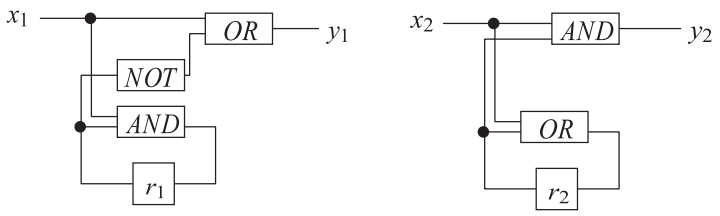
2.6 Exercises

EXERCISE 2.1. Consider the following two sequential hardware circuits:



Questions:

(a) Give the transition systems of both hardware circuits.

(b) Determine the reachable part of the transition system of the synchronous product of these transition systems. Assume that the initial values of the registers are r_1=0 and r_2=1.

EXERCISE 2.2. We are given three (primitive) processes P_1, P_2, and P_3 with shared integer variable x. The program of process $P_i$is as follows:

Algorithm 1 Process P_i

for $k_i = 1, \ldots, 10$ **do**
  LOAD(x);
  INC(x);
  STORE(x);
od

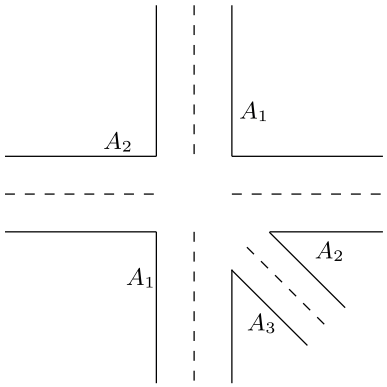
That is, P_i executes ten times the assignment $x := x{+}1$. The assignment $x := x{+}1$ is realized using the three actions LOAD(x), INC(x) and STORE(x). Consider now the parallel program:

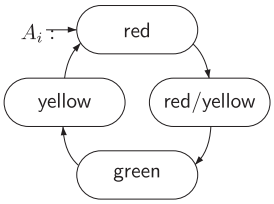
Algorithm 2 Parallel program P

$x := 0$;
$P_1 \;||\; P_2 \;||\; P_3$

Question: Does P have an execution that halts with the terminal value $x = 2$?

EXERCISE 2.3. Consider the following street junction with the specification of a traffic light as outlined on the right.





(a) Choose appropriate actions and label the transitions of the traffic light transition system accordingly.

(b) Give the transition system representation of a (reasonable) controller C that switches the green signal lamps in the following order: $A_1, A_2, A_3, A_1, A_2, A_3, \ldots$.
(*Hint: Choose an appropriate communication mechanism.*)

(c) Outline the transition system $A_1 \| A_2 \| A_3 \| C$.

EXERCISE 2.4. Show that the handshaking operator $\|$ that forces two transition systems to synchronize over their common actions (see Definition 2.26 on page 48) is associative. That is, show that

$$(TS_1 \| TS_2) \| TS_3 \quad = \quad TS_1 \| (TS_2 \| TS_3)$$

where TS_1, TS_2, TS_3 are arbitrary transition systems.

EXERCISE 2.5. The following program is a mutual exclusion protocol for two processes due to Pnueli [118]. There is a single shared variable s which is either 0 or 1, and initially 1. Besides, each process has a local Boolean variable y that initially equals 0. The program text for process P_i $(i = 0, 1)$ is as follows:

l0: **loop forever do**
  begin
  l1: Noncritical section
  l2: $(y_i, s) := (1, i)$;
  l3: **wait until** $((y_{1-i} = 0) \ \vee \ (s \neq i))$;
  l4: Critical section
  l5: $y_i := 0$
end.

Here, the statement $(y_i, s) := (1, i)$; is a *multiple assignment* in which variable $y_i := 1$ and $s := i$ is a single, atomic step.

Questions:

(a) Define the program graph of a process in Pnueli's algorithm.

(b) Determine the transition system for each process.

(c) Construct their parallel composition.

(d) Check whether the algorithm ensures mutual exclusion.

(e) Check whether the algorithm ensures starvation freedom.

The last two questions may be answered by inspecting the transition system.

EXERCISE 2.6. Consider a stack of nonnegative integers with capacity n (for some fixed n).

(a) Give a transition system representation of this stack. You may abstract from the values on the stack and use the operations *top*, *pop*, and *push* with their usual meaning.

(b) Sketch a transition system representation of the stack in which the concrete stack content is explicitly represented.

EXERCISE 2.7. Consider the following generalization of Peterson's mutual exclusion algorithm that is aimed at an arbitrary number n ($n \geqslant 2$) processes. The basic concept of the algorithm is that each process passes through n "levels" before acquiring access to the critical section. The concurrent processes share the bounded integer arrays $y[0..n{-}1]$ and $p[1..n]$ with $y[i] \in \{1, \ldots, n\}$ and $p[i] \in \{0, \ldots, n{-}1\}$. $y[j] = i$ means that process i has the lowest priority at level j, and $p[i] = j$ expresses that process i is currently at level j. Process i starts at level 0. On requesting access to the critical section, the process passes through levels 1 through $n{-}1$. Process i waits at level j until either all other processes are at a lower level (i.e., $p[k] < j$ for all $k \neq i$) or another process grants process i access to its critical section (i.e., $y[j] \neq i$). The behavior of process i is in pseudocode:

while true **do**
  ... noncritical section ...
  forall $j = 1, \ldots, n{-}1$ **do**
    $p[i] := j$;
    $y[j] := i$;
    wait until $(y[j] \neq i) \vee \left(\bigwedge_{0<k\leqslant n, k\neq i} p[k] < j\right)$
  od
  ... critical section ...
  $p[i] := 0$;
od

Questions:

(a) Give the program graph for process i.

