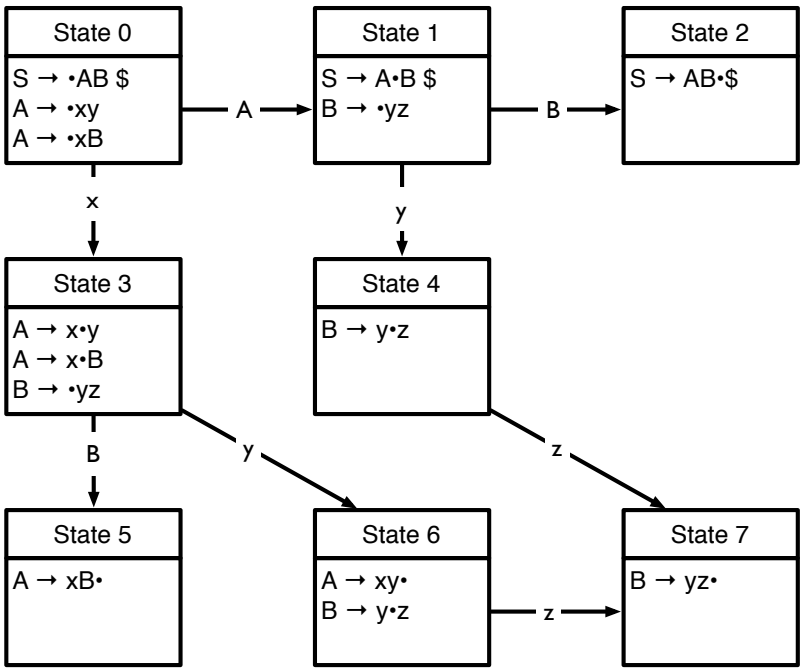
ECE 573
Problem Set 4: LR(1) parsers

1. Build the CFSM for this grammar. Is it an LR(0) grammar? Why or why not?

$$
\begin{array}{lrcl}
1. & S & \rightarrow & AB\$ \\
2. & A & \rightarrow & xy \\
3. & A & \rightarrow & xB \\
4. & B & \rightarrow & yz
\end{array}
$$

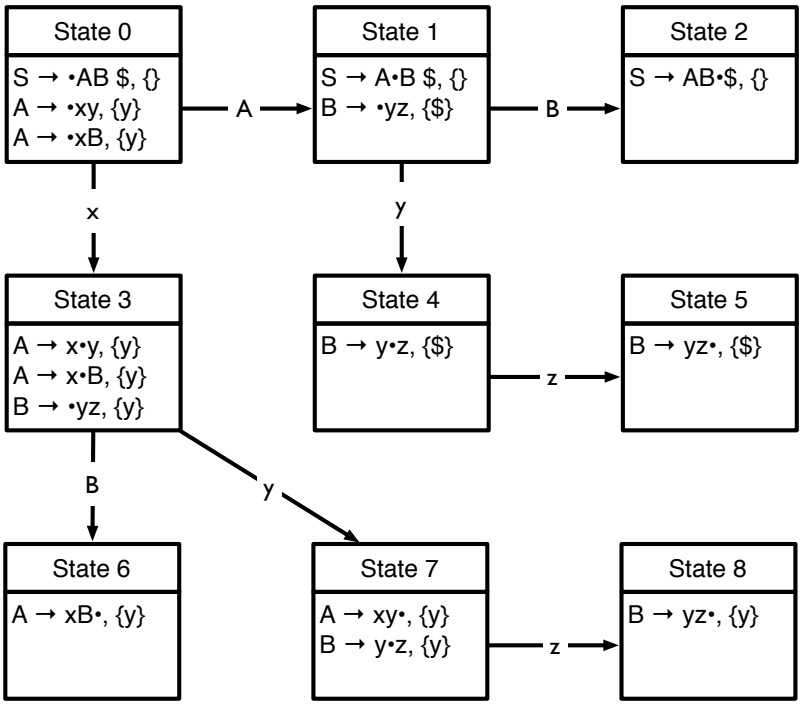
Answer:



This is not an LR(0) grammar; state 6 has a shift/reduce conflict.

2. Build the LR(1) machine for the grammar in the previous problem. Is this an LR(1) grammar? Why or why not?

Answer:



This is an LR(1) grammar; the shift/reduce conflict is gone. In state 7, we will reduce if the lookahead is y (recall: reductions are conditioned on the lookahead of the configuration) while we will shift if the lookahead is z (shifts are conditioned on the next token in the configuration).

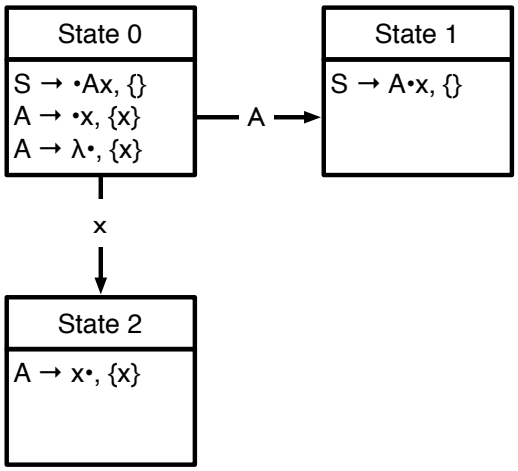
Note that the LR(1) machine has one extra state than the CFSM. This is because the lookaheads differ for the two ways we can get to the configuration $B \rightarrow yz\cdot$. An *LALR* machine would combine states 5 and 8, because they only differ in their lookahead.

3. In the project, we saw an example of a grammar that was not LR(1) because the grammar was ambiguous (there are multiple parse trees for a given string). Ambiguity, however, is not necessary for a grammar to not be LR(1). Give an example of a grammar that is *un*ambiguous, but not LR(1).

Answer:

1. $S \rightarrow Ax\$$
2. $A \rightarrow x$
3. $A \rightarrow \lambda$

Note that there are only two strings in this grammar (x and xx), and each string can only be derived in one way, so the grammar is unambiguous. However, this is what the LR(1) machine for this grammar looks like:



Note that in State 0, we must decide whether to shift x or reduce $A \rightarrow \lambda$. However, the lookahead for $A \rightarrow \lambda$ is x, which means that we don't know whether to choose shift or reduce when the lookahead is x. This problem can be solved either by rewriting the grammar to eliminate the conflict or by using 2 tokens of lookahead.