CSC4180 Assignment 3

120090102 CAO Ruoxi

April 2024

Q1: Resolve Ambiguity for Micro Language's Grammar

1. Sorry, I can't. I scan all the grammar rules in Micro grammar and delinmate its left recursive problem:

```
start ::= program SCANEOF
program ::= BEGIN_ statement_list END

statement_list ::= statement statement_list'
statement_list' ::= statement statement_list'
statement_list' ::= ''

statement ::= ID ASSIGNOP expression SEMICOLON
statement ::= READ LPAREN id_list RPAREN SEMICOLON
statement ::= WRITE LPAREN expr_list RPAREN SEMICOLON
id_list ::= ID
if_list ::= id_list COMMA ID

expr_list ::= expression expr_list'
expr_list' ::= COMMA expression expr_list'
expr_list' ::= ''

expression ::= primary expression_tail
expression_tail ::= PLUSOP primary expression_tail
expression_tail ::= MINUSOP primary expression_tail
expression_tail ::= ''

primary ::= LPAREN expression RPAREN
primary ::= ID
primary ::= INTLITERAL
```

   I enter them into the LL(1) web demo and encounter no conflicts. Actually except for the left recursive problem, the origin grammar has no problem. Meanwhile, the left recursive problem may not lead to ambiguity.

2. (a) No. Multiple defined entries in an LL(1) parsing table indicate conflicts for an LL(1) parser, showing that the grammar is not LL(1) due to FIRST and FOLLOW set overlaps. However, this doesn't automatically mean the grammar is ambiguous. Ambiguity and being non-LL(1) are distinct issues; a grammar can be non-LL(1) yet unambiguous.

   (b) Yes. If there is no multiple defined entry in the LL(1) parsing table of the grammar, then for any pair of a nonterminal and a token, the parser chooses the only production rule, which eliminates the ambiguity.

Q2: Simple LL(1) and LR(0) Parsing Exercises

LL(1) Grammar

1.
```
E ::= T N
N ::= ''
N ::= + E
T ::= int M
M ::= ''
M ::= * T
T ::= ( E )
```

2. Two tables:

Nonterminal	Nullable?	First	Follow
S	✕	int, (	
E	✕	int, (	), $
N	✓	+	), $
T	✕	int, (	+,), $
M	✓	*	+,), $

Figure 1: NullableFirstFollowTable

	$	+	int	*	(	)
S			S ::= E $		S ::= E $	
E			E ::= T N		E ::= T N	
N	N ::= ε	N ::= + E				N ::= ε
T			T ::= int M		T ::= (E)	
M	M ::= ε	M ::= ε		M ::= * T		M ::= ε

Figure 2: Transition Table

3. The parse tree:

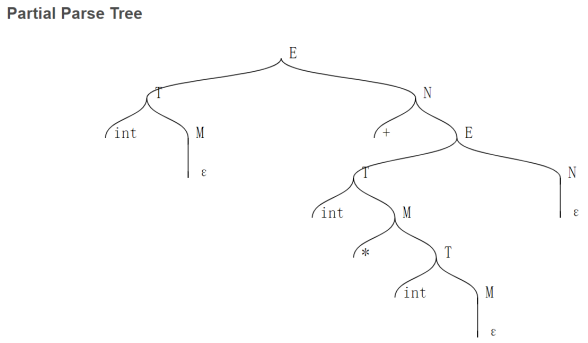

Figure 3: The Parse Tree

LR(0) Grammar

1. Play with the LR(0) web demo with its default grammar:

2. (a) Why the LL(1) grammar fail to pass LR(0) parser?
LR(0) parsers easily come across reduce-reduce or shift-reduce conflicts, where the parser cannot decide which production rule to reduce with. This type of conflict is not a problem for LL(1) parsers because they always select the unique production rule based on the current input and predictive parsing table.

State	Item set
0	{S' ::= • S $, S ::= • (L), S ::= • x}
1	{S ::= (• L), L ::= • S, L ::= • L ; S, S ::= • (L), S ::= • x}
2	{S ::= x • }
3	{S' ::= S • $}
4	{S' ::= S $ • }
5	{L ::= S • }
6	{S ::= (L •), L ::= L • ; S}
7	{S ::= (L) • }
8	{L ::= L ; • S, S ::= • (L), S ::= • x}
9	{L ::= L ; S • }

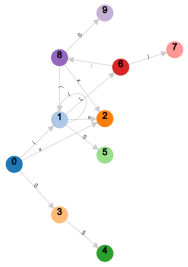

Figure 4: the LR(0) automaton

(b) Why the LR(0) grammar fail to pass LL(1) parser?
The configurations in LR(0) states depend on the current state and the current token only. LR(0) parsers make decisions without looking ahead at any symbols. They do not care about looking ahead one symbol. LL(1) parsers decide which production rule to use by looking ahead by only one symbol. This means if it cannot decide which production rule to use by looking ahead one symbol, the grammar is not LL(1).

State	Item set
0	{S' ::= • E $, E ::= • T E', T ::= • F T', F ::= • (E), F ::= • id}
1	{F ::= (• E), E ::= • T E', T ::= • F T', F ::= • (E), F ::= • id}
2	{F ::= id • }
3	{S' ::= E • $}
4	{E ::= T • E', E' ::= • + T E', E' ::= • }
5	{T ::= F • T', T' ::= • * F T', T' ::= • }
6	{T' ::= * • F T', F ::= • (E), F ::= • id}
7	{T ::= F T' • }
8	{T' ::= * F • T', T' ::= • * F T', T' ::= • }
9	{T' ::= * F T' • }
10	{E' ::= + • T E', T ::= • F T', F ::= • (E), F ::= • id}
11	{E ::= T E' • }
12	{E' ::= + T • E', E' ::= • + T E', E' ::= • }
13	{E' ::= + T E' • }
14	{S' ::= E $ • }
15	{F ::= (E •)}
16	{F ::= (E) • }

Figure 5: LL(1) grammar in LR(0) Parser

2. Nullable/First/Follow Table and Transition Table

Nonterminal	Nullable?	First	Follow
S	×	(, x	), $, ;
L	×	(, x	), ;

	$	(	)	x	;
S		S ::= (L) S ::= S $		S ::= x S ::= S $	
L		L ::= S L ::= L ; S		L ::= S L ::= L ; S	

Figure 6: LR(0) grammar in LL(1) Parser