## Exercise 3: Table Generation (3)

The following grammar is LR(1), but not LALR(1):

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

Create the parser tables for both LR(1) and LALR(1) analysis and show where the conflict occurs. Also draw the PDA for the LR(1) and LALR(1) analysis and explain why the conflict occurs in the LALR(1) table but not in the LR(1) table.

**Solution**

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

```
0  S' = S #.
1  S = d X b.
2  S = d Y a.
3  S = X a.
4  S = Y b.
5  X = c.
6  Y = c.
```

We first build the LR(1) table:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | S'= . S # | | shift | d | 1 | |
| | S = . d X b | / # | shift | c | 2 | |
| | S = . d Y a | / # | shift | X | 3 | |
| | S = . X a | / # | shift | Y | 4 | |
| | X = . c | / a | shift | S | 5 | |
| | S = . Y b | / # | | | | |
| | Y = . c | / b | | | | |
| 1 | S = d . X b | / # | shift | c | 6 | |
| | X = . c | / b | shift | X | 7 | |
| | S = d . Y a | / # | shift | Y | 8 | |
| | Y = . c | / a | | | | |
| 2 | X = . c | / a | red | a | 5 | (X = c) |
| | Y = . c | / b | red | b | 6 | (Y = c) |
| 3 | S = X . a | / # | shift | a | 9 | |
| 4 | S = Y . b | / # | shift | b | 10 | |
| 5 | S' = S . # | | acc | # | | |
| 6 | X = c . | / b | red | b | 5 | (X = c) |
| | Y = c . | / a | red | a | 6 | (Y = c) |
| 7 | S = d X . b | / # | shift | b | 11 | |
| 8 | S = d Y . a | / # | shift | a | 12 | |
| 9 | S = X a . | / # | red | # | 3 | (S = X a) |
| 10 | S = Y b . | / # | red | # | 4 | (S = Y b) |
| 11 | S = d X b . | / # | red | # | 1 | (S = d X b) |
| 12 | S = d Y a . | / # | red | # | 2 | (S = d Y a) |

There is no LR conflict here. However, we see that the states 2 and 6 have the same kernel items (albeit with different successor symbols). When we create the LALR(1) table, these two states are merged.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | S'= . S # | | shift | d | 1 | |
| | S = . d X b | / # | shift | c | 2 | |
| | S = . d Y a | / # | shift | X | 3 | |
| | S = . X a | / # | shift | Y | 4 | |
| | X = . c | / a | shift | S | 5 | |
| | S = . Y b | / # | | | | |
| | Y = . c | / b | | | | |
| 1 | S = d . X b | / # | shift | c | **2 (!)** | |
| | X = . c | / b | shift | X | 6 | |
| | S = d . Y a | / # | shift | Y | 7 | |
| | Y = . c | / a | | | | |
| 2 | X = . c | / **ab** | red | **a,b** | 5 | (X = c) |
| | Y = . c | / **ab** | red | **a,b** | 6 | (Y = c) |
| 3 | S = X . a | / # | shift | a | 8 | |
| 4 | S = Y . b | / # | shift | b | 9 | |
| 5 | S' = S . # | | acc | # | | |
| 6 | S = d X . b | / # | shift | b | 10 | |
| 7 | S = d Y . a | / # | shift | a | 11 | |
| 8 | S = X a . | / # | red | # | 3 | (S = X a) |
| 9 | S = Y b . | / # | red | # | 4 | (S = Y b) |
| 10 | S = d X b . | / # | red | # | 1 | (S = d X b) |
| 11 | S = d Y a . | / # | red | # | 2 | (S = d Y a) |

In state 1, c is used to go to the existing state 2 and not to a new state 6 as in the LR(1) table. By merging states 2 and 6 from the LR(1) table, the successors of the items are merged. Both items in state 2 now have a and b as possible successors, and these symbols can be used to reduce according to both production 5 (X = c) and 6 (Y = c). We therefore have a *reduce-reduce* conflict that did not occur in the LR(1) table. The pushdown automata for LR(1) and LALR(1) make this even better visible.