Model Checking I
Exercises on Models and Modelling with (some) Solutions

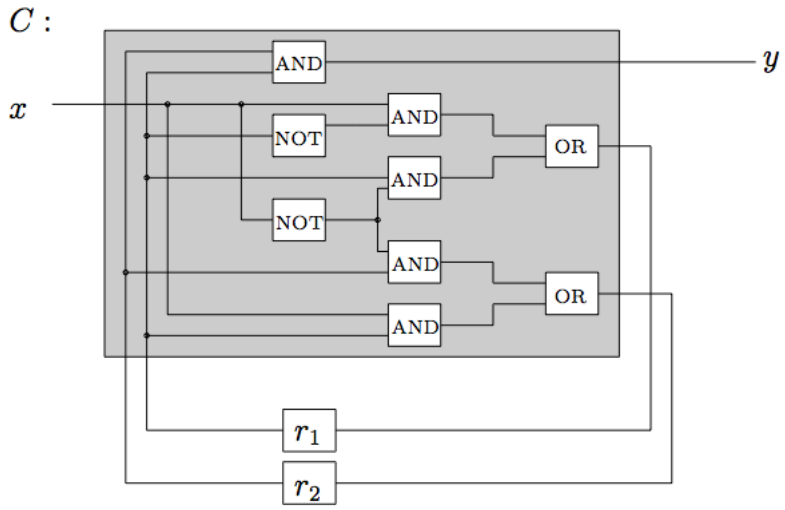
Teacher: Luca Tesei

Master of Science in Computer Science - University of Camerino

Contents

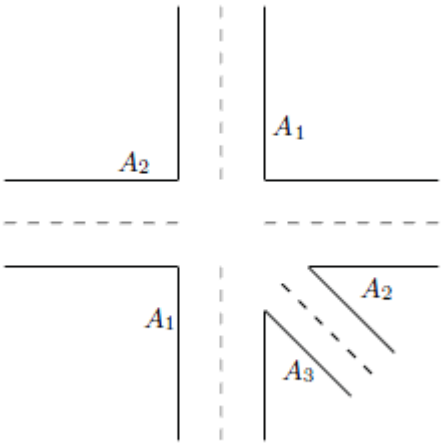
1 Transition systems and Program Graphs

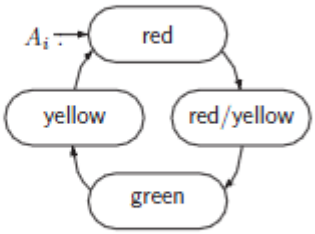
Exercise 1.1. *Consider the following sequential hardware circuit:*



Give the transition system representation of C.

Exercise 1.2. *Consider the following street junction with the specification of a traffic light as outlined on the right.*



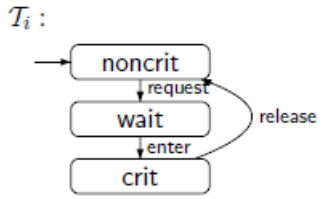


1. *Choose appropriate actions and label the transitions of the traffic light transition system accordingly.*

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

3. *Outline the transition system $A1||A2||A3||C$.*

Exercise 1.3. *A concurrent system comprises P_1,...,P_n competing processes (without shared memory) that access common resource within their critical sections. We assume that the resources may only be accessed exclusively and that £k equivalent instances are available.*
Further, let $n, k \in N$ with $2 \leq k \leq n$.
Process P_i can be described by a transition system $\mathcal{T}_i$ with three states and the actions request, enter and release as indicated below: a) Develop a transition system representation of an arbiter that



communicates with the processes using actions request and release. The arbiter should assure that there are no more than k processes within their critical section at the same time.

b) Sketch the transition system of the parallel composition

$$(\mathcal{T}_1 \;|||\; \mathcal{T}_2 \;|||\; \mathcal{T}_3) \;||_{Syn}\; Arbiter$$

with $Syn = \{request, release\}$ for $k = 2$. You need not consider the states $wait_i$

Exercise 1.4. *Recall the Peterson's algorithm for mutual exclusion (Example 2.25 of the book, page 45). Consider the following variant (for process P_1):*

```
loop forever
  (* non-critical section *)
  x := 2;
  b1 := true;
  wait until ( x = 1 or not b2);
  (*critical section *)
  b1 := false;
  (* non-critical section *)
end loop
```

In particular, note that the assignments `x := 2;` *and* `b1 := true;` *are* **not** *surrounded by the atomic operator* `< >` *and are in reverse order with respect to the original formulation.*

1. *Formalize the behaviour of processes P_1 and P_2 as program graphs.*
2. *Show that there is the possibility that both processes are in the critical section at the same moment. Hint: you don't need to derive the whole transition system of the parallel composition of the two program graphs, but only the part that is needed to show that the wrong state can be reached from the initial state*

Exercise 1.5. *Consider the train crossing example (Example 2.29 of the book, page 50). There, it is possible that a train enters a crossing while the gate is open! Alter this system in the following ways:*

- *A signal is added for the train. The signal can be green or red. The controller changes the signal to green when and only when the track gates are closed. The controller changes the signal to red before opening the gates again.*

- *The train does not enter the crossing when the signal is red.*
- *The controller still does not synchronize with the train on an enter action.*

1. *Give the transition system representation of controller, gates, signal and train (separately).*
2. *Give the transition system representation of the combined system.*
3. *Argue why the train never crosses the road when the train gates are still open.*

