CS333 : Compiler Design

TUTORIAL 4 (Bottom Up Parsing : SLR Parsing) Posted : February 12, 2024

P1. We have seen that left recursive rules in a context free grammar are not suitable for top down parsing. Does bottom up parsing have an analogous problem ? Examine any bottom up SLR(1) parser with respect to recursive rules, left recursive or right recursive, and justify your observations.

P2. The stack contents and the input symbol at some point during parsing by a shift reduce parser is shown below with the nonterminal C on top of the stack. The terminals on the stack are {a. b, c, d} and the nonterminals are {A, B, D, C}

STACK : a A b B c D d C

Identify all the potential handles in the stack.

P3. Explain why the blank entries in the Goto part of a SLR (1) parsing table are not error entries in the usual sense of similar entries in the Action part.

P4. A LR(0) grammar is one for which the corresponding LR(0) parsing table has no conflicts; a reduce action in a state of an LR(0) parser is performed on all terminals T. Consider the following context free grammar.

$S \rightarrow X$
$X \rightarrow Ma \mid bMc \mid dc \mid bda$
$M \rightarrow d$

(a) Determine whether the grammar as given is LR(0). Report all the conflicts if your answer is in the negative.

(b) Recall that in SLR(1), a reduce action, $A \rightarrow \alpha$, is performed for all terminals in FOLLOW(A). Determine whether the grammar as given is SLR(1). Report all the conflicts if your answer is in the negative.

P5 (a) Identify which of the grammars, as given below, is LR(0) and/or SLR(1) and why ?

(i)
$S \rightarrow A\, B\, c$
$A \rightarrow a \mid \varepsilon$
$B \rightarrow b \mid \varepsilon$

(ii)
$E \rightarrow -E \mid (E) \mid V\, R$
$V \rightarrow id\, T$
$R \rightarrow -E \mid \varepsilon$
$T \rightarrow (E) \mid \varepsilon$

(b) If the grammars as given are not SLR(1), can they be transformed to an equivalent SLR(1) ?

P6. We would like to process program fragments, as given below, for the purpose of parsing.

```
int  a [10] [20];
int  b [10] [20];
int i, j;
i = 10;
j = 16;
b[i+1] [ j+2] = a [i-1] [j+1] + a [i ] [j – 2] + 25;
```

Write a CFG so that such code fragment can be generated. Your grammar should be able to (i) generate declarations of scalars and arrays as denoted by the first 3 lines above, and (ii) also generate assignments involving arrays and scalars, as given in the last 3 lines of the sample code above. Assume that an array is stored in row-major representation , if such information is necessary.

(b) Determine whether the grammar written by you in part (a), will admit a SLR(1) parser by constructing the automaton or otherwise.

(c) Construct the parse tree for the program fragment of part (a) using your grammar and a SLR(1) parser.

P7. Construct a grammar, if possible, such that it

- **(a)** is SLR(1) but not LL(1)
- **(b)** is LL(1) but not SLR(1)

********* End of Tutorial Sheet 4 *********