Exercise 2: Table Generation (2)

Given the following grammar, create the parser table for bottom-up syntax analysis and compute the guide symbols for error recovery.

```
S = a.
S = a X.
X = b.
X = b X c.
```

Is this grammar LR(0), LR(1) or LALR(1)? Give reasons for your answer.

Solution

The productions are first numbered consecutively and a pseudo-production for S' is added:

```
0   S' = S #.
1   S = a.
2   S = a X.
3   X = b.
4   X = b X c.
```

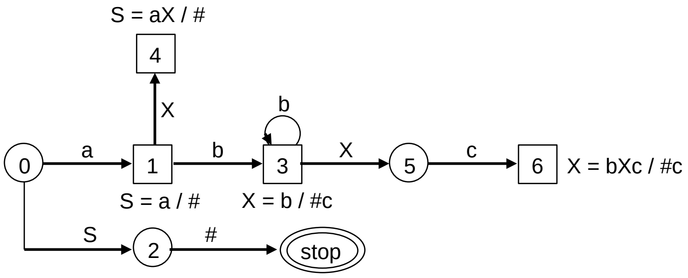
When creating the table, the first terminal symbol action in each state determines the guide symbol (shown in the rightmost column).

0	S'= . S #		shift	**a**	1	**a**
	S = . a	/ #	shift	S	2	
	S = . a X	/ #				
1	S = a .	/ #	red	#	1 (S = a)	#
	S = a . X	/ #	shift	b	3	
	X = . b	/ #	shift	X	4	
	X = . b X c	/ #				
2	S' = S . #		acc	#		#
3	X = b .	/ #c	red	#,c	3 (X = b)	#
	X = b . X c	/ #c	shift	b	3 (!)	
	X = . b	/ c	shift	X	5	
	X = . b X c	/ c				
4	S = a X .	/ #	red	#	2 (S = a X)	#
5	X = b X . c	/ #c	shift	**c**	6	**c**
6	X = b X c .	/ #c	red	**#,c**	4 (X = b X c)	#

This results in the following parser table:

	a	b	c	#	S	X	guide
0	s1				s2		a
1		s3		r1		s4	#
2				acc			#
3		s3	r3	r3		s5	#
4				r2			#
5			s6				c
6			r4	r4			#

The corresponding pushdown automaton looks like this:



The grammar is not LR(0) because both shift and reduce actions occur in states 1 and 3. We therefore need a lookahead symbol to distinguish between these actions.

It is LR(1) because in each state, the lookahead symbol can be used to decide whether a shift or a reduce action should be executed.

It is also LALR(1) because no LR conflict was introduced by the merging of states. In state 3, there is a shift to a (potentially new) state, which was merged with state 3 because the kernel items are the same.