Exercises on Transition Systems

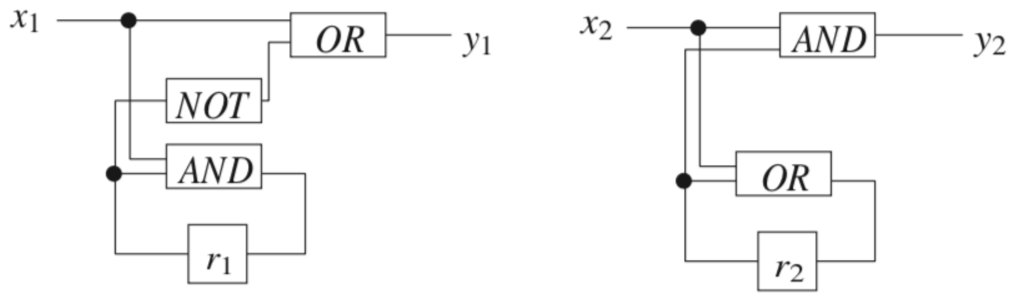
Beau De Clercq

October 2019

Ex. 2.1

Question

Consider the following two sequential hardware circuits:

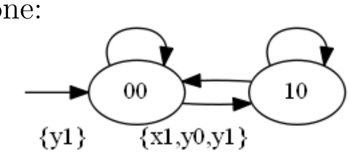


(a) Give the transition systems of both hardware circuits.
(b) Determine the reachable part of the transition system of the synchronous product of these transition systems. Assume that the initial values of the registers are r1=0 and r2=1.

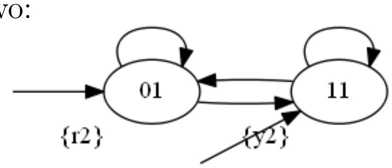
Answer

(a)

- Circuit one:

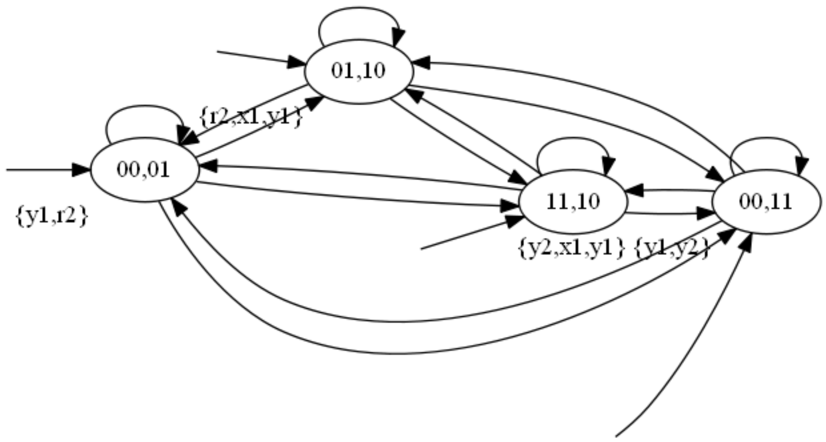


- Circuit two:



(b)

- Reachable part:



- Not reachable: (00,00), (11,11), ...

