CS164 Programming Languages and Compilers **Spring 2024**

Written Assignment 2 Solutions

Assigned: February 8 **Due:** February 22 at 3:00pm

1. Use left-factoring and/or elimination of left recursion to convert the following grammars into LL(1) grammars. You may assume that these grammars are unambiguous.

   (a) **Solution:**

$$
\begin{aligned}
S &\rightarrow\ BS' \mid S' \\
S' &\rightarrow\ aS' \mid \epsilon \\
B &\rightarrow\ CB' \\
B' &\rightarrow\ xCB' \mid \epsilon \\
C &\rightarrow\ c \mid \epsilon
\end{aligned}
$$

   (b) **Solution:**

$$
\begin{aligned}
L &\rightarrow\ \mathit{int}\ L' \mid (L) \\
L' &\rightarrow\ +L'' \mid \epsilon \\
L'' &\rightarrow\ L \mid *\ L
\end{aligned}
$$

   (c) **Solution:**

$$
\begin{aligned}
A &\rightarrow B\ A \mid \epsilon \\
B &\rightarrow \mathit{bool} \mid \mathit{and\ bool}
\end{aligned}
$$

2. Consider the following grammar describing a certain sort of expression language:

$$\begin{array}{rcl} A & \rightarrow & A \, * \, B \mid C \\ B & \rightarrow & B \, + \, C \mid C \\ C & \rightarrow & (\; A \;) \mid int \end{array}$$

The nonterminals are A, B, and C, while the terminals are $*$, $+$, (,), and int.

(a) Eliminate left recursion from this grammar.

Solution:

$$\begin{array}{rcl} A & \rightarrow & C \; A' \\ A' & \rightarrow & * \; B \; A' \mid \epsilon \\ B & \rightarrow & C \; B' \\ B' & \rightarrow & + \; C \; B' \mid \epsilon \\ C & \rightarrow & (A) \mid int \end{array}$$

(b) Give the First and Follow sets for each nonterminal in the grammar obtained in part (a).

Solution:

	$First$	$Follow$
A	$int, ($	$\$,)$
A'	$*, \epsilon$	$\$,)$
B	$int, ($	$\$, *,)$
B'	$+, \epsilon$	$\$, *,)$
C	$int, ($	$\$, *, +,)$

(c) Using this information, construct an LL parsing table for the grammar obtained in part (a).

Solution:

	$*$	$+$	(	)	int	\$
A			CA'		CA'	
A'	$*BA'$			ϵ		ϵ
B			CB'		CB'	
B'	ϵ	$+CB'$		ϵ		ϵ
C			(A)		int	

(d) Suppose we generated an LL parser for the grammar using the table you constructed. What would go wrong if it tried to parse the following input string?

$$+int * int * (int + int)$$

(That is, when we get an error, how much of the input string has been consumed, and what is the parser trying to do?)

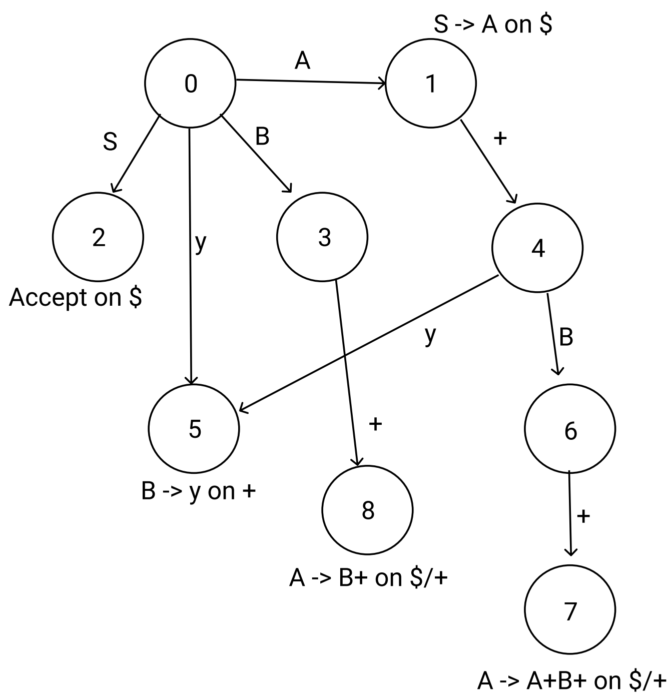
Solution: None of the input string will be consumed. When it sees the first + it will fail because the production A has no entry in the table for lookahead token +.

