Exercise 0.1 Model the following classical measuring problem in Promela, in such a way that SPIN finds a solution as a violation of an assertion, temporal logic formula, or similar.

> Alice wants to give exactly 5 liters of water to Bob. She has a large supply of water and two buckets: one taking 10 liters, and one taking 7 liters. Her only means of measuring volumes of water is to pour water into, out of, or between buckets, knowing their volume (10 and 7 liters). How can she end up with exactly 5 liters in one of the buckets?

Exercise 0.2 Show how to define a stack data structure in Promela (a stack is a container of objects, which is Last-in-First-out, i.e., the last inserted element is the first to be removed). You should provide a recepy for which variables to declare to represent the stack. You should define two operations

- `push(x)` pushes the value of the variable `x` onto the stack, in one atomic operation. If the stack is full, the operation should be unexecutable.
- `pop(x)` pops the top value from the stack and assigns its value to `x`. If the stack is empty, the operation should be unexecutable.

You can, e.g., use `inline` definitions for these operations. You can make your own decision about the types of elements in the stack, how to define the capacity of the stack, etc.

Exercise 0.3 Below is a Promela model of a simple system with two concurrent processes that manipulate a shared variable `N`.

```
byte    N = 0 ;

active [2] proctype P() {
        byte temp , i ;
        i = 1 ;
        do
        :: i < 10  -> temp = N ; temp++ ; N = temp ; i++
        :: else -> break
        od
}
```

What is the *smallest* possible value that the variable `N` can assume when both processes have terminated?

Szymanski's protocol

In this section, we will introduce a generic system model which is intended to capture the behaviour of many existing mutual exclusion protocols, e.g. Dijkstra's mutual exclusion problem, Lamport's bakery algorithm, Burn's protocol, Szymanski's algorithm. In our model, a program consists of an arbitrary number of identical processes, ordered in a

linear array. The process behaviours are defined through a finite set of *actions*. An action represents a change of local state of a process. An action may be conditioned on both the local state of the process, and the *context* in which it may take place. The context represents a global condition on the local states of the rest of processes inside the system. The ordering of the processes may reflect the actual physical ordering (e.g. Szymanski's algorithm), or the values assigned to local variables inside processes (e.g. the ticket given to each process during the execution of Lamport's bakery protocol).

An idealized version of Szymanski's mutual exclusion algorithm can be given as follows. In the algorithm, an arbitrary number of processes compete for a critical section. The local state of each process consists of a control state ranging over the integers from 1 to 7 and of two boolean flags, w and s. A pseudo-code version of the actions of any process i could look as follows:

$$
\begin{array}{ll}
1: & \textbf{await } \forall j : j \neq i : \neg s_j \\
2: & w_i, s_i := \mathit{true}, \mathit{true} \\
3: & \textbf{if } \exists j : j \neq i : (pc_j \neq 1) \wedge \neg w_j \\
 & \qquad \textbf{then } s_i := \mathit{false}, \text{ goto } 4 \\
 & \qquad \textbf{else } w_i := \mathit{false}, \text{ goto } 5 \\
4: & \textbf{await } \exists j : j \neq i : s_j \wedge \neg w_j \textbf{ then } w_i, s_i := \mathit{false}, \mathit{true} \\
5: & \textbf{await } \forall j : j \neq i : \neg w_j \\
6: & \textbf{await } \forall j : j < i : \neg s_j \\
7: & s_i := \mathit{false}, \text{ goto } 1
\end{array}
$$

For instance, according to the code at line 6, if the control state of a process i is 6, and if the context is that the value of s is *false* in all processes to the left, then the control state of i may be changed to 7. In a similar manner, according to the code at line 4, if the control state of a process i is 4, and if the context is that there is at least another process (either to the right or to the left of i) where the value of s is *true* and the value of w is *false*, then the control state and the values of w and s in i may be changed to 5, *false*, and *true*, respectively.