Ex. 2.2

Question

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, ..., 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;
P1 || P2 || P3
```

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

Answer

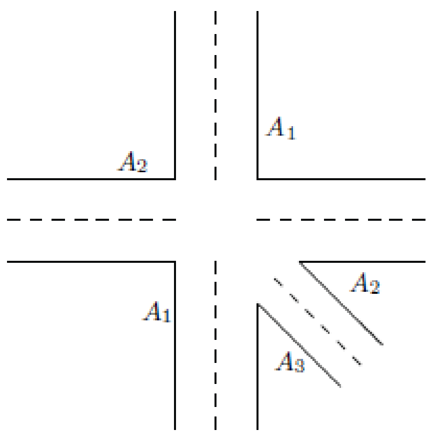
The answer to this question depends on how you interpret the question.

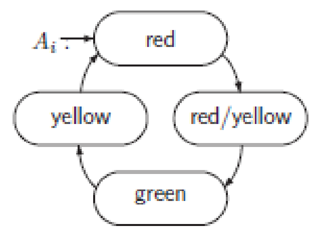
- If you allow processess to run completely independent from each other, it is possible to have an execution that halts with x = 2:
  - First load x := 0 in P_1, P_2, P_3
  - Then let P_1 finish (x == 10)
  - After that, let P_2 run 9 times and store x = 9
  - Increment P_3 1 time and store
  - Load x := 1 stored by P_3 in P_2
  - Load x := 1 in P_3
  - Finish P_3 before P_2 to end with x == 2
- If you don't allow this and expect processes to wait until all others have finished their loop then it isn't possible.

Ex. 2.3

Question

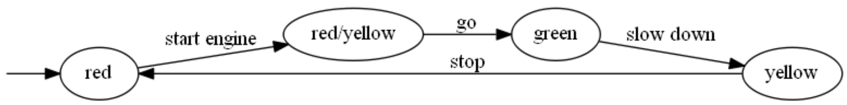
Consider the following street junction with the specification of a traffic light as outlined on the right.

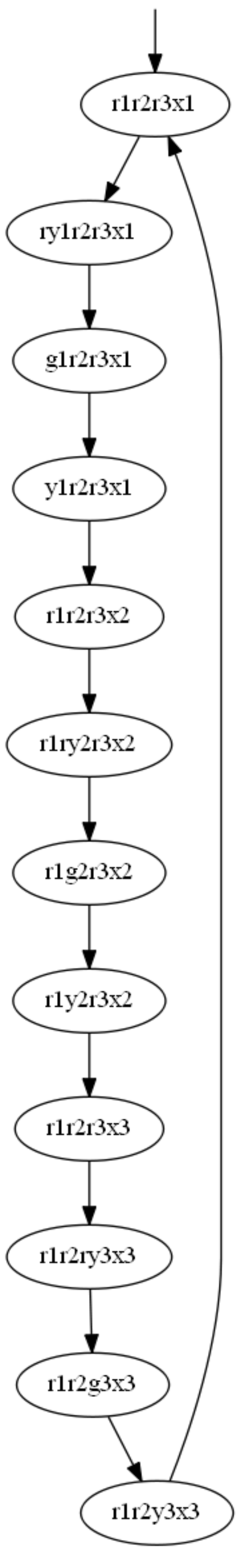




(a) Choose appropriate actions and label the transitions of the traffic light transition system accordingly.
(b) Give the transition system representation of a (reasonable) controller C that switches the green signal lamps in the following order: A1,A2,A3, A1,A2,A3, ...
(Hint: Choose an appropriate communication mechanism.)
(c) Outline the transition system $A1 \| A2 \| A3 \| C$.

Answer

- 


r1r2r3x1
ry1r2r3x1
g1r2r3x1
y1r2r3x1
r1r2r3x2
r1ry2r3x2
r1g2r3x2
r1y2r3x2
r1r2r3x3
r1r2ry3x3
r1r2g3x3
r1r2y3x3

•

- /

Ex. 2.4

Question

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 $(TS_1 \| TS_2) \| TS_3 = TS_1 \| (TS_2 \| TS_3)$ where TS_1, TS_2, TS_3 are arbitrary transition systems.

Answer