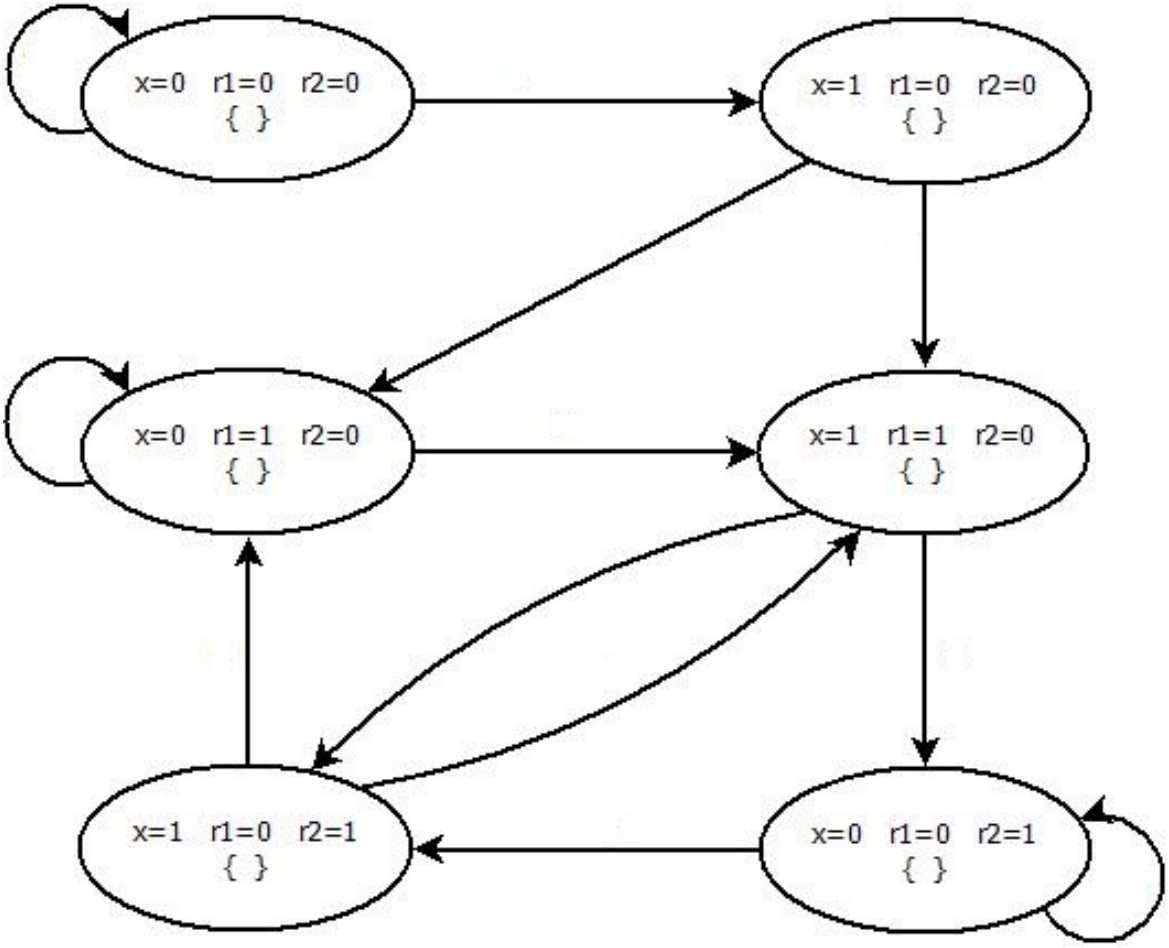
Solutions

Solution of Exercise 1.1

The logical formulas expressing the connections of the wires and the logical gates are the following:

$y = r_1 \wedge r_2$
$r_1 = (x \wedge \neg r_1) \vee (\neg x \wedge r_1)$
$r_2 = (\neg x \wedge r_2) \vee (r_1 \wedge x)$
The set $AP = \{y\}$ is considered.

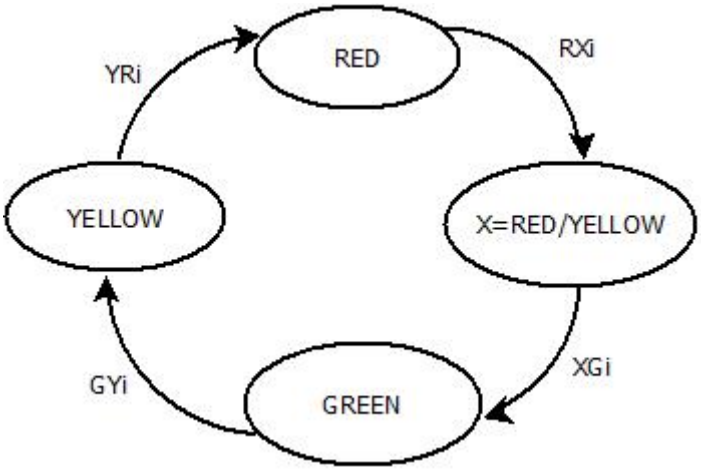
Initially, all registers are set to zero and input ce be zero or one.



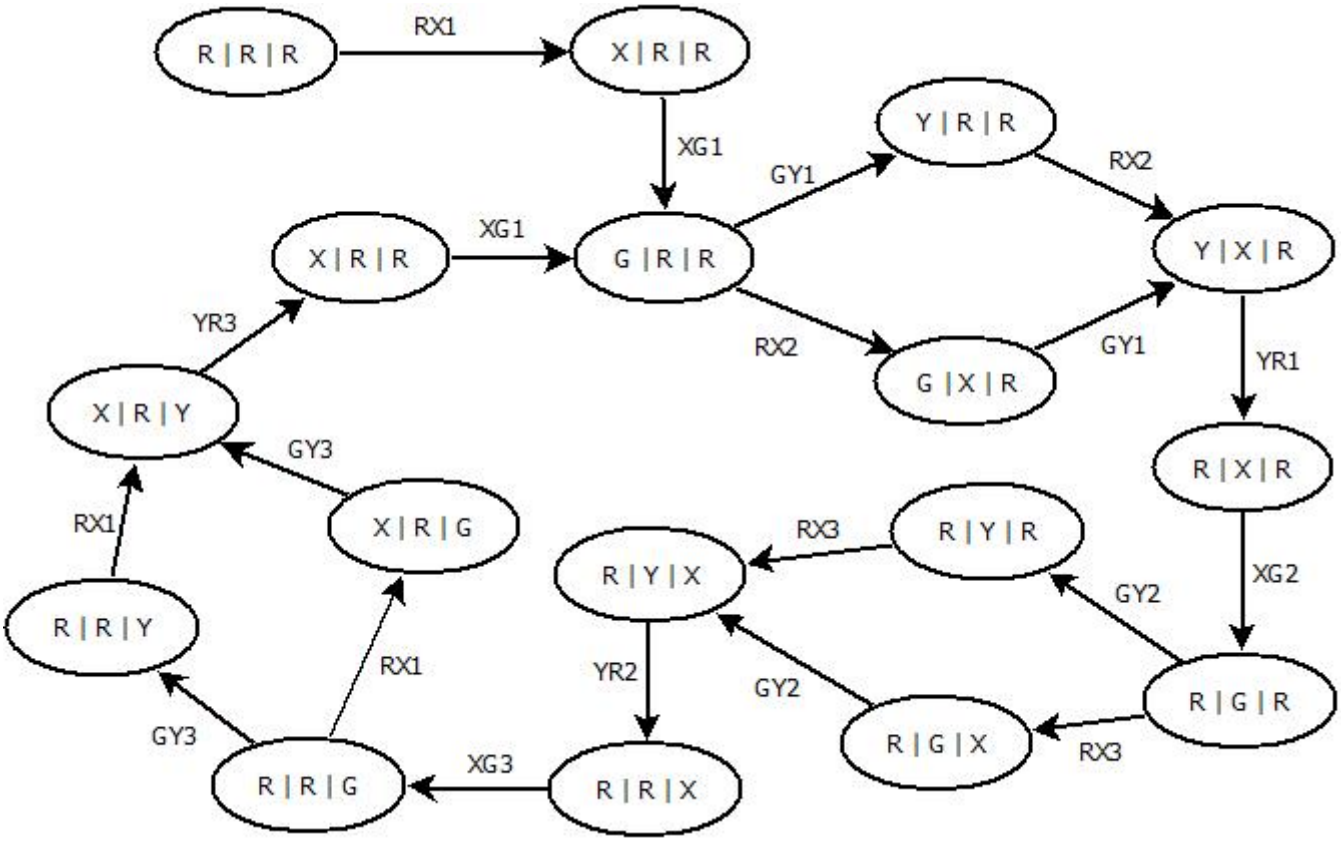
The states $< x = 0, r_1 = 1, r_2 = 1 >$ and $< x = 1, r_1 = 1, r_2 = 1 >$ are not reachable.

