Exercises on Transition Systems

Beau De Clercq

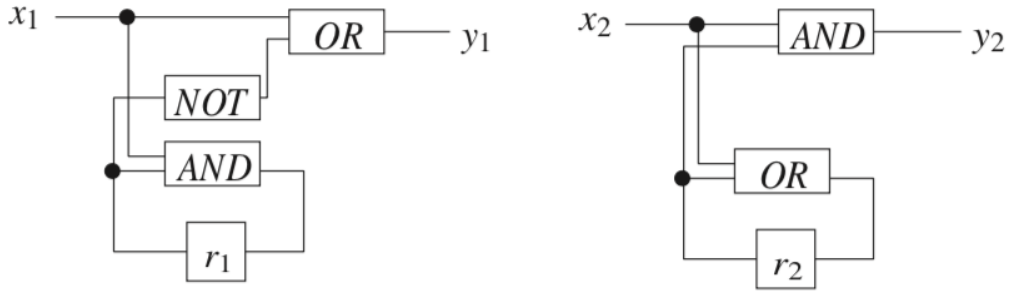
October 2019

Contents

Ex. 2.1

Question

Consider the following two sequential hardware circuits:

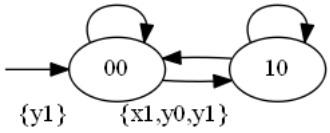


(a) Give the transition systems of both hardware circuits.
(b) Determine the reachable part of the transition system of the synchronous product of these transition systems. Assume that the initial values of the registers are r1=0 and r2=1.

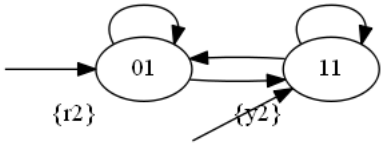
Answer

(a)

- Circuit one:

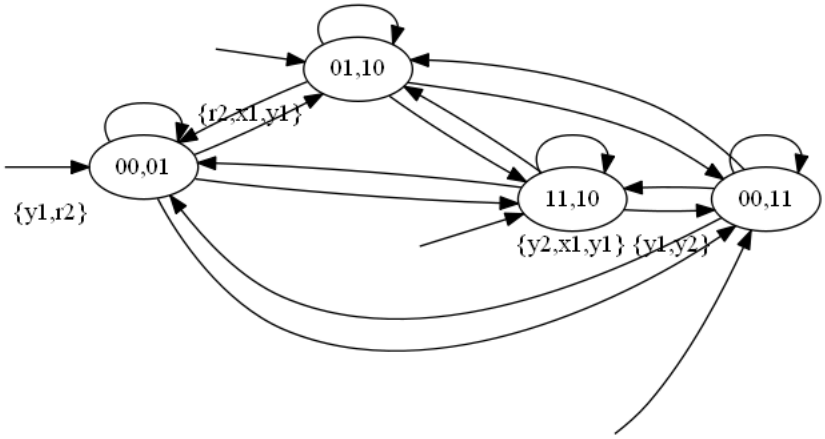


- Circuit two:



(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;
P_1 || P_2 || P_3
```

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

Answer

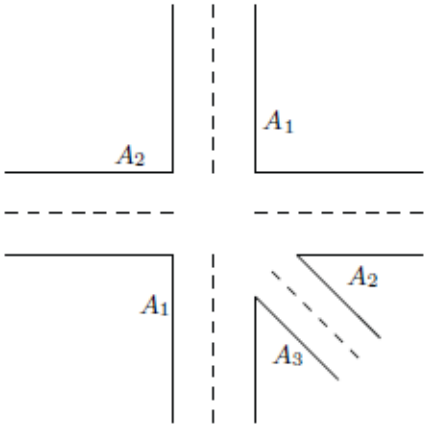
The answer to this question depends on how you interpret the question.

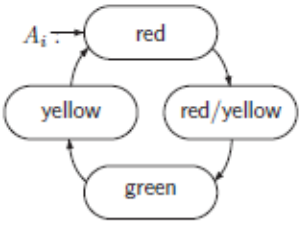
- If you allow processess to run completely independent from each other, it is possible to have an execution that halts with x = 2:
  - First load x := 0 in P_1, P_2, P_3
  - Then let P_1 finish (x == 10)
  - After that, let P_2 run 9 times and store x = 9
  - Increment P_3 1 time and store
  - Load 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

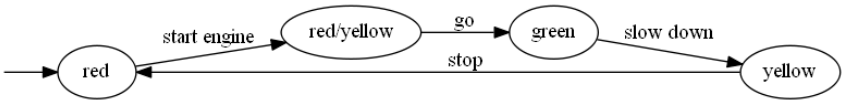
Consider the following street junction with the specification of a traffic light as outlined on the right.

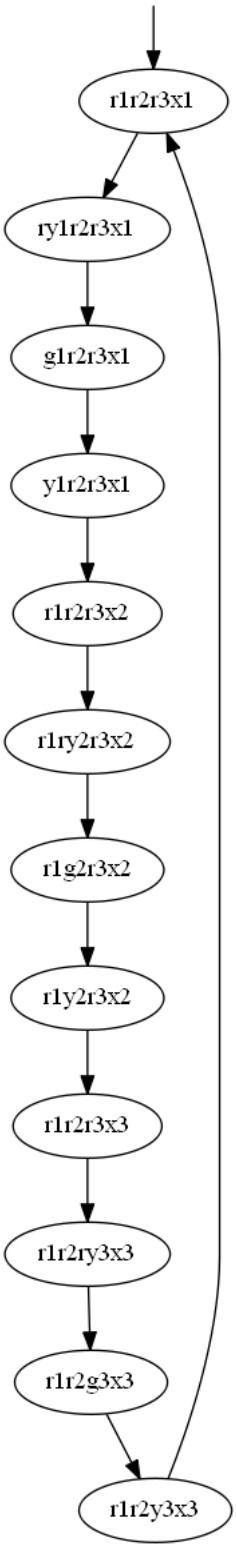




(a) Choose appropriate actions and label the transitions of the traffic light transition system accordingly.
(b) Give the transition system representation of a (reasonable) controller C that switches the green signal lamps in the following order: A1,A2,A3, A1,A2,A3, ...