3. Consider the following LR(1) grammar:

$$
\begin{array}{lcl}
S & \rightarrow & A \\
A & \rightarrow & A + B + \;\;|\;\; B + \\
B & \rightarrow & y
\end{array}
$$

(Each '+' is a separate token.)

and its corresponding DFA:



Complete the table below, showing the trace of an LR(1) parser (which uses the DFA above) on the input provided. The "Stack" column must show the stack (with the top at right), the "Input" column shows the not-yet-processed input terminals, and the "Action" column must show whether the parser performs a shift action or a reduce action or accepts the input. In the case of a reduce action, please indicate which production is used.

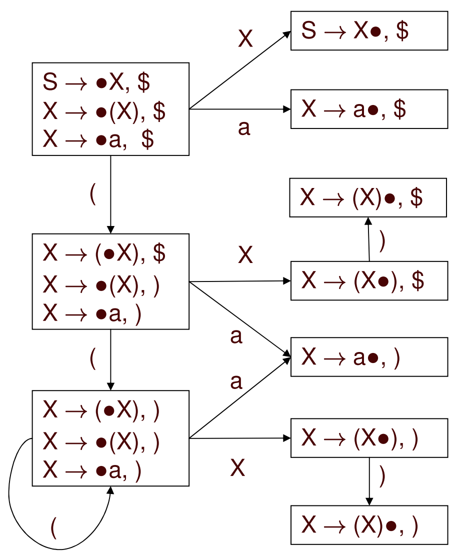
Stack (with top at right)	Input	Action
	► y + + y + $	shift
y	► + + y + $	reduce $B \rightarrow y$
B	► + + y + $	shift
B +	► + y + $	reduce $A \rightarrow B+$
A	► + y + $	shift
A +	► y + $	shift
A + y	► + $	reduce $B \rightarrow y$
A + B	► + $	shift
A + B +	► $	reduce $A \rightarrow A + B+$
A	► $	reduce $S \rightarrow A$
S	► $	accept

4. Consider the following grammar with start state S:

$$
\begin{aligned}
S &\rightarrow X \\
X &\rightarrow (X) \mid a
\end{aligned}
$$

The following figure shows a skeleton of the LR(1) parsing DFA for this grammar.

Solution:



(a) Complete the DFA skeleton. You need to fill in the LR(1) items for each state of the DFA, add new transitions, and label each transition. (You should not add any new states though.)
Hint: One of the states has a self-loop.

(b) Is the grammar LR(1)? Is the grammar LR(2)? Is the grammar LL(1)? Why or why not?
Solution:
The grammar is LR(1) because its LR(1) parsing DFA for the grammar has no conflicts.
The grammar is LR(2) because it is LR(1).
The grammar is LL(1) because its LL(1) parsing table, as shown below, has no conflicts.

	a	(	)	\$
S	**X**	**X**		
X	**a**	**(X)**		

(c) **Optional challenge question - WILL NOT BE GRADED:** Use your DFA to parse $((((a))))$. Show the sequence of shift/reduce steps (by filling a table like that in question 3).
Solution:
The "Stack" column shows the stack (with the top at right), the "Input" column shows the not-yet-processed input terminals, and the "Action" column shows whether the parser performs a shift action or a reduce action or accepts the input.