Solution of Exercise 1.2

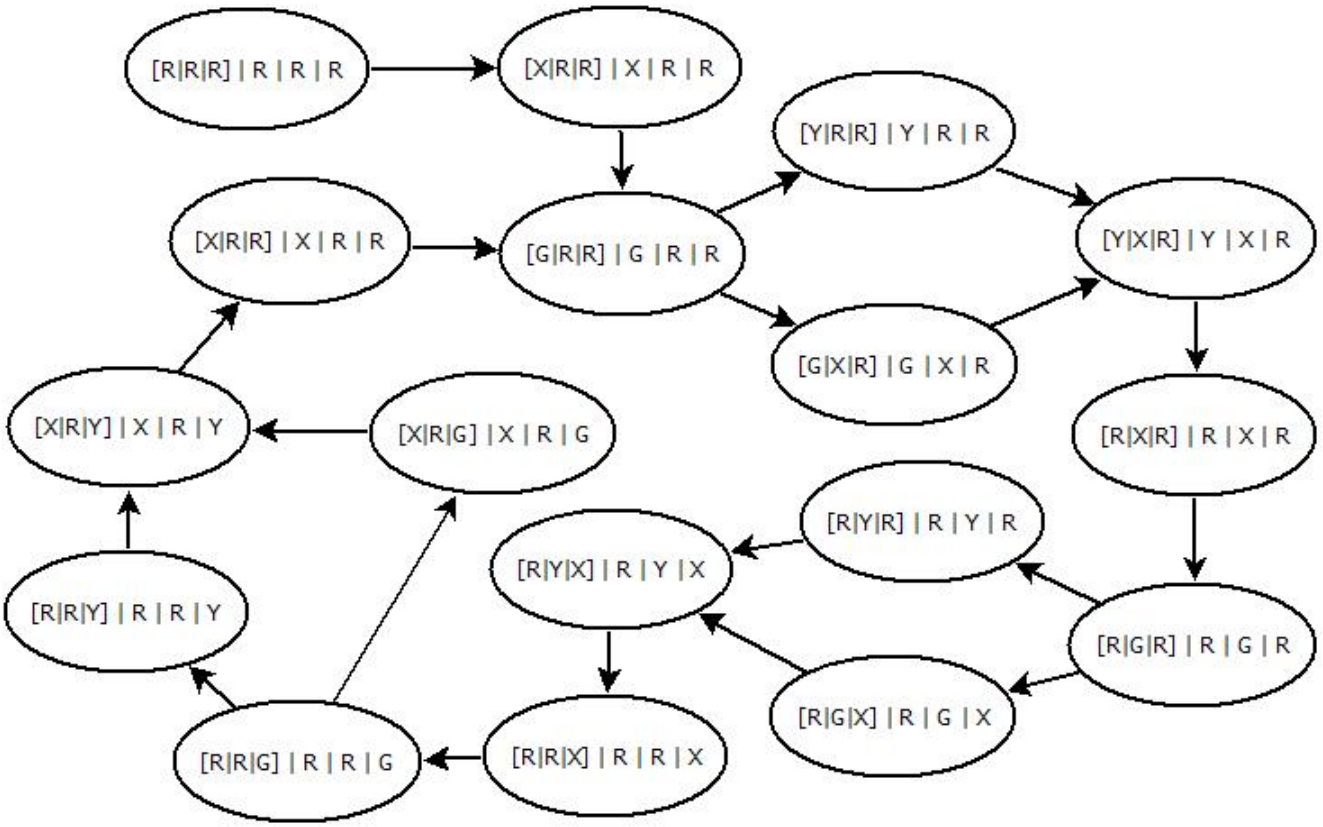
(1)



(2)

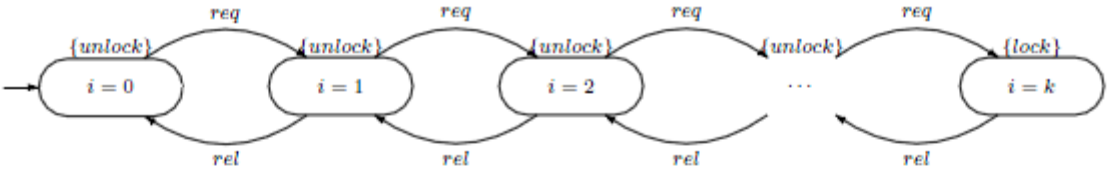


(3)


[R|R|R] | R | R | R
[X|R|R] | X | R | R
[Y|R|R] | Y | R | R
[Y|X|R] | Y | X | R
[X|R|R] | X | R | R
[G|R|R] | G | R | R
[G|X|R] | G | X | R
[X|R|Y] | X | R | Y
[X|R|G] | X | R | G
[R|X|R] | R | X | R
[R|Y|R] | R | Y | R
[R|Y|X] | R | Y | X
[R|R|Y] | R | R | Y
[R|G|R] | R | G | R
[R|G|X] | R | G | X
[R|R|G] | R | R | G
[R|R|X] | R | R | X

Solution of Exercise 1.3

(a) The arbiter that counts the number of resources that are currently used:



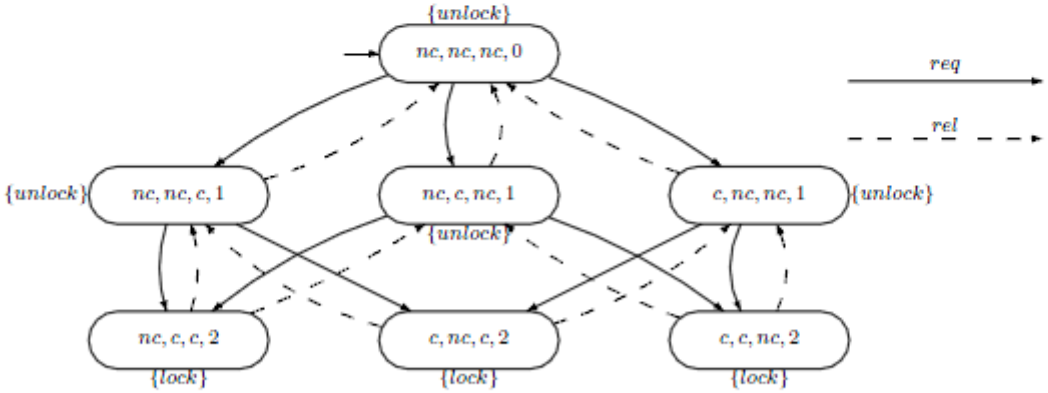
Formally, the arbiter is given by $S = \{0, \ldots, k\}, \mathsf{Act} = \{req, rel\}, AP := \{unlock, lock\}$ and

$$L(i) := \begin{cases} \{lock\} & \text{if } i = k \\ \{unlock\} & \text{otherwise} \end{cases}$$

The transition relation is given by

$$i \xrightarrow{req} i+1 \quad \text{for } 0 \leq i < k$$
$$i \xrightarrow{rel} i-1 \quad \text{for } 0 < i \leq k$$

(b) In the solution, we only allow two processes within the critical region and neglect the states $wait_i$:

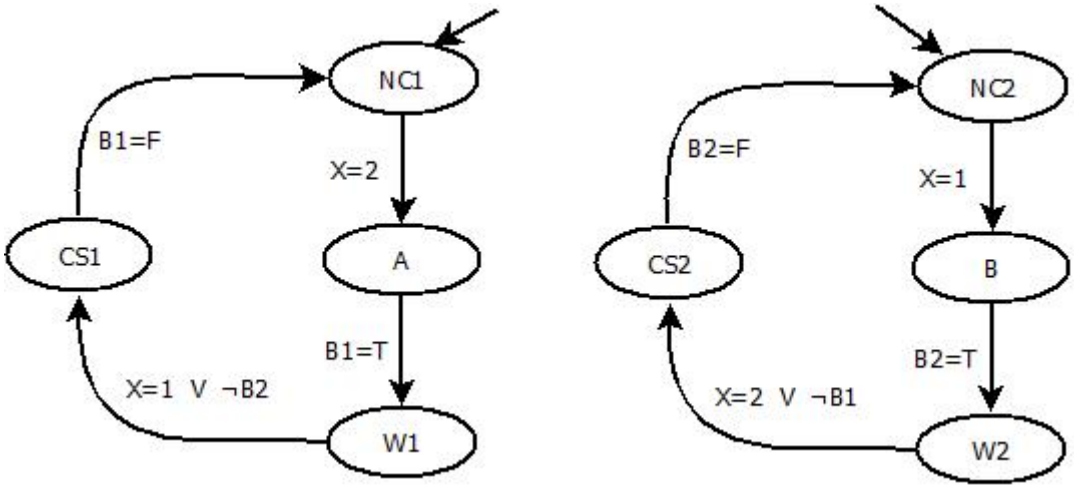


$S = \{\langle s_1, s_2, s_3, i\rangle \mid s_i \in \{noncrit, crit\}, i \in \{0,1,2\}\}$
$\mathsf{Act} = \{req, rel\}, AP = \{lock, unlock\}$

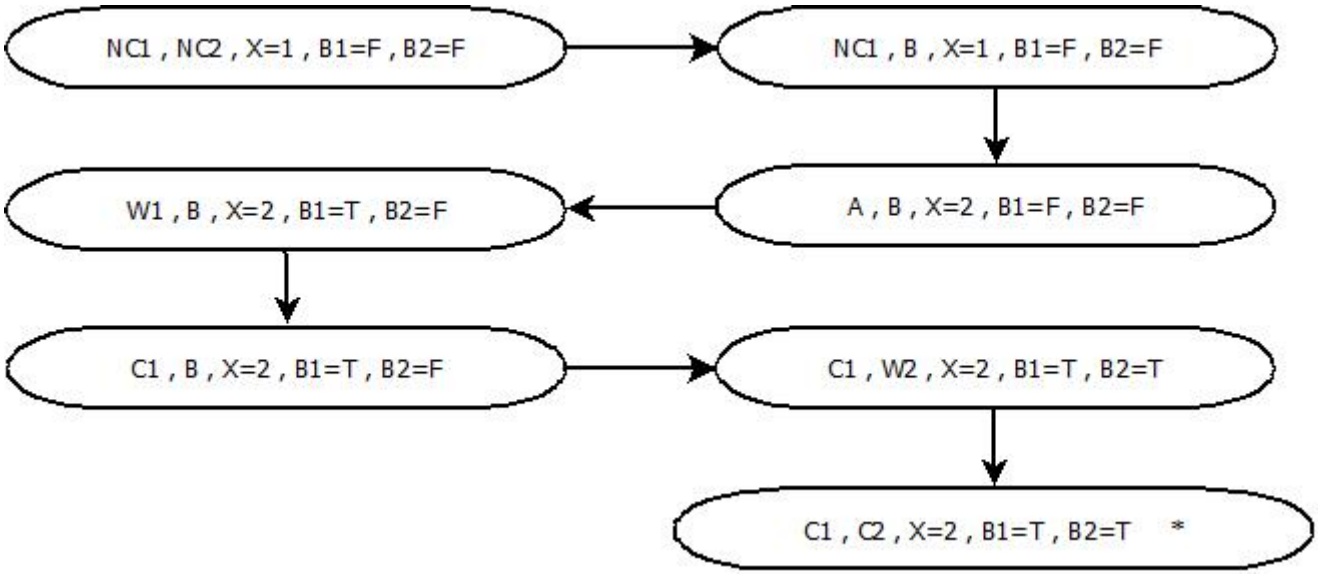
$$L(\langle s_1, s_2, s_3, i\rangle) = \begin{cases} \{lock\} & \text{if } i = 2 \\ \{unlock\} & \text{otherwise} \end{cases}$$

Solution of Exercise 1.4

We consider the set $var = \{x, b_1, b_2\}$ where $dom(x) = \{1, 2\}$, $dom(b_1) = dom(b_1) = \{true, false\}$.
The initial condition is $b_1 = false$ and $b_2 = false$.
The program graphs for preocess P_1 and P_2 are the following:



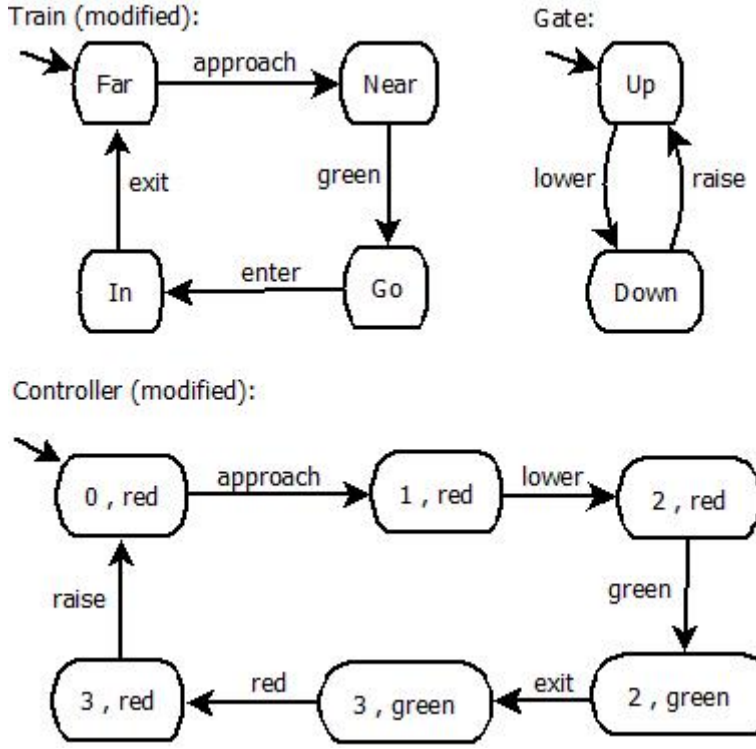
The following is a portion of the transition system derived from the program graph $PG_1|||PG_3$ that share the variables var.



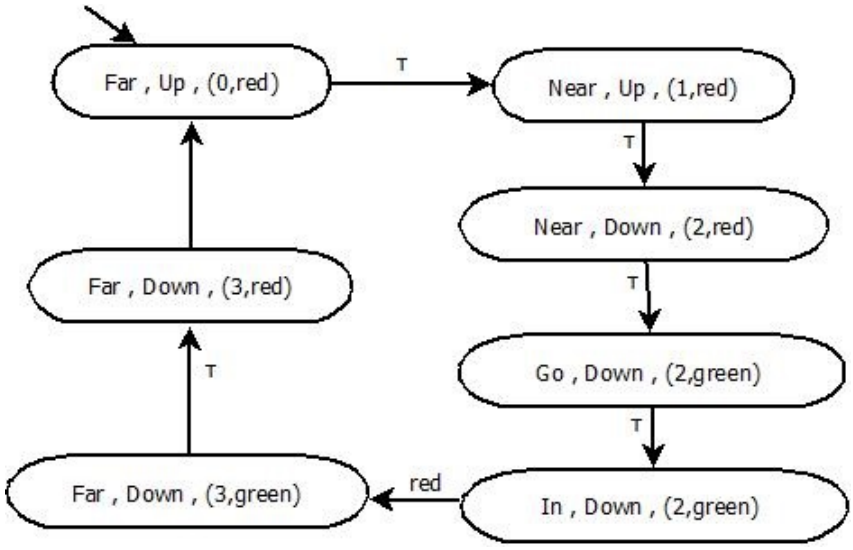
The paths from the initial state to the state " $*$ " shows that a state in which both P_1 and P_2 are in the critical section at the same time is reachable.