(b) Determine the number of states (including the unreachable states) in the parallel composition of n processes.

(c) Prove that this algorithm ensures mutual exclusion for n processes.

(d) Prove that it is impossible that all processes are waiting in the for-iteration.

(e) Establish whether it is possible that a process that wants to enter the critical section waits ad infinitum.

EXERCISE 2.8. In channel systems, values can be transferred from one process to another process. As this is somewhat limited, we consider in this exercise an extension that allows for the transfer of *expressions*. That is to say, the send and receive statements $c!v$ and $c?x$ (where x and v are of the same type) are generalized into $c!expr$ and $c?x$, where for simplicity it is assumed that *expr* is a correctly typed expression (of the same type as x). Legal expressions are, e.g., $x \wedge (\neg y \vee z)$ for Boolean variables x, y, and z, and channel c with $dom(c) = \{0, 1\}$. For integers x, y, and an integer-channel c, $|2x + (x - y)\text{div}17|$ is a legal expression.

Question: Extend the transition system semantics of channel systems such that expressions are allowed in send statements.
(Hint: Use the function η such that for expression expr, $\eta(expr)$ is the evaluation of expr under the variable valuation η.)

EXERCISE 2.9. Consider the following mutual exclusion algorithm that uses the shared variables y_1 and y_2 (initially both 0).

Process P_1:	Process P_2:
while true **do**	**while** true **do**
... noncritical section ...	... noncritical section ...
$y_1 := y_2 + 1$;	$y_2 := y_1 + 1$;
wait until $(y_2 = 0) \vee (y_1 < y_2)$	**wait until** $(y_1 = 0) \vee (y_2 < y_1)$
... critical section ...	... critical section ...
$y_1 := 0$;	$y_2 := 0$;
od	**od**

Questions:

(a) Give the program graph representations of both processes. (A pictorial representation suffices.)

(b) Give the reachable part of the transition system of $P_1 \parallel P_2$ where $y_1 \leqslant 2$ and $y_2 \leqslant 2$.

(c) Describe an execution that shows that the entire transition system is infinite.

(d) Check whether the algorithm indeed ensures mutual exclusion.

(e) Check whether the algorithm never reaches a state in which both processes are mutually waiting for each other.

(f) Is it possible that a process that wants to enter the critical section has to wait ad infinitum?

Exercise 2.10. Consider the following mutual exclusion algorithm that was proposed 1966 [221] as a simplification of Dijkstra's mutual exclusion algorithm in case there are just two processes:

```
 1 Boolean array b(0;1) integer k, i, j,
 2 comment  This is the program for computer i, which may be
      either 0 or 1, computer j =/= i is the other one, 1 or 0;
 3 C0: b(i) := false;
 4 C1: if k != i then begin
 5 C2: if not b(j) then go to C2;
 6     else k := i; go to C1 end;
 7     else critical section;
 8     b(i) := true;
 9     remainder of program;
10     go to C0;
11     end
```

Here `C0`, `C1`, and `C2` are program labels, and the word "computer" should be interpreted as process.

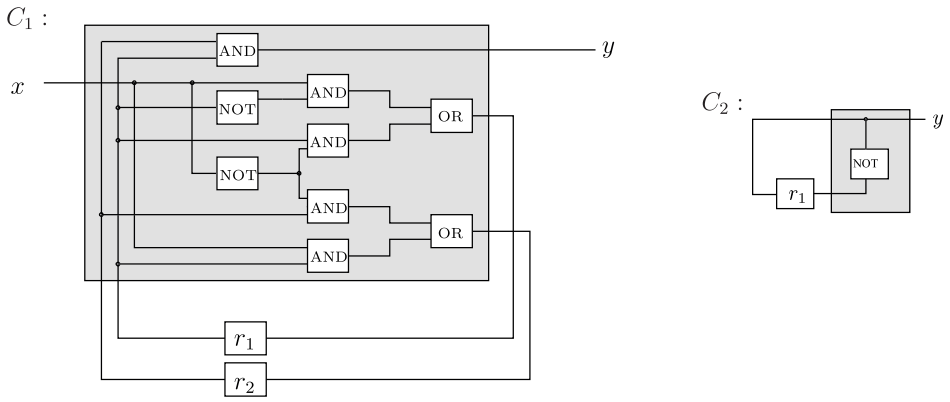
Questions:

(a) Give the program graph representations for a single process. (A pictorial representation suffices.)

(b) Give the reachable part of the transition system of $P_1 \parallel P_2$.

(c) Check whether the algorithm indeed ensures mutual exclusion.

EXERCISE 2.11. Consider the following two sequential hardware circuits C_1 and C_2:



(a) Give the transition system representation $TS(C_1)$ of the circuit C_1.

(b) Let $TS(C_2)$ be the transition system of the circuit C_2. Outline the transition system $TS(C_1) \otimes TS(C_2)$.

EXERCISE 2.12. Consider the following leader election algorithm: For $n \in \mathbb{N}$, n processes $P_1, \ldots, P_n$ are located in a ring topology where each process is connected by an unidirectional channel to its neighbor in a clockwise manner.

To distinguish the processes, each process is assigned a unique identifier $id \in \{1, \ldots, n\}$. The aim is to elect the process with the highest identifier as the leader within the ring. Therefore each process executes the following algorithm:

send (id); *initially set to process' id*
while (true) **do**
  receive (m);
  if ($m = id$) **then stop**; *process is the leader*
  if ($m > id$) **then send** (m); *forward identifier*
od

(a) Model the leader election protocol for n processes as a channel system.

(b) Give an initial execution fragment of $TS([P_1|P_2|P_3])$ such that at least one process has executed the send statement within the body of the whileloop. Assume for $0 < i \leqslant 3$, that process P_i has identifier $id_i = i$.