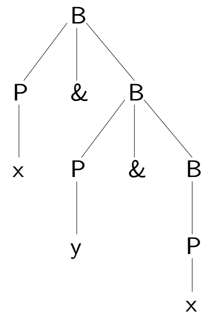
Programmation Avancée et Compilation 2001
(Solutions)

6 Bottom-Up Parsing

Exercise 12

(a) The following syntax-tree shows a derivation of x&y&x (in fact it is a unique tree, because the given grammar is not ambiguous):



The word can be reduced by the following steps:

x&y&x
P&y&x
P&P&x
P&P&P
P&P&B
P&B
B

The corresponding rightmost derivation is simply the reverse of the reduction steps:

$$\mathsf{B} \Rightarrow \mathsf{P\&B} \Rightarrow \mathsf{P\&P\&B} \Rightarrow \mathsf{P\&P\&P} \Rightarrow \mathsf{P\&P\&x} \Rightarrow \mathsf{P\&y\&x} \Rightarrow \mathsf{x\&y\&x}.$$

The word x&y& cannot be derived. The reason is that & is followed by B in the only production it appears, and B is not nullable (see also the discussion in Exercise 1b).

A shift-reduce parser tries to produce a rightmost derivation by performing a reduction in reverse order (see the reduction steps in part (a) of this exercise). It (obviously!) has two actions, shift and reduce:

- In a shift-operation, the parser puts the current input symbol on the stack, while entering the next state (expressed by a corresponding subscript).
- In a reduce-operation, the parser pops as many symbols from the stack as there are on the right-hand side of the applied production (throwing away the states noted in between), and performs a shift operation on the left-hand side, starting from the state emerging after having removed all necessary symbols and states.

(b) The completed table reads as follows:

Stack	Input	Action
$_1$	x&y&x$	shift
$_1\mathsf{x}_5$	&y&x$	reduce P ::= x
$_1\mathsf{P}_3$	&y&x$	shift
$_1\mathsf{P}_3\&_4$	y&x$	shift
$_1\mathsf{P}_3\&_4\mathsf{y}_6$	&x$	reduce P ::= y
$_1\mathsf{P}_3\&_4\mathsf{P}_3$	&x$	shift
$_1\mathsf{P}_3\&_4\mathsf{P}_3\&_4$	x$	shift
$_1\mathsf{P}_3\&_4\mathsf{P}_3\&_4\mathsf{x}_5$	$	reduce P ::= x
$_1\mathsf{P}_3\&_4\mathsf{P}_3\&_4\mathsf{P}_3$	$	reduce B ::= P
$_1\mathsf{P}_3\&_4\mathsf{P}_3\&_4\mathsf{B}_7$	$	reduce B ::= P & B
$_1\mathsf{P}_3\&_4\mathsf{B}_7$	$	reduce B ::= P & B
$_1\mathsf{B}_2$	$	accept

(c) Now let's have a look at an analogous execution of the stack-automaton on "x&y&":

Stack	Input	Action
$_1$	x&y&x$	shift
$_1\mathsf{x}_5$	&y&$	reduce P ::= x
$_1\mathsf{P}_3$	&y&$	shift
$_1\mathsf{P}_3\&_4$	y&$	shift
$_1\mathsf{P}_3\&_4\mathsf{y}_6$	&$	reduce P ::= y
$_1\mathsf{P}_3\&_4\mathsf{P}_3$	&$	shift
$_1\mathsf{P}_3\&_4\mathsf{P}_3\&_4$	$	???

In state 4, the automaton does not have any rule that would be applicable to the empty input, it can only perform shift-operations. Hence, the parser gets stuck. At this moment, error-recovery should be invoked. (What do you think such an error-message should tell, and how could it be achieved?)

Remark: In Exercise 11, you have constructed a top-down parser by hand. Note that the construction of a bottom-up parser for a typical programming-language by hand would require too much effort and is prone to errors. This is where parser-generators like JavaCUP enter the game.