Solution of Exercise 1.5

The new signal is controlled by the controller that adds a synchronization with the train (green) before the train could enter.



The gate is unchanged. Note that the only two non-synchronizing actions are enter and red. The transition system $Train||Gate||Controller$ is the following one.



There is not any reachable state in which the train crosses while the gates are open.

2 Channel Systems and nano Promela

Exercise 2.1. *Consider the following generalization of Peterson's mutual exclusion algorithm that is aimed at an arbitrary number $n(n \geq 2)$ 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:

- *$y[0..n-1]$ with $y[k] \in \{1, ..., n\}$ such that $y[j] = i$ means that process i has the lowest priority at level j.*
- *$p[1..n]$ with $p[i] \in \{0, ..., n-1\}$ such that $p[i] = j$ expresses that process i is currently at level j.*

Each process starts at level 0; before entering the critical secrion, it has to pass through levels 1 to $n-1$. 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 behaviour of process i is given by the following algorithm:

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

Questions:

1. *Formally define the program graph for process i.*
2. *Determine the number of states (including the unreachable states) in the parallel composition of n processes.*
3. *Prove that this algorithm ensures mutual exclusion for n processes.*

Exercise 2.2. *Consider the following leader election algorithm: For $n \in N$, n processes $P1, ..., Pn$ 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 $id \in 1, ..., n$. The aimis to elect the process with the highest identifier as the leader within the ring. Therefore each process executes the following algorithm:

```
send (id);
while (true) do
    Receive (m);
    if (m=id) then
        stop;
    end
    if (m>id) then
        send(m);
    end
end
```

1. *Model the leader election protocol for n processes as a channel system.*
2. *Give an initial execution fragment of* $TS([P1|P2|P3])$ *such that at least one process has executed the send statement within the body of the while loop. Assume for* $0 < i \leq 3$*, that process* Pi *has identifier* $id_i = i$*.*

Exercise 2.3. *Consider a system consisting of n processes* $P0, ..., Pn-1$ *and a central moderator* M *in a fully connected network.*
Each process $Pi(for 0 \leq i < n)$ *executes the same algorithm and stores a unique identifier* $id_i \in N$*.*
Further, we assume that n is known a priori.
In order to elect a leader, the system is supposed to determine the process with the highest id and communicate it to every process.

- *Informally describe how to solve the leader election problem in the above setting.*
- *Write nanoPromela models for the algorithm of the process and the moderator. Add comments!*
- *Formally derive the program graphs for a process and the moderator.*

Exercise 2.4. *Consider a system composed of the following components:* `Machine`*,* `Executor` *and* `Queue`*.*

- *The* `Machine` *can be in* `running` *or in* `standby` *mode.*
- *The* `Executor` *can be in* `idling` *or in* `executing` *mode.*
- *Whenever the* `Machine` *is running and the* `Executor` *is idling, then the latter can accept tasks from the* `Queue`*, to be executed.*
- *After executing a task, the* `Executor` *can terminate it and go back to the idling state.*
- *The* `Queue` *collects, from an external source, tasks to be executed in a buffer of maximum length 2. The source of the tasks should not be modeled, but it must be assumed that it can produce infinitely many tasks.*
- *Whenever the* `Machine` *is running, the* `Queue` *is empty and the* `Executor` *is idling, then the* `Machine` *can switch to the standby mode to save energy.*
- *Whenever the* `Machine` *is in standby, the* `Queue` *is not empty and the* `Executor` *is idling, then the* `Machine` *can switch to the running mode in order to start processing tasks again.*
- *Initially, the* `Machine` *is in standby, the* `Queue` *is empty and the* `Executor` *is idling.*

1. *Model the scenario sketched above as a Channel System. Define clearly all channels with their type and capacity and define clearly all the shared variables with their type and initial values.*

2. *Draw the Transition System resulting from the defined Channel System by the standard semantics.*

3. *Argue, justifying your answer, that your model is deadlock-free.*

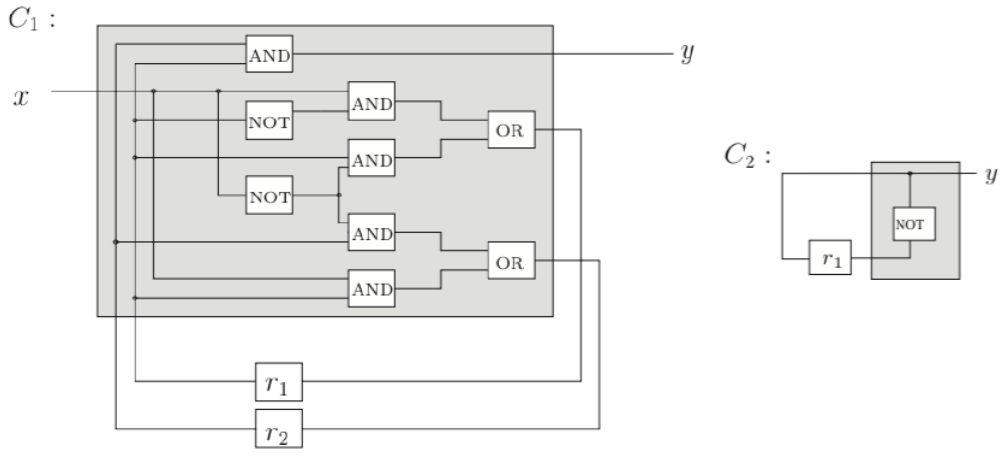
Exercise 2.5. *Consider the Hyman's mutual exclusion algorithm for two processes P_1 and P_2. It uses a shared integer variable $k \in \{1,2\}$ and 2 boolean variables b_i, $i \in \{1,2\}$. Each process P_i executes the algorithm below, where i is the index of the process and j is used as the index of the other process:*

```
while (true) do
begin
  'noncritical section';
  b_i := true;
  while (k != j ) do
  begin
     while (b_j) do skip;
     k := i;
  end
  'critical section';
  b_i := false;
end
```

1. *Model the behavior of the generic process P_i in nanoPromela. Define clearly all channels with their type and capacity and define clearly all the shared variables with their type and initial values.*

2. *Specify if you want to use the test-and-set semantics or the two-steps one. Argue whether or not this choice could have any effect on the correctness of your implementation of the algorithm.*

Exercise 2.6. *Consider the following two sequential hardware circuits C_1 and C_2:*



