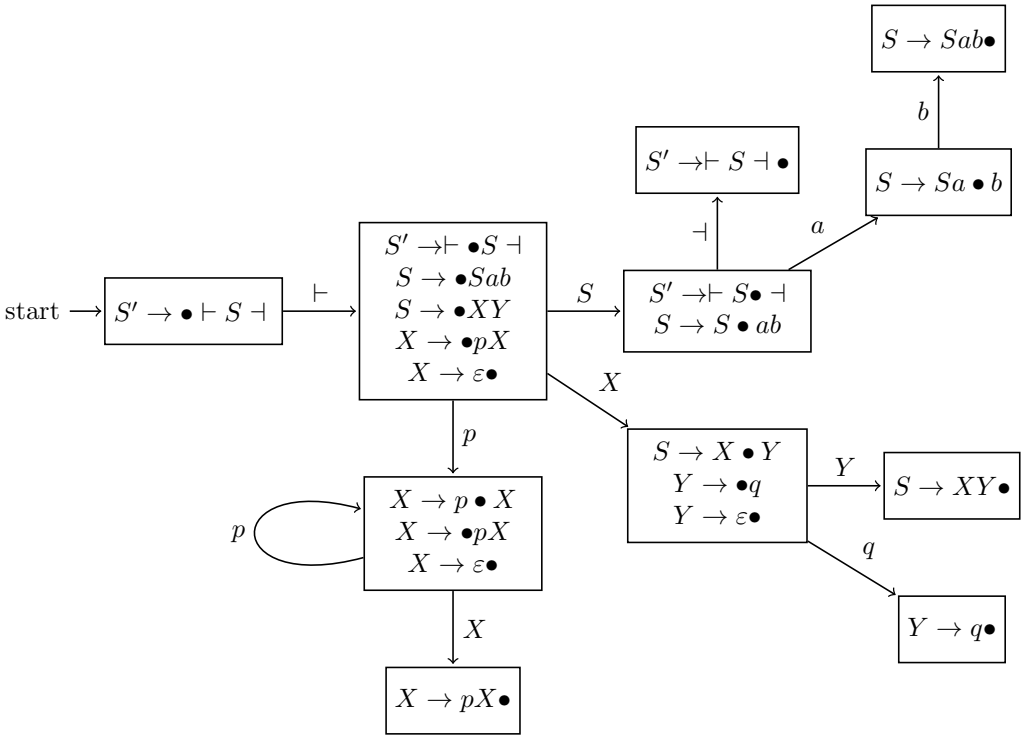
CS 241 – Week 9 Tutorial Solutions

Bottom-Up Parsing: LR Parsing

Spring 2015

LR Parsing Solutions

1. We just apply the algorithm from class. We start with our initial transition and then add states based upon whether a terminal or non-terminal is the next thing seen and move our $\bullet$ along as we consume symbols. Remember that when a state has an item with $X \to \alpha \bullet B$ that we include all the B productions in the same state with $\bullet$ at the leftmost part of the RHS.



We can see that we have two **shift-reduce** errors in our machine. We know that this grammar is not LR(0), however, it is SLR(1). It is left as an exercise for you to verify this by using *Follow* sets and augmenting the machine.

2. *Note: the grammar is numbered from 0 instead of 1 because the generated SLR(1) parsing table uses rule numbers starting from 0.*

   Algorithm (explained in words):

   Begin with 0 on the state stack and an empty symbol stack. Look at the top of the state stack, then

look at the first symbol of the remaining input, and find corresponding shift or reduce action in the SLR(1) table.

shift n:

Pop the symbol off the input and push it on the symbol stack. Then push the state number n onto the state stack. Look up the next action.

reduce n:

Look up rule n in the grammar. Count the number of symbols on the right side of the rule. Pop this number of symbols from the symbol stack, and the same number of states from the state stack. Push the symbol on the left side of the rule on to the symbol stack. Look up the state currently on top of the state stack (after the pops), then look up the symbol you just pushed to the symbol stack. You should find a "shift n" action. Push that state number n to the state stack. Look up the next action.

We accept when we have shifted $\dashv$. We could do a final reduction to S', but this is unnecessary. This is why the provided SLR(1) table does not contain any actions for state 6.

Trace of the algorithm:

Action	State Stack	Symbol Stack	Remaining Input
initialize	0		$\vdash$pqab$\dashv$
shift $\vdash$, 1	0 1	$\vdash$	pqab$\dashv$
shift p, 5	0 1 5	$\vdash$p	qab$\dashv$
reduce X $\rightarrow$	0 1 5 9	$\vdash$pX	qab$\dashv$
reduce X $\rightarrow$ pX	0 1 3	$\vdash$X	qab$\dashv$
shift q, 7	0 1 3 7	$\vdash$Xq	ab$\dashv$
reduce Y $\rightarrow$ q	0 1 3 2	$\vdash$XY	ab$\dashv$
reduce S $\rightarrow$ XY	0 1 8	$\vdash$S	ab$\dashv$
shift a, 4	0 1 8 4	$\vdash$Sa	b$\dashv$
shift b, 10	0 1 8 4 10	$\vdash$Sab	$\dashv$
reduce S $\rightarrow$ Sab	0 1 8	$\vdash$S	$\dashv$
shift $\dashv$, 6	0 1 8 6	$\vdash$S$\dashv$	

The reversed right canonical derivation is obtained by simply reading the rules used in reductions from top to bottom, and then adding the rule for the final reduction to S' at the end. Here is the reversed derivation in the CFG-R format:

X
X p X
Y q
S X Y
S S a b
S' $\vdash$ S $\dashv$

Notice if we apply these production rules from bottom to top we get a right-canonical derivation:

S'
$\Rightarrow \vdash$ S $\dashv$
$\Rightarrow \vdash$ S a b $\dashv$
$\Rightarrow \vdash$ X Y a b $\dashv$
$\Rightarrow \vdash$ X q a b $\dashv$
$\Rightarrow \vdash$ p X q a b $\dashv$

$\Rightarrow \vdash$ p q a b $\dashv$

We can easily obtain a parse tree from this derivation by looking at what rule was used to expand each non-terminal symbol in the derivation.