$$\begin{aligned}
(TS_1 \| TS_2) \| TS_3 &= (S_1 \times S_2, Act_1 \times Act_2, -\,>, I_1 \times I_2, AP_1 \cup AP_2, L) \| TS_3 \\
&= (S_1 \times S_2 \times S_3, Act_1 \times Act_2 \times Act_3, -\,>, I_1 \times I_2 \times I_3, AP_1 \cup AP_2 \cup AP_3, L) \\
&= TS_1 \| (S_2 \times S_3, Act_2 \times Act_3, -\,>, I_2 \times I_3, AP_2 \cup AP_3, L) \\
&= TS_1 \| (TS_2 \| TS_3)
\end{aligned}$$

Ex. 2.5

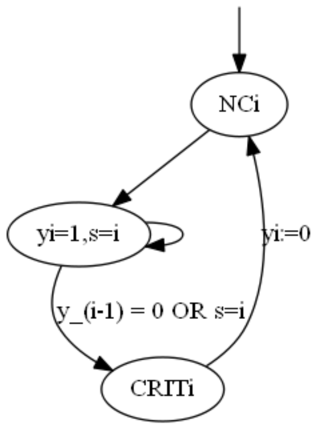
Question

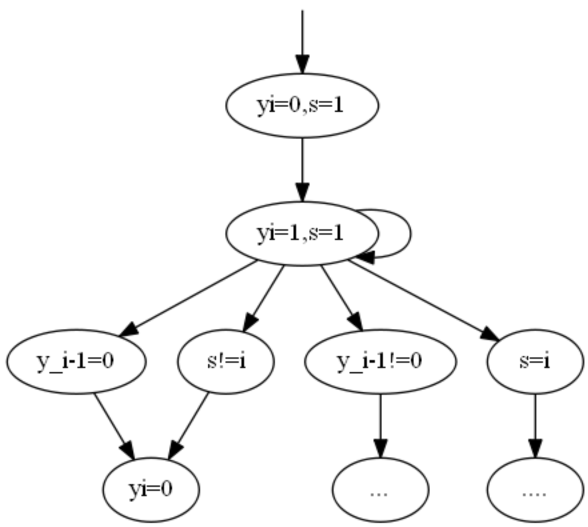
The following program is a mutual exclusion protocol for two processes due to Pnueli [118]. There is a single shared variable s which is either 0 or 1, and initially 1. Besides, each process has a local Boolean variable y that initially equals 0. The program text for process P_i (i = 0, 1) is as follows:

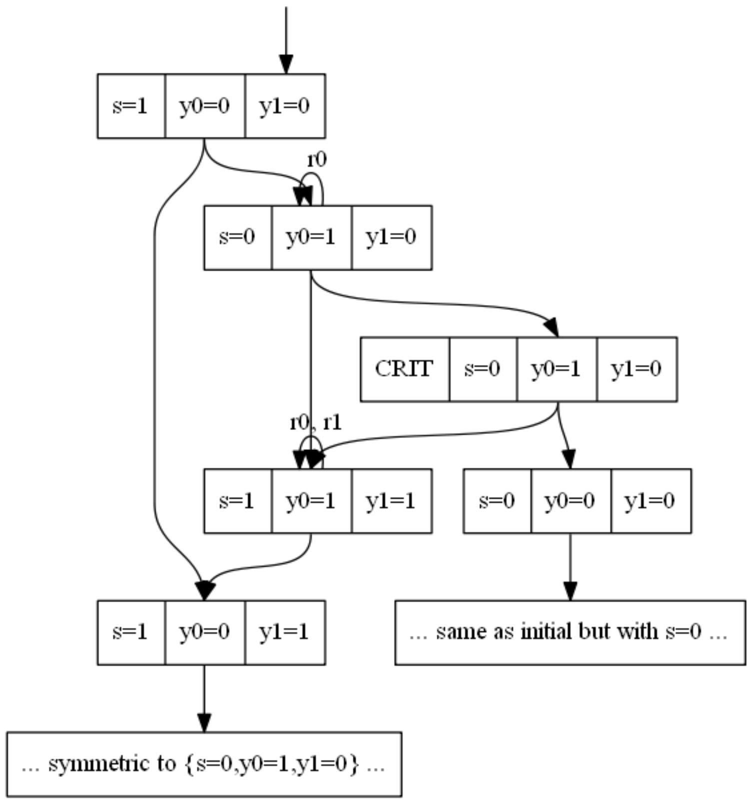
l0: **loop forever do**
  begin
  l1: Noncritical section
  l2: $(y_i, s) := (1, i)$;
  l3: **wait until** $((y_{1-i} = 0) \ \vee \ (s \neq i))$;
  l4: Critical section
  l5: $y_i := 0$
end.

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.
(a) Define the program graph of a process in Pnuelis 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.

Answer

- 

- 

- 

- P = $\{CRIT_1, CRIT_2, r_1, r_2\}$
  ME is ensured: it cannot happen that 2 processes execute the critical section at the same time due to the **wait** condition.

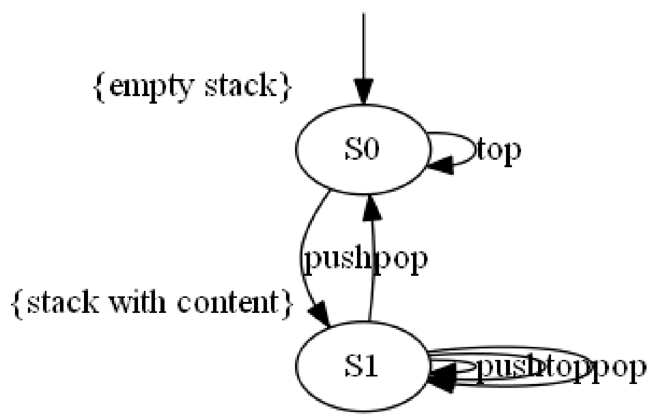
- Starvation freedom is not ensured: it may happen that a process is always surpassed by one or more other processes.

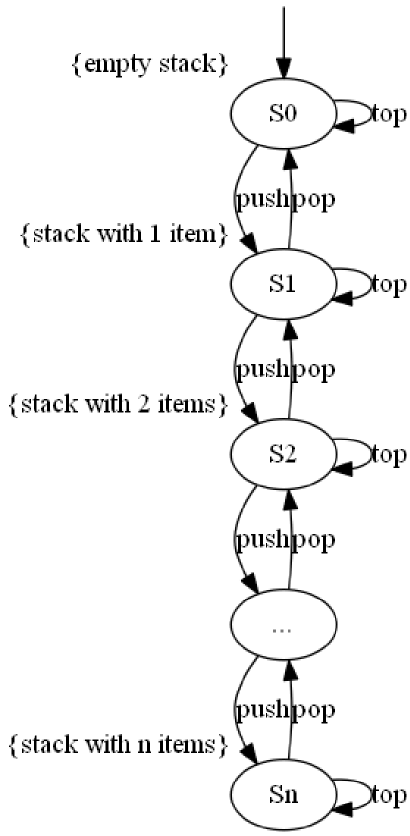
Ex. 2.6

Question

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.

Answer

- 

- 
{empty stack}
S0
top
push
pop
{stack with 1 item}
S1
top
push
pop
{stack with 2 items}
S2
top
push
pop
...
push
pop
{stack with n items}
Sn