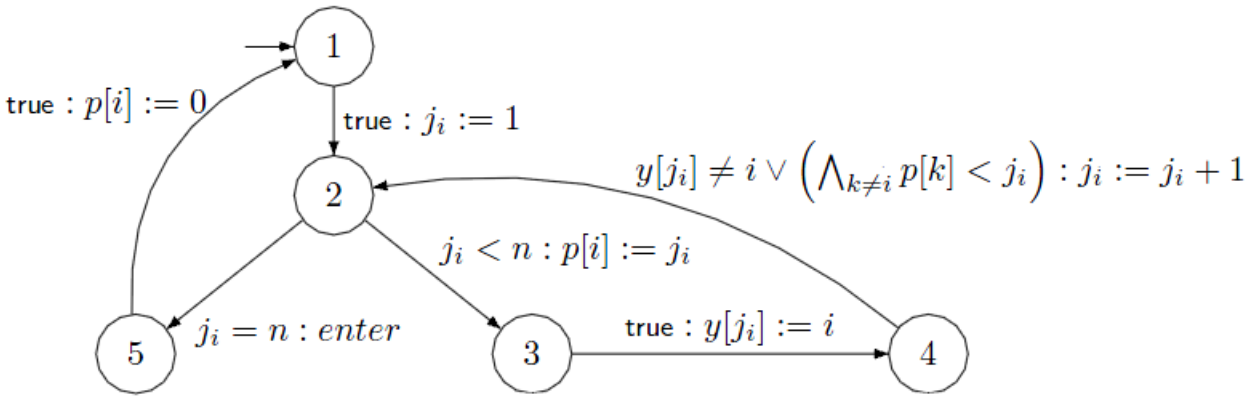
1. *Give the transition system representation $TS(C_1)$ of the circuit C_1.*

2. *Let $TS(C_2)$ be the transition system of the circuit C_2. Draw the transition system $TS(C_1) \otimes TS(C_2)$, i.e., the synchronous product between $TS(C_1)$ and $TS(C_2)$.*

Solutions

Solution of Exercise 2.1

1. The program graph PG_i of process i is given as:



   Note that we consider i as a constant here and that the variables j_i are not shared.

2. The cardinality of the set of states of $TS(PG_1|||...PG_n)$ can be deduced as follows:
   Let $PG = (Loc, Act, Effect, \rightarrow, loc_0, g_0)$ be the formal representation of the program from part (1) where:

   - $Loc = \{1, 2, 3, 4, 5\}$
   - $Act = \{j_i := 1, p[i] := j_i, y[j_i] := i, j_i := j_i + 1, enter, p[i] := 0 | i \in \{1, ..., n\}\}$

   According to the algorithm, we have:

$$dom(y[k]) = \{1, ..., n\} \text{ for all } k \in \{0, ..., n-1\}$$
$$dom(ji) = dom(p[k]) = \{0, ..., n-1\} \text{ for all } k, i \in \{1, ..., n\}$$

   Therefore it follows $|dom(y[k])| = |dom(ji)| = |dom(p[k])| = n$.
   The arrays y and p have capacity n:
   The state space of the transition system is

$$S = Loc1 \times ... \times Locn \times Eval(\{p[k], y[l], ji | i, k \in \{1, ..., n\} and l \in \{0, ..., n-1\}\}).$$

   Therefore we get $|S| = 5^n \times n^{3n}$

3. We prove a stronger statement that implies mutual exclusion:
   For level $j \in \{0, ..., n-1\}$, there are at most $n - j$ processes on levels $\geq j$. By definition, a process P_i is in level j iff $p[i] = j$. We proceed by induction over j:

   - basis $(j = 0)$: The statement holds, as $n - j = n - 0 = n$ and there are no more than n processes in the system.

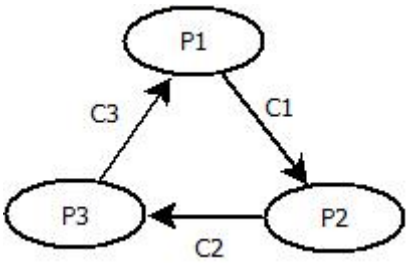
- induction step ($j \rightsquigarrow j+1$): The induction hypothesis implies that there are at most $n-j$ processes on levels $\geq j$. We show that there is at least one process that cannot move from level j to level $j+1$. By contradiction, assume there also were $n-j$ processes on levels $\geq j+1$ (i.e. no process is stuck at level j).
  Let i be the last process that writes to $y[j]$. Therefore the old value of $y[j]$ that corresponds to the previous process k at level j is overwritten and we have $y[j] = i$. Hence the condition $y[j] \neq i$ cannot be true.
  According to the algorithm,
  - process k writes $p[k]$ before it writes $y[j]$
  - process i reads $p[k]$ only after it wrote to $y[j]$.

  Therefore every time process i reads $p[k]$, process k already set $p[k] = j$ and for process i, the second condition $p[k] < j$ is not fulfilled either.
  We assumed that process i enters level $j+1$. This yields a contradiction since it cannot leave the wait–loop.

According to the idea of the algorithm, a process enters the critical section when it leaves the wait–loop in level $n-1$. As we proved, in level $n-1$, there may only be $n-(n-1) = 1$ processes in levels $\geq (n-1)$. Therefore we have the desired mutual exclusion property.

Solution of Exercise 2.2

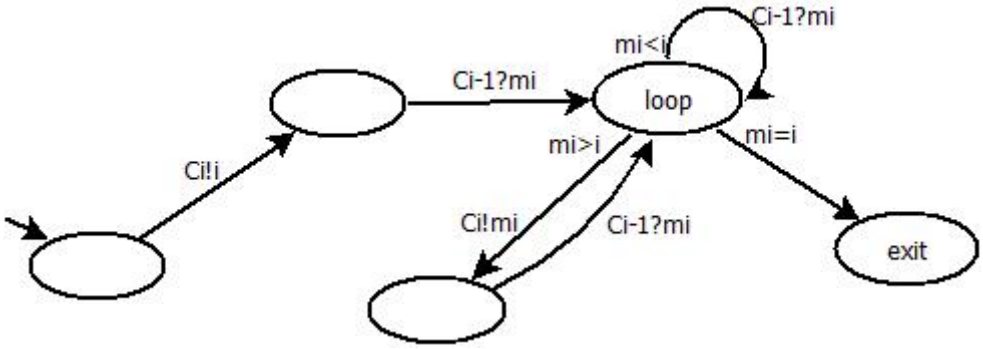
1. Channel system of process i:
   $P_i \xrightarrow{c_i} P_{i+1}$
   $Cap(ci) = 0$



   Behaviour of process i:

$c_i!i$
$c_{i-1}?mi$
do
$:: m_i > i \Longrightarrow c_i!m_i; c_{i-1}?m_i$
$:: m_i < i \Longrightarrow c_{i-1}?m_i$
od

2. .