(Hint: Choose an appropriate communication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

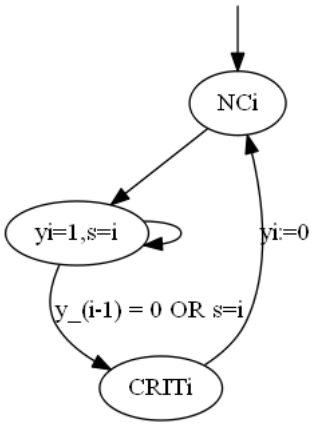
Question

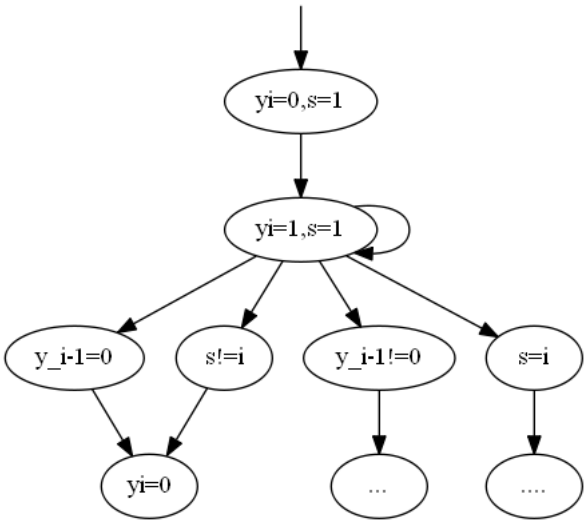
The following program is a mutual exclusion protocol for two processes due to Pnueli [118]. There is a single shared variable s which is either 0 or 1, and initially 1. Besides, each process has a local Boolean variable y that initially equals 0. The program text for process P_i (i = 0, 1) is as follows:

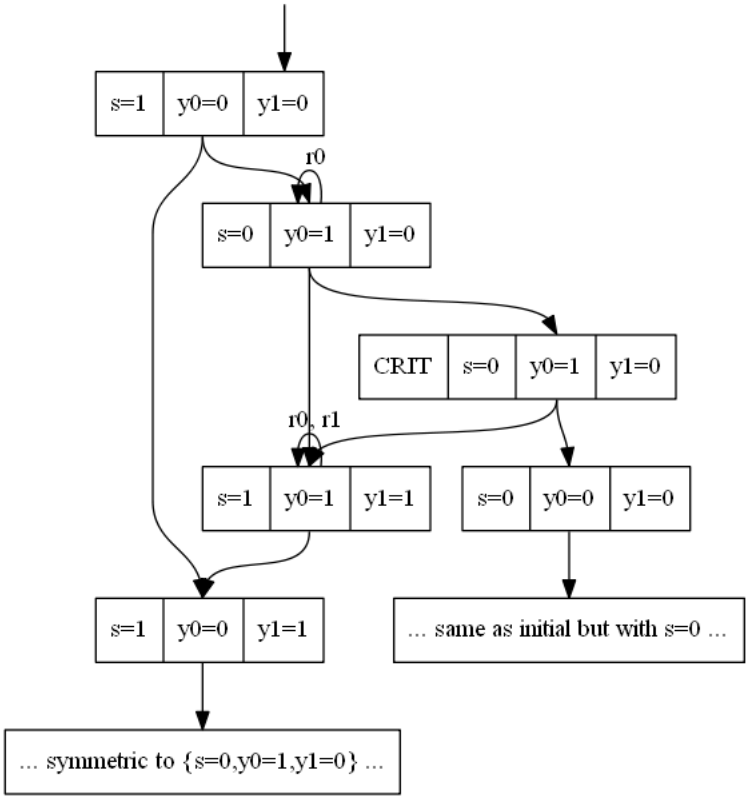
l0: **loop forever do**
begin
l1: Noncritical section
l2: $(y_i, s) := (1, i)$;
l3: **wait until** $((y_{1-i} = 0) \vee (s \neq i))$;
l4: Critical section
l5: $y_i := 0$
end.

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

- 

- 



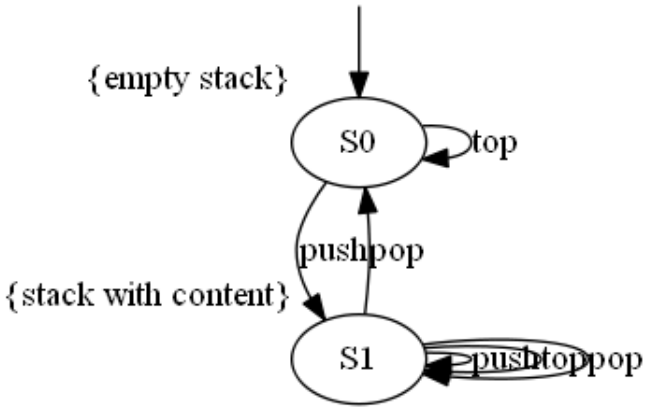
- P = $\{CRIT_1, CRIT_2, r_1, r_2\}$
  ME is ensured:
- Starvation freedom is not ensured:

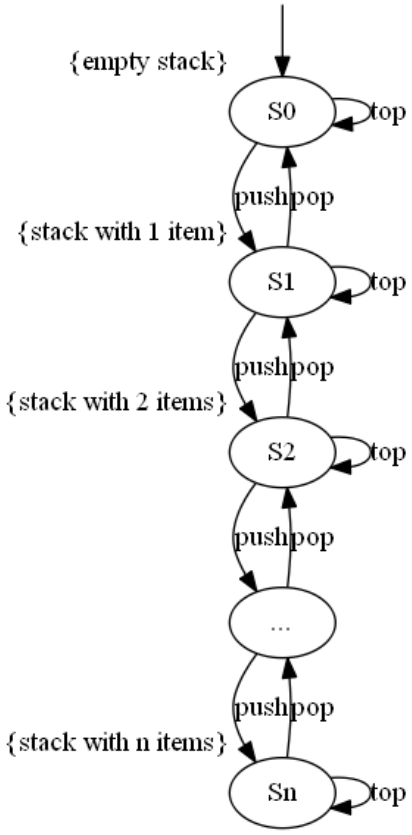
Ex. 2.6

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 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

Ex. 2.8

Question

Answer

Ex. 2.9

Question

Answer

Ex. 2.10

Question

Answer

Ex. 2.11

Question

Answer