



**Tutorials for Verification**  
**Exercise sheet 2**

**Exercise 1: Peterson’s algorithm revisited**

Consider the following variant of Peterson’s algorithm, where the assignment to  $b_i$  and  $x$  is not atomic:

```

while true {
    .....
    nc :   b1 := true;
    req :   x := 2;
    wt :   wait until(x = 1 ∨ ¬b2);
    cs :   ... critical section ...
           b1 := false;
    .....
}

```

```

while true {
    .....
    nc :   b2 := true;
    req :   x := 1;
    wt :   wait until(x = 2 ∨ ¬b1);
    cs :   ... critical section ...
           b2 := false;
    .....
}

```

Give the program graphs and the transition system for the interleaving of these programs. Is mutual exclusion still ensured?

Consider another variant of the algorithm where the statements at locations  $nc$  and  $req$  of the first process are swapped (the second process remains as it is). Is mutual exclusion still ensured?

**Exercise 2: Alternating bit protocol**

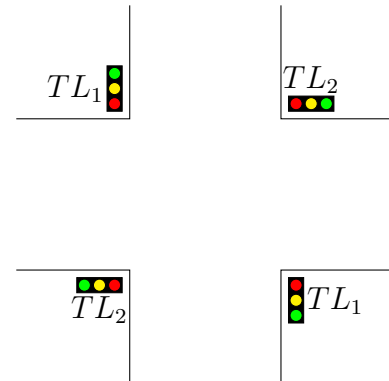
In the lecture a model of the alternating bit protocol was introduced. Let the capacity of both channels be nonzero. Convince yourself informally that for any execution, either there is a time point after which no more messages arrive or all messages arrive (perhaps with repetitions) in the correct order. Is it so easy to give a mathematical proof, for example, by induction? What problems do you see? How would a machine proceed to show correctness?

**Exercise 3: Handshaking**

It is obvious to see that for every fixed set of synchronization actions the pairwise handshaking operator is associative: if  $H = H'$ , then  $(T_1 \parallel_H T_2) \parallel_{H'} T_3 = T_1 \parallel_H (T_2 \parallel_{H'} T_3)$ . What about  $H \neq H'$ ?

### Exercise 4: Traffic lights

Consider a simple traffic light for a crossing as depicted on the right. The two traffic lights labelled with  $TL_1$  always show the same color; likewise the lights labelled with  $TL_2$ . The goal is to describe the behaviour of  $TL_1$  and  $TL_2$  by two transition systems.



- Create two transition system  $TS_1$  and  $TS_2$  for traffic lights, one for each direction of a crossing. Insert suitable actions on which these system can synchronise, so that at least one of the lights shows red in each state.
- Compute the transition system  $TS_1 || TS_2$ . Show that this system is safe.
- Extend the system by pedestrian traffic lights.

### Exercise 5: Webshop

In this exercise, a web shop should be designed as channel system. The web shop should receive orders on the channel *order* with the domain  $Customer \times Products$ . Then it should send a request to the Schufa on channel *check* with domain  $Customer$  to check if the customer is creditworthy. The reply is then sent on the channel *result* with domain  $\{0, 1\}$ . If the result is 0, the web shop sends a *denial* (channel without any data value). Otherwise, the web shop sends the product over the channel *deliver* with domain  $Customer \times Products$ .

- Implement the web shop as program graph over the channels mentioned above and over auxiliary variables.
- After the web shop was running for some time, it was found that the number of requests in the queue *order* is growing. After some investigation, the cause for this was found: The algorithm handles all orders sequentially. At each time there is only one event sent to the Schufa and there is a large delay before the *reply* is received. It would be better to immediately continue with the next order instead of waiting for the *reply*.  
Split up the shop into two components, one sending out checks to the Schufa and another one handling the replies. You probably need another channel to enqueue the unfinished orders.
- Convince yourself of the correctness of the shop: every customer gets its product but only if he is creditworthy.

### Exercise 6: One more Peterson (\*)

Reconsider the algorithm from Exercise 1, where the inner loops are changed to “wt: **wait until**  $x=1$ ” for the first and “wt: **wait until**  $x=2$ ” for the second process. Would you use this variant of the algorithm? Why or why not?