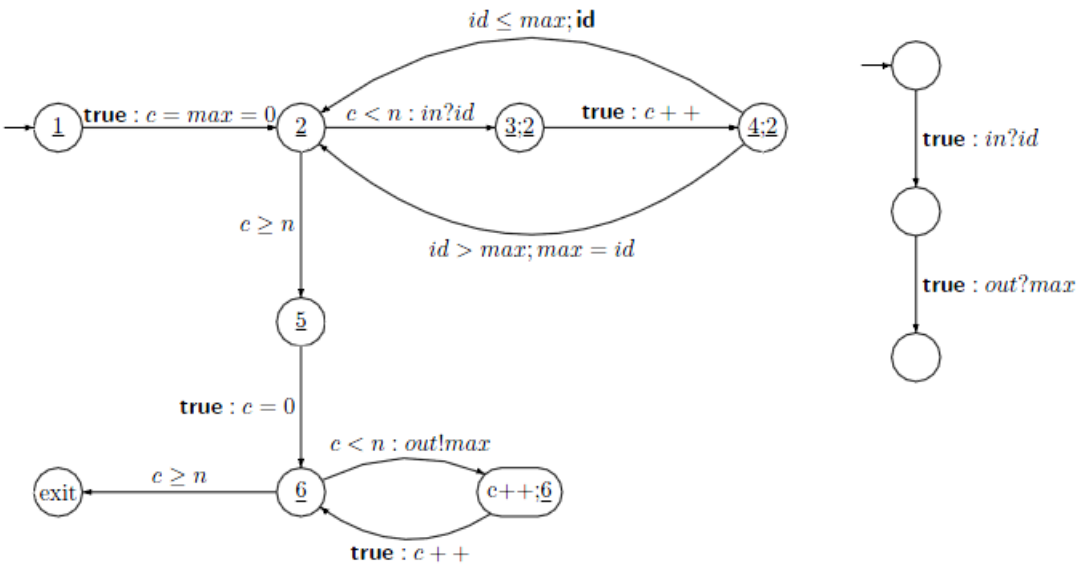
Solution of Exercise 2.3

- The protocol is trivial: Each process informs the moderator of its unique identifier. In a second round, the moderator sends messages to all processes that include the highest process identifier received in the first round.
- In nanoPromela, this can be formalized as follows:

```
# model of a process
in!id;
out?max;

# moderator
atomic { c = 0; max = 0; }
do
:: c < n => in?id;
            c++;
            if
            :: id > max => max = id;
            :: else     => skip;
            fi;
od;
c = 0;
do
:: c < n => out!max;
            c++;
od;
```

- The program graph of the moderator M and process P_i are:



The transitions above are defined by the inference rules for nanoPromela.
For example:

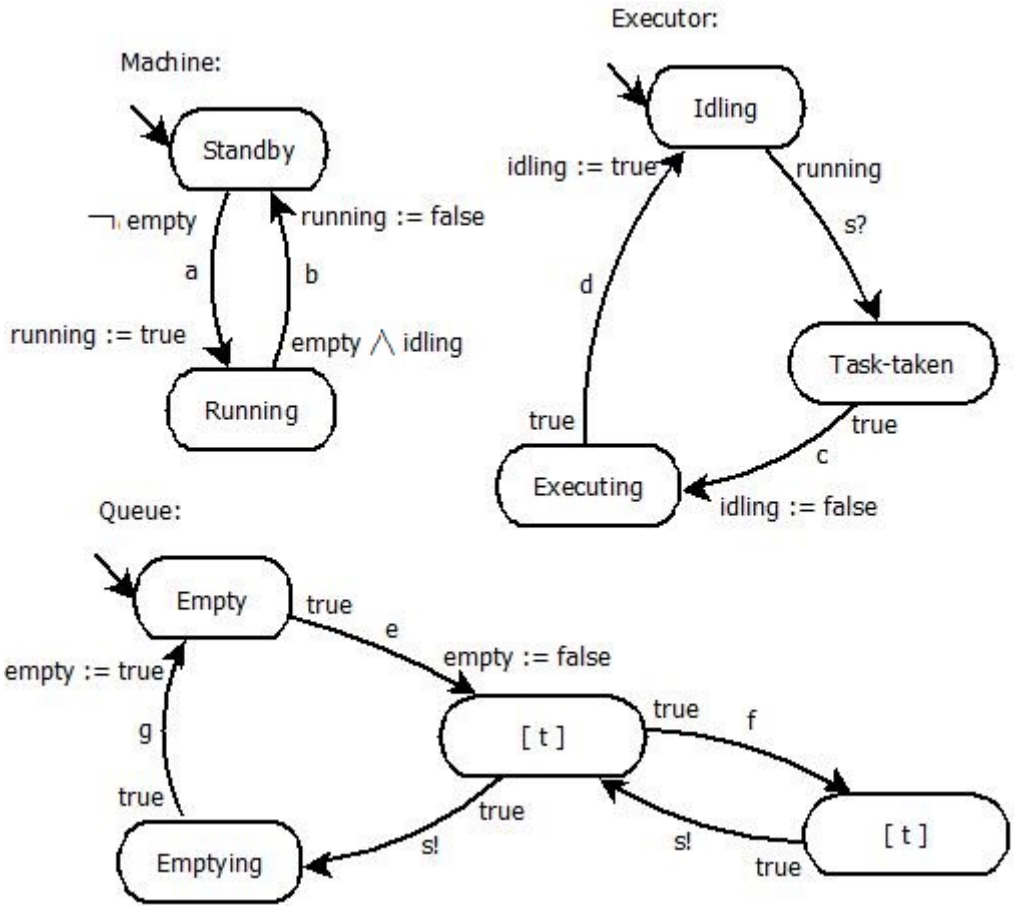
$$\text{loop}\ \frac{\text{seq. comp.}\ \dfrac{\text{recv}\ \dfrac{-}{in?id \xrightarrow{\textbf{true}:in?id} exit}}{in?id;\underline{3} \xrightarrow{\textbf{true}:\text{id?id}} \underline{3}}}{\underline{2} \xrightarrow{c<n:in?id} \underline{3};\underline{2}}$$

Solution of Exercise 2.4

1. let $Var = \{running, idling, empty\}$ where $dom(x) = \{true, false\}\ \forall\ x \in Var$
   let $Chan = \{s\}$ with $cap(s) = 0$, handshaking channel.
   The initial condition
   $g_0 \equiv running = false \wedge idling = true \wedge empty = true$
   $\equiv \neg running \wedge idling \wedge empty$