Stack (with top at right)	Input	Action
	▶ ((((a)))) \$	shift
(	▶ (((a)))) \$	shift
((	▶ ((a)))) \$	shift
(((	▶ (a)))) \$	shift
((((	▶ a)))) \$	shift
((((a	▶)))) \$	reduce $X \to a$
((((X	▶)))) \$	shift
((((X)	▶))) \$	reduce $X \to (X)$
(((X	▶))) \$	shift
(((X)	▶)) \$	reduce $X \to (X)$
((X	▶)) \$	shift
((X)	▶) \$	reduce $X \to (X)$
(X	▶) \$	shift
(X)	▶ \$	reduce $X \to (X)$
X	▶ \$	reduce $S \to X$
S	▶ \$	accept

5. Is the following grammar ambiguous? Is it LR(1)? Explain both of your answers.

$$
\begin{array}{rcl}
S & \to & X\ Y\ t \\
S & \to & A\ Z\ n \\
X & \to & b \\
Y & \to & L \\
Z & \to & L \\
L & \to & a \\
A & \to & m
\end{array}
$$

Solution: The grammar is unambiguous because it contains exactly two strings and each string has a unique derivation.

$$
\begin{array}{c}
S \Rightarrow X\ Y\ t \Rightarrow b\ L\ t \Rightarrow bat \\
S \Rightarrow A\ Z\ n \Rightarrow m\ L\ n \Rightarrow man
\end{array}
$$

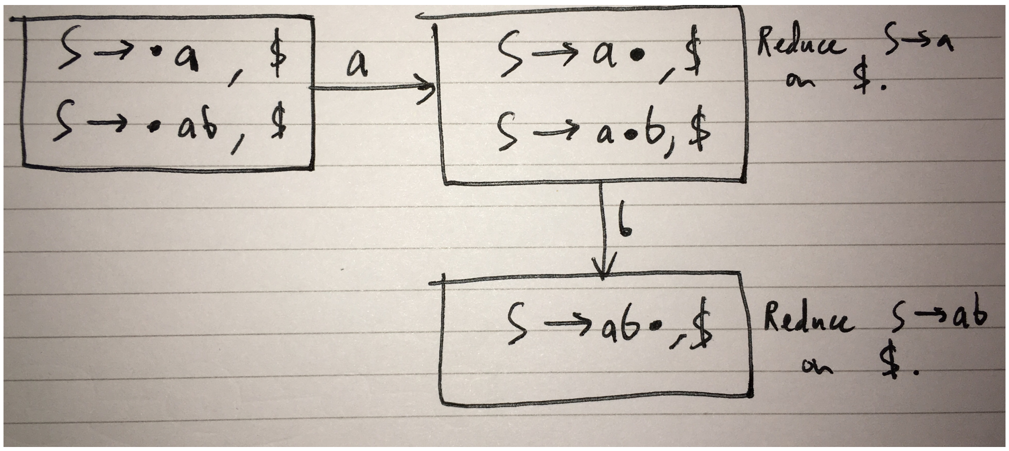
The grammar is LR(1). The two strings the grammar contains have different starting letter, so there can never be any conflicts once the parser has reduced the first letter to X or A (or rejected the string).

6. In each of the following cases, give as simple a grammar as you can.

(a) Give an LR(1) grammar that is not LL(1). Explain why your grammar is LR(1) and not LL(1).
Solution:

$$\begin{aligned} S &\rightarrow a \\ S &\rightarrow ab \end{aligned}$$

The grammar is not LL(1) because it is not left factored. The grammar is LR(1) because the LR(1) parsing DFA for the grammar, as shown below, has no conflicts.



(b) Give an unambiguous grammar that is not LR(1). Explain why your grammar is unambiguous and not LR(1).
Hint: In each of the above cases, there exists a grammar that generates a language with only two strings.
Solution:

$$\begin{aligned} S &\rightarrow Uab | Vac \\ U &\rightarrow d \\ V &\rightarrow d \end{aligned}$$

The grammar is unambiguous because it contains exactly two strings and each string has a unique derivation.

$$\begin{aligned} S &\Rightarrow Uab \Rightarrow dab \\ S &\Rightarrow Vac \Rightarrow dac \end{aligned}$$

The grammar is not LR(1) because of the following reason. Consider an input that begins with "da". After shifting the d, the parser must reduce to either U or V. However, based solely on the lookahead character of a, we cannot decide which of U or V is correct, leading to a reduce/reduce conflict. Note that the grammar is LR(2).