

Solutions to Written Assignment 4

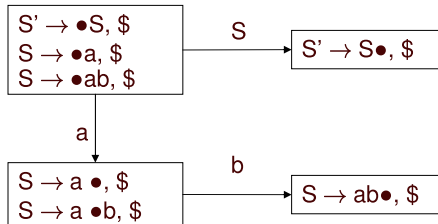
1. In each of the following cases, give as simple a grammar as you can.

- (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.

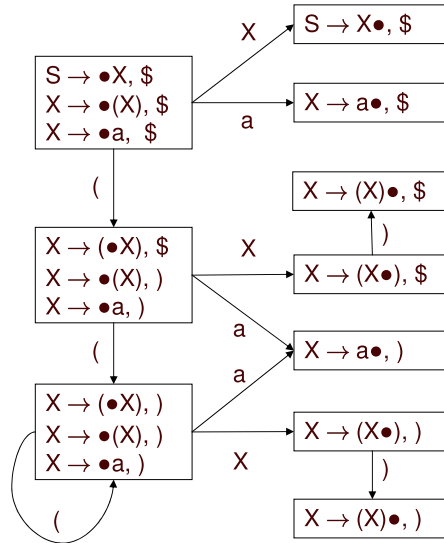
$$\begin{aligned} S &\Rightarrow Uab \Rightarrow dab \\ S &\Rightarrow Vac \Rightarrow dac \end{aligned}$$

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).

2. Consider the following grammar with start state S :

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

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



- (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) Use your DFA to parse $((((a))))$ - show the sequence of shift/reduce steps.

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
((▶)))) \$	reduce $X \rightarrow a$
((▶)))) \$	shift
((▶)))) \$	reduce $X \rightarrow (X)$
((▶)))) \$	shift
((▶)))) \$	reduce $X \rightarrow (X)$
((▶)))) \$	shift
((▶)))) \$	reduce $X \rightarrow (X)$
((▶)))) \$	shift
((▶)))) \$	reduce $X \rightarrow (X)$
((▶)))) \$	shift
((▶)))) \$	reduce $X \rightarrow (X)$
((▶)))) \$	shift
((▶)))) \$	reduce $S \rightarrow X$
((▶)))) \$	accept