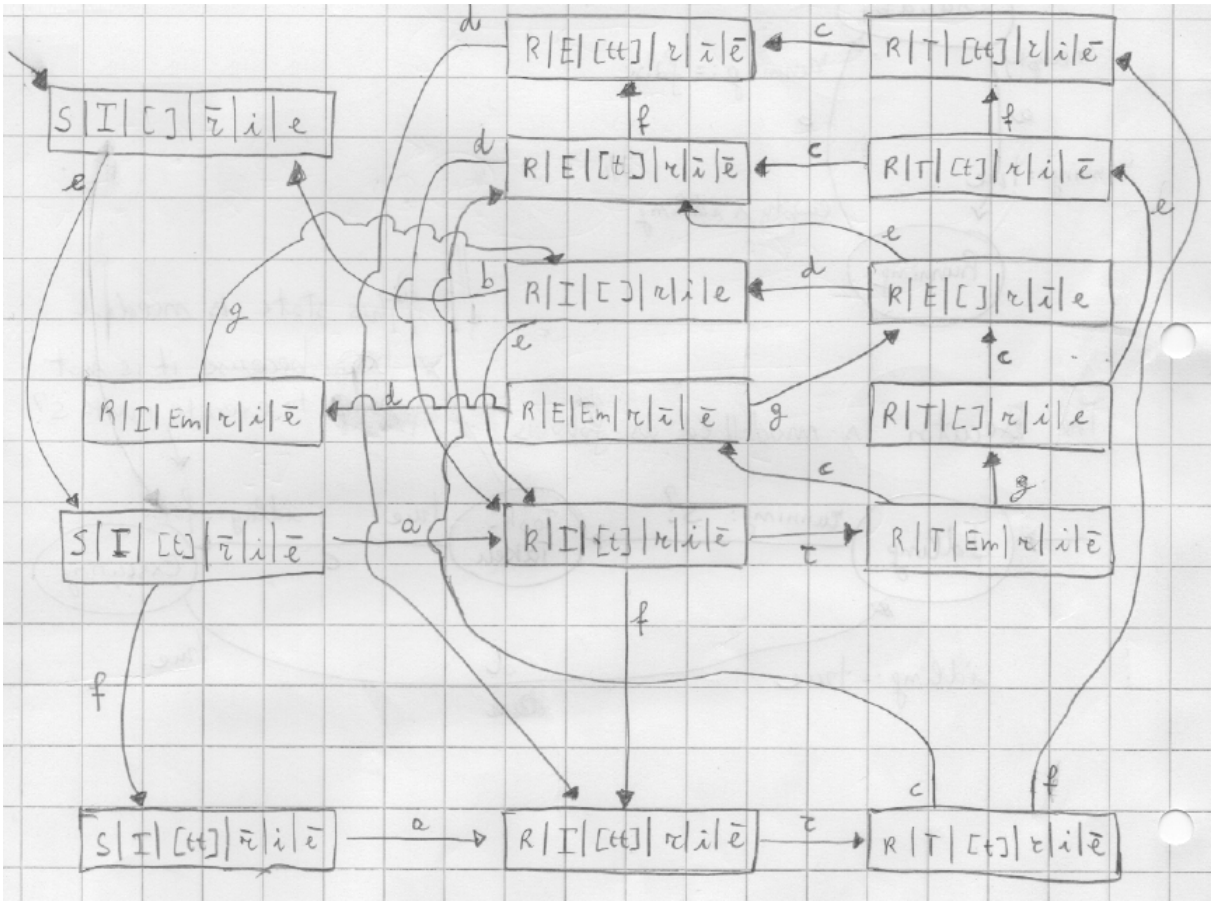
   Machine, Executor and Queue are modelled as follows:

   (Executor: "Task-taken" state is needed because it is not possible to execute with "$s?$"
   Queue: "Emptying" state is needed because in Channel Systems the execution of a communication is disjoint from the updating of variables)



2. The transition system resulting from the Channel System is the following:
   where

   - S = Standby
   - R = Running
   - I = Idling
   - E = Executing
   - [ ] = Empty
   - E = Emptying
   - T = Task-Taken
   - i = idling
   - r = running
   - e = empty
   - $\overline{i}$ = idling
   - $\overline{r}$ = running
   - $\overline{e}$ = empty



3. All the states in the generated TS have at least one outgoing transition, then there cannot be any deadlock.

Solution of Exercise 2.5

1. There are no channels. The variables are k, b_1, b_2 with $dom(x) = \{1, 2\}$, $dom(b_i) = \{true, false\}$ for all $i \in \{1, 2\}$.

   The nano Promela code corresponding to the given algorithm is the following:

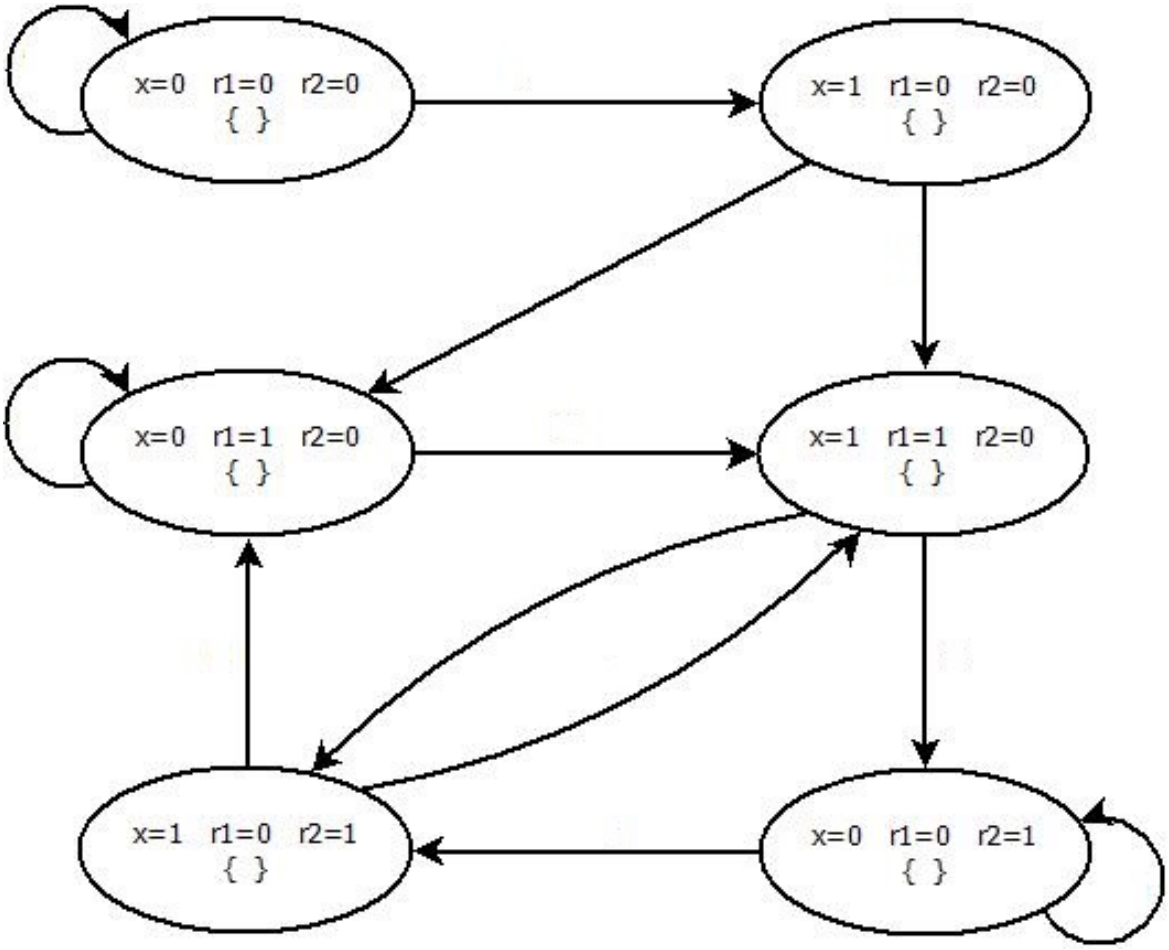
```
P1 :
DO :: true ⇒ skip; // non-critical section
              b1 := true;
              DO :: k != 2 ⇒ DO :: b2 ⇒ skip;
                                OD
                              k := 1;
              OD
              skip; // critical section
              b1 := false;
OD
```

The code for P_2 is symmetric, it can be obtained exchanging 1 with 2 and 2 with 1 in the code above.

2. In general it is more convenient to use $test - and - set$ semantics because it guarantees more atomicity between the evaluation of guards and the execution of the relative guarded commands. However, in this particular case, we can see that the atomicity between the guards $k! = 2$ and b_2 with respect to the first basic command executable $skip$; is not influent at all in the behaviour of the algorithm.
   Thus, in this particular case, the choice between $test - and - set$ or $two - steps$ semantic is irrelevant.

Solution of Exercise 2.6

1. $TS(C_1)$



$TS(C_2)$

2. $TS(C_2) \otimes TS(C_2)$