top

Ex. 2.7

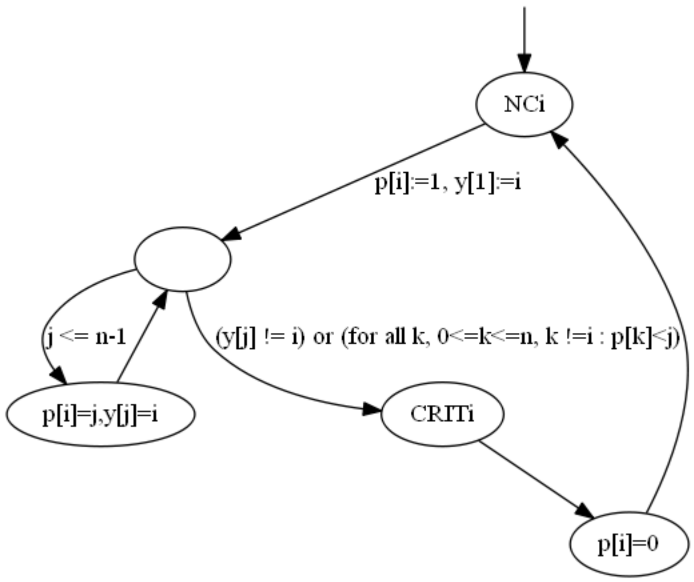
Question

Consider the following generalization of Petersons 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..n1] and p[1..n] with y[i] 1, . . . , n and p[i] 0, . . . , n1 .
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 n1. 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

(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.

Answer



-
- Each processes has 5 states.
  Combining 2 processes gives $5^2 = 25$ (each state can be combined with 5 other states) possible states, combining 3 processes gives 5^3 possible states, ... $\Rightarrow$ combining n processes gives 5^n possible states.
- /
- /
- Either a process will enter the critical section because it doesn't have the lowest priority ($y[j] \neq i$) OR because all other processes exited the critical section. Assuming that every process will leave the critical section after a reasonable amount of time, a process will never have to wait ad infinitum to enter the critical section.

Ex. 2.8

Question

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, $|2x + (x \;\; y)\mathrm{div}17|$ is a legal expression.

Extend the transition system semantics of channel systems such that expressions are allowed in send statements.

Answer

- S needs to be extended with Eval(expr)
- The relation '$\rightarrow$' needs to be extended with Eval(expr) where it states 'a value along channel c'
- L needs to be extended with Cond(expr)

Ex. 2.9

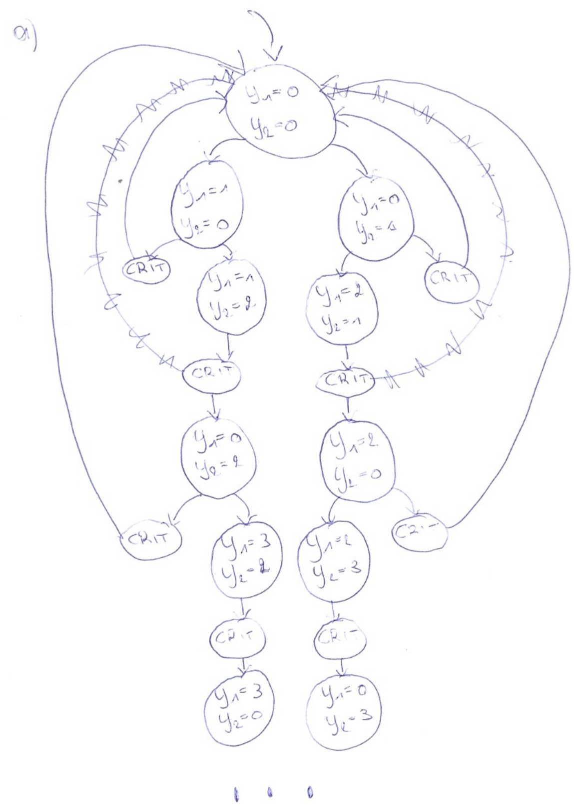
Question

Consider the following mutual exclusion algorithm that uses the shared variables y1 and y2 (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**

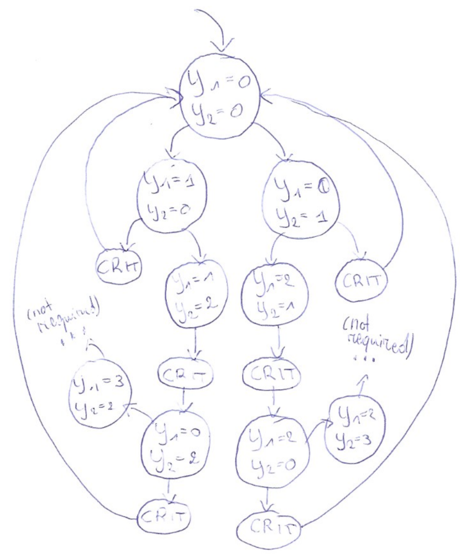
(a) Give the program graph representations of both processes. (A pictorial representation suffices.)
