

Bottom-Up Parsing

Question 1

Show that the following simplified if-then-else grammar is not LR(0)
 $S \rightarrow ictS \mid ictSeS \mid a$

Answer

One way to show this is to show that the given grammar is ambiguous. This proves that the grammar is not LR(0), since LR(0) grammars are non-ambiguous. We show that the given grammar is ambiguous by showing two different derivations/parse trees. Figure 1 shows two different derivations.

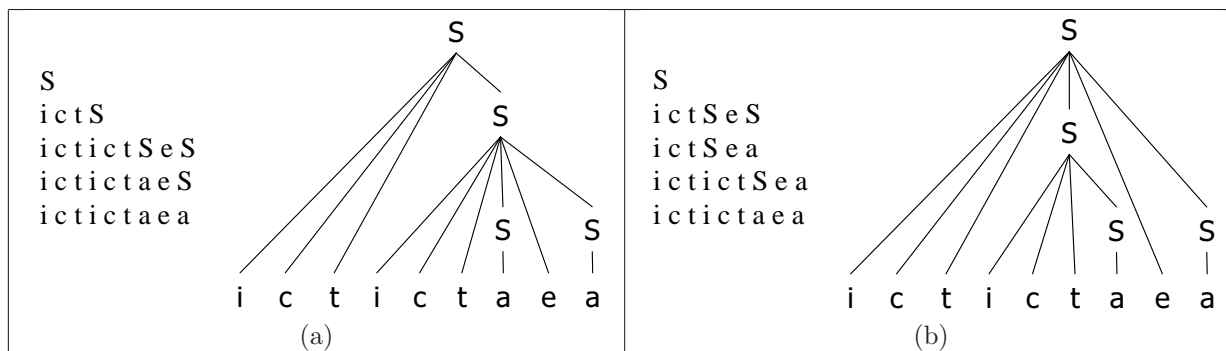


Figure 1: Two different derivations of the input `ictictaea`: (a) Leftmost derivation; and (b) Rightmost derivation

Question 2

Investigate the two possibilities to patch the parsing table in the above grammar, and their consequences. In both cases show the parsing of nested if-then-else statement: `ictictaea`

Conclude that it is not possible to parse this grammar with LR(0), even with conflict resolution.

Bonus (10 points): Construct an LR(1) parsing table (Modern Compiler Design book, pages 165–170) and show that after conflict resolution it is possible to parse the input string given above.

Answer

We start by adding an extra rule $S' \rightarrow S\$$ to make the resulting transition diagram and parsing table more readable (this is not a must).

Using the algorithms for LR(0) transition diagram, we construct the transition diagram shown in Figure 2. There is a shift/reduce conflict in state 8 between the shift item $S \rightarrow ictS \bullet eS$, and a reduce item $S \rightarrow ictS \bullet$. (The existence of the conflict also shows that the grammar is not LR(0).)

Resolving the conflict in favor of shifting results with the table shown in Figure 3(a), and resolving the conflict in favor of reduce results with the table shown in Figure 3(b).

Using the parsing table in Figure 3(a), we get the following run for the LR(0) parsing algorithm on the input `ictictaea` \$:

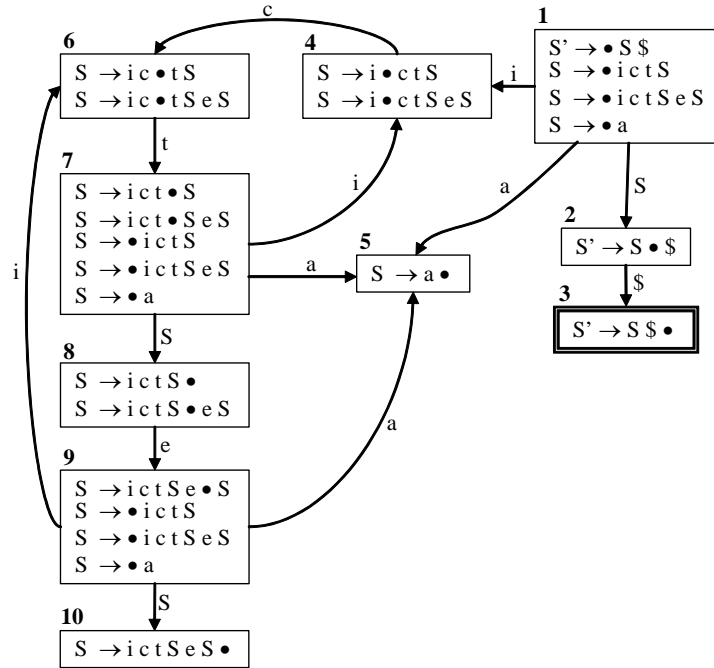


Figure 2: Transition diagram for the simplified if-then-else grammar with the extra rule $S' \rightarrow S\$$. States are sets of LR(0) items. State numbers are displayed above them. State 3 is the accepting state when the stack contains only the symbol S'

	i	c	t	e	a	\$	S	Action
1	4				5		2	shift
2						3		shift
3	$S' \rightarrow S\$$ (accept)							reduce
4		6						shift
5	$S \rightarrow a$							shift
6			7					shift
7	4				5		8	shift
8				9				shift
9							10	shift
10	$S \rightarrow ictSeS$							reduce

(a)

	i	c	t	e	a	\$	S	Action
1	4				5		2	shift
2						3		shift
3	$S' \rightarrow S\$$ (accept)							reduce
4		6						shift
5	$S \rightarrow a$							shift
6			7					shift
7	4				5		8	shift
8	$S \rightarrow ictS$							reduce
9							10	shift
10	$S \rightarrow ictSeS$							reduce

(b)

Figure 3: Parsing tables for the simplified if-then-else grammar: (a) Solving the conflict in favor of shifting (the terminal 'e' and transitioning to state 9); and (b) solving the conflict in favor of reducing (by the rule $S \rightarrow ictS$). The empty slots correspond to transition to the error state

Stack	Input	Action
S_1	i c t i c t a e a \$	shift
$S_1 i S_4$	c t i c t a e a \$	shift
$S_1 i S_4 c S_6$	t i c t a e a \$	shift
$S_1 i S_4 c S_6 t S_7$	i c t a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4$	c t a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6$	t a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7$	a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7 a S_5$	e a \$	reduce $S \rightarrow a$
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7 S S_8$	e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7 S S_8 e S_9$	a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7 S S_8 e S_9 a S_5$	\$	reduce $S \rightarrow a$
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7 S S_8 e S_9 S S_{10}$	\$	reduce $S \rightarrow i c t S e S$
$S_1 i S_4 c S_6 t S_7 S S_8$	\$	error at token \$

Using the parsing table in Figure 3(b), we get the following run:

Stack	Input	Action
S_1	i c t i c t a e a \$	shift
$S_1 i S_4$	c t i c t a e a \$	shift
$S_1 i S_4 c S_6$	t i c t a e a \$	shift
$S_1 i S_4 c S_6 t S_7$	i c t a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4$	c t a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6$	t a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7$	a e a \$	shift
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_7 a S_5$	e a \$	reduce $S \rightarrow a$
$S_1 i S_4 c S_6 t S_7 i S_4 c S_6 t S_8 S S_8$	e a \$	reduce $S \rightarrow i c t S$
$S_1 i S_4 c S_6 t S_7 S S_8$	e a \$	reduce $S \rightarrow i c t S$
$S_1 S S_2$	e a \$	error at token e

Since the input is accepted by the language defined by the grammar (as demonstrated by the derivations in the answer to question 1), we have shown that it is impossible to parse this grammar with LR(0) with any conflict resolution. Therefore this grammar is not LR(0).

Bonus (10 points): Construct an LR(1) parsing table (Modern Compiler Design book, pages 165–170) and show that after conflict resolution it is possible to parse the input string given above.

