SLR(1) grammar. As a result, the LR parsing table for such a grammar will have no shift/reduce or reduce/reduce conflicts.

program	→	CLASS ID LCB field_decl_list method_decl_list RCB
field_decl_list	→	field_decl field_decl_list
field_decl_list	→	ϵ
method_decl_list	→	method_decl method_decl_list
method_decl_list	→	ϵ
field_decl	→	type ID ASSIGN INTCONSTANT SEMICOLON
method_decl	→	return_type ID LPAREN RPAREN
return_type	→	type
return_type	→	VOID
type	→	INT
type	→	BOOL