(b) Give the reachable part of the transition system of P1 $\|$ P2 where y1 $\leq$ 2 and y2 $\leq$ 2.
(c) Describe an execution that shows that the entire transition system is infinite.
(d) Check whether 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?

Answer


a)
y1=0 y2=0
y1=1 y2=0
CRIT
y1=1 y2=2
CRIT
y1=0 y2=2
CRIT
y1=3 y2=2
CRIT
y1=3 y2=0
y1=2 y2=1
CRIT
y1=2 y2=0
y1=2 y2=3
y1=0 y2=3
CRIT

-

b)



-

- $P_1- > P_2- > P_1- > P_2- >$ ... basically just eternally switching between increasing y1 and y2 before the other can gain access to the critical section

- A process can only enter the critical section if the other process just left the critical section or if the counter of process i is smaller then that of process j. Since a process can only make its own counter greater than that of the other process, it will only get access if the other process left the critical section. If process i increments it counter before process j, process j will have access to the CS before process i.

- Due to the way the counters increment it cannot happen that both processes are mutually waiting for each other.

- No: either the other process has their counter set to 0 or it wants access to the CS next.

Ex. 2.10

Question

Consider the following mutual exclusion algorithm that was proposed 1966 [221] as a simplification of Dijkstras mutual exclusion algorithm in case there are just two processes:

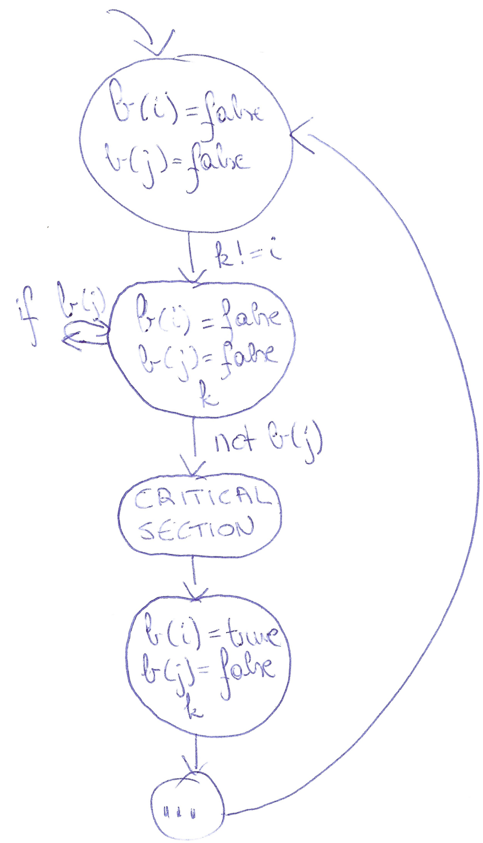
```
Boolean array b(0;1) integer k, i, j,
comment  This is the program for computer i, which may be
   either 0 or 1, computer j =/= i is the other one, 1 or 0;
C0: b(i) := false;
C1: if k != i then begin
C2: if not b(j) then go to C2;
    else k := i; go to C1 end;
    else critical section;
    b(i)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