Answer

Figure 4 shows the LR(1) transition diagram for the given grammar. State 13 contains a conflict (as expected, since the given grammar is ambiguous), which we resolve in favor of shifting.

Figure 5 shows the LR(1) parsing table. The table is in fact 3-dimensional, that is, in each state the action is determined according to the current stack symbol and the lookahead token. Since most decisions are identical for all lookahead tokens, the table only shows the differences by dividing the cells where the lookahead token matters. Notice how in state 13 we resolve the shift-reduce conflict by shifting on the lookahead token e and reducing on the lookahead token $\$$. Note that to save space we do not show state 0 with the rule $S' \rightarrow S\$$.

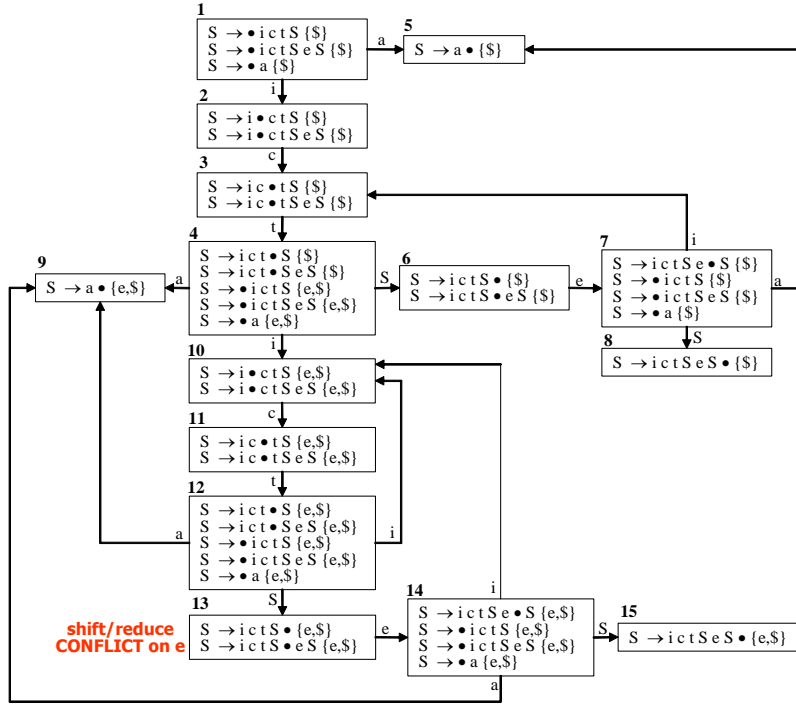


Figure 4: LR(1) Transition diagram for the simplified if-then-else grammar with the extra rule $S' \rightarrow S\$$. States are sets of LR(1) items. State numbers are displayed above them.

	i	c	t	e	a	S
1	s2				s5	
2		s3				
3			s4			
4	s10				s9	g6
5					i c t e a \$ S → ictS	
6				s7		i c t e a \$ S → ictS
7	s3				s5	g8
8						i c t e a \$ S → ictSeS
9					i c t e a \$ S → a S → a	
10		s11				
11			s12			
12	s10				s9	g13
13						i c t e a \$ s14 S → ictS
14	s10				s9	g15
15						i c t e a \$ S → ictSeS S → ictSeS

Figure 5: LR(1) parsing table for the given grammar simplified if-then-else grammar with the extra rule $S' \rightarrow S\$$. The symbols in the top row correspond to stack symbol and the symbols in the divided cells correspond to lookahead tokens

Using the parsing table in Figure 5, we get the following run:

