Compiler Design - HW3

Release: Wednesday 1402/02/20 - Due: Wednesday 1402/3/03 at 11:59pm

Spring 2023 - Sharif University of Technology

---

0 Homework2

0.1

Homework 2 - Question Number 6.

1 Semantic

1.1

Consider we have the following program:

```
1. program S()
2.      var x[1...4], y, real
3.      procedure A (a:  integer)
4.          var z[1...5] integer
5.      end A
6. end S
```

Write corresponding symbol table and scope stack status at end of line 4.
(Symbol table columns: NumberOfLine-Lexeme-proc/var/param/array-No.arg_cell-type-scope)

2 Intermediate Code Generation

2.1

Consider the following grammar:

$$\begin{aligned}
switch\_stmt &\rightarrow switch\ (expression)\ \{case\_stmts\_default\_tmt\} \\
expression &\rightarrow ID \\
case\_stmts &\rightarrow case\_stmts\ case\_stmt \quad \mid \epsilon \\
case\_stmt &\rightarrow case\ NUM\ :\ statement \\
default\_stmt &\rightarrow default\ :\ statement\ \mid \epsilon \\
statement &\rightarrow ID\ =\ expression;
\end{aligned}$$

a) Add the necessary action symbols to the grammar and write the corresponding semantic routines so that the three address codes of this type of expressions can be generated. Note that you can only use the semantic stack for saving the required information, and you can only use three address codes similar to those used in the example routines of Lecture Note 8.
Note that this grammar has no break statement. Therefore, every default case, if present, will be executed.

b) Generate the three address codes of the following input sample using the semantic routines that you have written in part a.

```
switch (a) :  {
     case 1:
       b = a;
     case 2:
       c = a;
     default:
       c = b;
}
```

2.2

The goal of the grammer below is to detect a specific form of algebraic expression.
For instantce, the expression

$$(x+1)*\ \texttt{if}\ (y+z)\ \texttt{then}\ (t+r)\ \texttt{else}\ (w+u)\ \texttt{fi}\ *(s+2)$$

If the expression $(y+z)$ is true, the entire expression equals $(x+1)*(t+r)*(s+2)$, and if it's false, it equals $(x+1)*(w+u)*(s+2)$.
"if", indicates the begining of a conditional statement, and "fi" indicates the end of it.

$$A \to CB$$
$$B \to \epsilon$$
$$B \to +C\ \#\text{add}\ B$$
$$C \to DE$$
$$E \to \epsilon$$
$$E \to *D\ \#\text{mult}\ E$$
$$D \to\ \#\text{pid}\ id$$
$$D \to (A)$$
$$D \to\ \text{if}\ (A)\ \text{then}\ (A)\ \text{else}\ (A)\ \text{fi}$$

a) In the term "$D \to$ if (A) then (A) else (A) fi", insert the action symbols needed and write the necessary semantic procedures.

b) Make changes in the rule "$D \to$ if (A) then (A) else (A) fi", in order for the program to generate bottom up code.

3 Bottom up parsing

3.1

Consider following grammar:

$$S \to Xb \mid aa$$
$$X \to b \mid aXb$$

a) Construct augmented grammar and find Follow sets for all Non-terminals.

b) Draw LR(1) transition diagram.

c) Is this grammar SLR(1)? Explain why or why not.

3.2

A CFG for regular expressions is as follows:

$$
\begin{aligned}
S &\to R && (1)\\
R &\to RR && (2)\\
R &\to R \mid R && (3)\\
R &\to R^* && (4)\\
R &\to \epsilon && (5)\\
R &\to a && (6)\\
R &\to (R) && (7)
\end{aligned}
$$

(note that | is a terminal symbol in the grammar)

(note that ϵ is a terminal symbol in the grammar)

Here are the LR(0) configurating sets for this grammar:

(1) $S \to .R$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .(R)$ $R \to .\epsilon$ $R \to .a$	(2) $S \to R.$ $R \to R.R$ $R \to R.\|R$ $R \to R.*$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .(R)$ $R \to .\epsilon$ $R \to .a$	(3) $R \to RR.$ $R \to R.R$ $R \to R.\|R$ $R \to R.*$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .(R)$ $R \to .\epsilon$ $R \to .a$	(4) $R \to (.R)$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .(R)$ $R \to .a$ $R \to .\epsilon$	(5) $R \to (R.)$ $R \to R.R$ $R \to R.\|R$ $R \to R.*$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .(R)$ $R \to .a$ $R \to .\epsilon$	(6) $R \to R\|.R$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .\epsilon$ $R \to .a$ $R \to .(R)$
(7) $R \to R\|R.$ $R \to R.R$ $R \to R.\|R$ $R \to R.*$ $R \to .RR$ $R \to .R\|R$ $R \to .R*$ $R \to .\epsilon$ $R \to .a$ $R \to .(R)$	(8) $R \to R*.$	(9) $R \to \epsilon.$	(10) $R \to a.$	(11) $R \to (R).$	

a) Construct an SLR(1) parse table for this grammar. The grammar is not SLR(1) and so there will be conflicts in your table. When this happens, list all actions that should be taken in a given state. Please use our numbering for the different reductions and the LR(0) configurating sets.

b) LR(1) is a much stronger parsing algorithm than SLR(1). Would using an LR(1) parser instead of the SLR(1) parser resolve the ambiguities? Why or why not?

c) Let's suppose that we want to resolve the conflicts in this grammar by using our knowledge of the precedence rules for regular expressions. In particular, we know that disjunction ("or") has lowest precedence and is left associative, concatenation has middle precedence and is left associative, and Kleene closure ("star") has highest precedence and is left associative. Given these rules, update the SLR(1) parser table you created in part (i) to resolve all of the conflicts in this grammar.