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

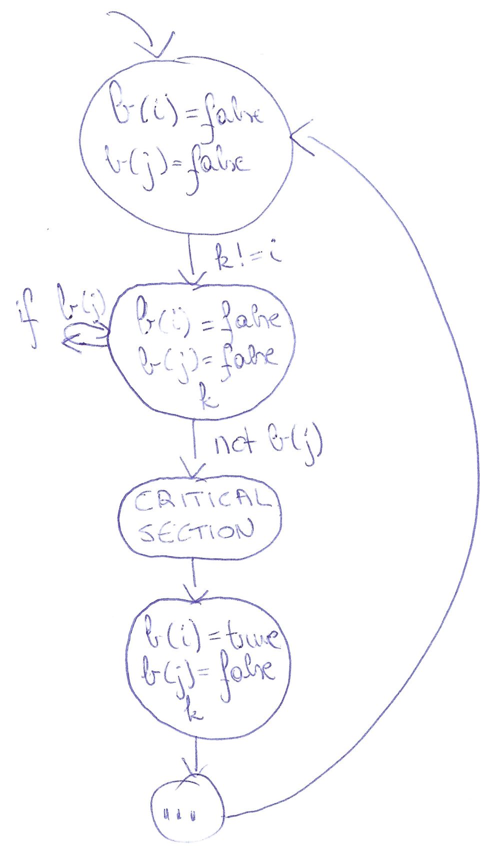
(a) Give the program graph representations for a single process. (A pictorial representation suffices.)
(b) Give the reachable part of the transition system of P1 ∥ P2.
(c) Check whether the algorithm indeed ensures mutual exclusion.

Answer

a)


b(i)=false
b(j)=false
k!=i
if b(j)
b(i)=false
b(j)=false
k
not b(j)
CRITICAL
SECTION
b(i)=true
b(j)=false
k
...

•



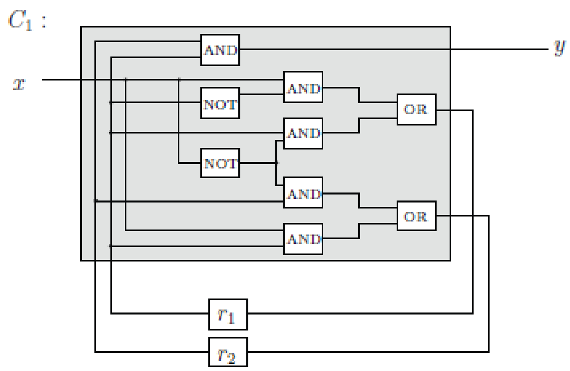
b(i) = false
b(j) = false
k != i
if b(j)
b(i) = false
b(j) = false
k
not b(j)
CRITICAL SECTION
b(i) = true
b(j) = false
k
...

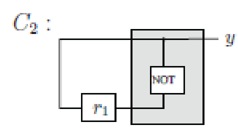
-
- /

Ex. 2.11

Question

Consider the following two sequential hardware circuits C1 and C2:

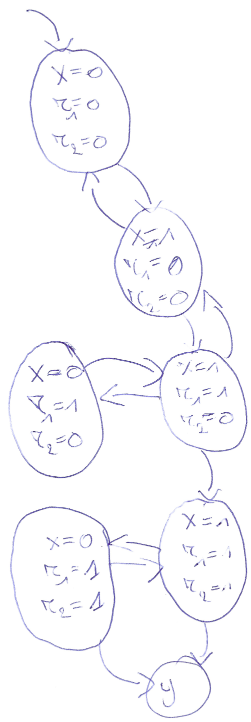




(a) Give the transition system representation TS(C1) of the circuit C1.
(b) Let TS(C2) be the transition system of the circuit C2. Outline the transition system TS(C1) $\otimes$ TS(C2).

Answer

a)



-
- C1 will get an extra line from r1 to an OR gate (other input is result of r1 AND r2’). Result of the OR gate goes to y and is negated by a NOT gate before going to an OR gate that leads to r1 (along with the original input for r1).