Stack	Input	Action
S_1	i c t i c t a e a \$	shift
$S_1 i S_2$	c t i c t a e a \$	shift
$S_1 i S_2 c S_3$	t i c t a e a \$	shift
$S_1 i S_2 c S_3 t S_4$	i c t a e a \$	shift
$S_1 i S_2 c S_3 t S_4 i S_{10}$	c t a e a \$	shift
$S_1 i S_2 c S_3 t S_4 i S_{10} c S_{11}$	t a e a \$	shift
$S_1 i S_2 c S_3 t S_4 i S_{10} c S_{11} t S_{11}$	a e a \$	shift
$S_1 i S_2 c S_3 t S_4 i S_{10} c S_{11} t S_{11} a S_9$	e a \$	reduce $S \rightarrow a$ (lookahead=e)
$S_1 i S_2 c S_3 t S_4 i S_4 c S_6 t S_8 S S_{13}$	e a \$	shift (lookahead=e)
$S_1 i S_2 c S_3 t S_4 i S_4 c S_6 t S_8 S S_{13} e S_{14}$	a \$	shift
$S_1 i S_2 c S_3 t S_4 i S_4 c S_6 t S_8 S S_{13} e S_{14} a S_9$	\$	reduce $S \rightarrow a$ (lookahead=\$)
$S_1 i S_2 c S_3 t S_4 i S_4 c S_6 t S_8 S S_{13} e S_{14} S S_{15}$	\$	reduce $S \rightarrow i c t S e S$ (lookahead=\$)
$S_1 i S_2 c S_3 t S_4 S S_6$	\$	reduce $S \rightarrow i c t S$ (lookahead=\$)
S	\$	stop

We see that after resolving the conflict in state 13 the parser is able to accept the input.

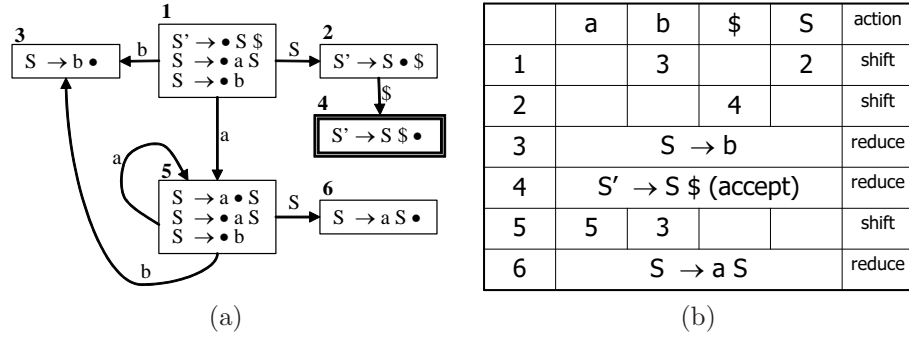


Figure 6: (a) Transition diagram, and (b) Parsing table

Question 3

Develop an LR(0) parser for the grammar: $S \rightarrow a S \mid b$.

Answer

We add the extra rule $S' \rightarrow S\$$ to make the resulting transition diagram and parsing table more readable (again, this is not a must). Figure 6 shows the transition diagram and the parsing table for the grammar.

Question 4

Apply your parser to an input 'aaab'. What is the conclusion that you draw from the usage of right-recursive grammars in LR parsers?

Answer

Using the parsing table in Figure 6(b), we get the following run:

Stack	Input	Action
S_1	a a a b \$	shift
$S_1 a S_5$	a a b \$	shift
$S_1 a S_5 a S_5$	a b \$	shift
$S_1 a S_5 a S_5 a S_5$	b \$	shift
$S_1 a S_5 a S_5 a S_5 b S_3$	\$	reduce $S \rightarrow b$
$S_1 a S_5 a S_5 a S_5 S S_6$	\$	reduce $S \rightarrow a S$
$S_1 a S_5 a S_5 S S_6$	\$	reduce $S \rightarrow a S$
$S_1 a S_5 S S_6$	\$	reduce $S \rightarrow a S$
$S_1 S S_2$	\$	shift
$S_1 S S_2 \$ S_4$		reduce $S' \rightarrow S \$$
$S_1 S'$		stop

We observe that the depth of the stack is proportional to the length of the input string, which means that the space consumption of the algorithm is linear in the size of the input. For left-recursion, the space consumption is constant in the size of the input.