CS 302 : Implementation of Programming Languages
TUTORIAL 4 (Error Analysis); February 16, 2017

The manually constructed LALR(1) parsing table for the pointer grammar G

0) S' --> S 1,2) S --> L **=** R | R

3, 4) L --> ***** R | **id** 5) R --> L

is given below.

State	*Action*				*Goto*		
	id	*****	**=**	**$**	S	L	R
0	s5	s4	error	error	1	2	3
1	error	error	error	**acc**	-	-	-
2	error	error	s6	r5	-	-	-
3	error	error	error	r2	-	-	-
4	s5	s4	error	error	-	8	7
5	error	error	r4	r4	-	-	-
6	s5	s4	error	error	-	8	9
7	error	error	r3	r3	-	-	-
8	error	error	r5	r5	-	-	-
9	error	error	error	r1	-	-	-

The following error routines along with diagnostic messages is available to you; feel free to add more appropriate ones with justifications.

e1 : missing symbol id **e2** : missing operator * **e3** : missing operator =

e4 : extraneous symbol id **e5 : extraneous symbol *** **e6 : extraneous symbol =**

(a) The parsing table is required to be augmented with error routines. The possible valid symbols at an error entry are to be analysed to determine the possible recovery action (identify the most appropriate error function)

(b) For each error routine used, write the recovery action to be taken manually so that the parser can resume normal parsing after performing the action specified by you. One error routine is done as illustration.

Error routine	*Diagnostic message*	*Recovery action*
e1	missing symbol id	Push id and state 5 on stack
......	.....	.....

P1. Do the steps (a) and (b) for the states 0 and 1 of the parsing table. Write the compact LALR(1) parsing table for these states as would be generated by yacc. Show the working of the parser for 2 strings, one with error arising out of these 2 states and the other without. Produce an input string for which the manually constructed LA LR(1) parser and yacc generated parser differ in their error detection capability.

P2. Do the steps (a) and (b) for the states 2 and 3 of the parsing table. Write the compact LALR(1) parsing table for these states as would be generated by yacc. Show the working of the parser for 2 strings, one with error arising out of these 2 states and the other without. Produce an input string for which the manually constructed LA LR(1) parser and yacc generated parser differ in their error detection capability.

P3. Do the steps (a) and (b) for the states 4 and 5 of the parsing table. Write the compact LALR(1) parsing table for these states as would be generated by yacc. Show the working of the parser for 2 strings, one with error arising out of these 2 states and the other without. Produce an input string for which the manually constructed LA LR(1) parser and yacc generated parser differ in their error detection capability.

P4. Do the steps (a) and (b) for the states 6 and 7 of the parsing table. Write the compact LALR(1) parsing table for these states as would be generated by yacc. Show the working of the parser for 2 strings, one with error arising out of these 2 states and the other without. Produce an input string for which the manually constructed LA LR(1) parser and yacc generated parser differ in their error detection capability.

P5. Do the steps (a) and (b) for the states 8 and 9 of the parsing table. Write the compact LALR(1) parsing table for these states as would be generated by yacc. Show the working of the parser for 2 strings, one with error arising out of these 2 states and the other without. Produce an input string for which the manually constructed LA LR(1) parser and yacc generated parser differ in their error detection capability.

P6. The following error-production rule is added to the pointer grammar G,

6) S --> error

Construct and report the new states that are added to the LALR(1) parsing table for G. Produce erroneous strings that show, if possible, this error recovery enabled parser to perform both insertion and deletion type of recovery.

Examine the behavior of the parser, if yyerrok is invoked after reduction by the error rule, i. e.,

6) S --> error {yyerrok;}

Make changes to your grammar / actions so that as many errors in the input are reported back by the parser as possible.

P7. The following error-production rule is added to the pointer grammar G,

6) S --> L error R

Construct and report the new states that are added to the LALR(1) parsing table for G. Produce erroneous strings that show, if possible, this error recovery enabled parser to perform both insertion and deletion type of recovery.

Examine the behavior of the parser, if yyerrok is invoked after reduction by the error rule, i.e.,

6) S --> L error R {yyerrok;}

Make changes to your grammar / actions so that as many errors in the input are reported back by the parser as possible.

P8. The following error-production rule is added to the pointer grammar G,

6) L --> error R

Construct and report the new states that are added to the LALR(1) parsing table for G. Produce erroneous strings that show, if possible, this error recovery enabled parser to perform both insertion and deletion type of recovery.

Examine the behavior of the parser, if yyerrok is invoked after reduction by the error rule, i. e.,

6) L --> error R {yyerrok;}

Make changes to your grammar / actions so that as many errors in the input are reported back by the parser as possible.

P9. The following error-production rule is added to the pointer grammar G,

6) L --> error

Construct and report the new states that are added to the LALR(1) parsing table for G. Produce erroneous strings that show, if possible, this error recovery enabled parser to perform both insertion and deletion type of recovery.

Examine the behavior of the parser, if yyerrok is invoked after reduction by the error rule, i.e.,

6) L --> error {yyerrok;}

Make changes to your grammar / actions so that as many errors in the input are reported back by the parser as possible.

P10. Consider the declaration grammar used in earlier tutorial sheets, which is used to declare one or more scalars (without initialization) of the basic types, integer, float and char.

(a) Generate LALR(1) parser using yacc and show the automaton as given in “y.output”.

(b) Now add error productions and yyerrok to the grammar to generate a parser with error recovery enabled and show the new automaton.

(c) Construct a few strings with errors and produce the parser dump in debugging mode for each input.

(d) Annotate the dump for each input to explain the movement of the parser between normal parsing and recovery modes and also the issuing (or suppression) of error messages.

Released on February 16, 2017
Supratim Biswas

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