Chapter 2-2

Modeling Concurrent Systems

Transition system

- State: the behavior of systems at a certain moment.
  - A state of a traffic light is the color of the light.
  - A state of a sequential program is the current values of variables and program counter.
  - A state of a synchronous hardware circuit the value of input bits and registers .

Transition systems

- Transition specify the evolution from one state to another.
  - Traffic light is a switch from one color to another.
  - Transition of a sequential program is the execution of a statement.
  - A transition of a synchronous hardware circuit is the change of registers and output bits on a new set of input.

Transition system

- Action: communication mechanisms between processes.
- Atomic propositions: formalize temporal characteristics.
  - X=10
  - X<200
  - There is more than a liter of fluid in the tank
  - There are no customers in the shop.

Transition system(TS)

- A *transition system TS* is a tuple (S, Act, $\rightarrow$, I, AP, L) where
  - S is a set of states,
  - Act is a set of actions,
  - $\rightarrow \subseteq S \times Act \times S$ is a transition relation,
  - $I \subseteq S$ is a set of initial states,
  - AP is a set of atomic propositions, and
  - $L : S \rightarrow 2^{AP}$ is a labeling function.
- *TS* is called *finite* if S, Act, and AP are finite.

TS(cont)

- $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \rightarrow$
  - a transition originating from s is selected *nondeterministically*.
  - The action α is performed and the transition system evolves from state s into the state s'.
  - when the set of initial states consists of more than one state, the start state is selected nondeterministically.
  - Actions are for modeling communication mechanisms. Others can omit actions.

TS(cont)

- Label function $L(s) \in 2^{AP}$
  - A state *s* satisfies a propositional logic formula Φ if $L(s)$ makes the formula Φ true; that is:

$$s \models \Phi \text{ iff } L(s) \models \Phi.$$

  - AP is chosen depending on the characteristics of interest.

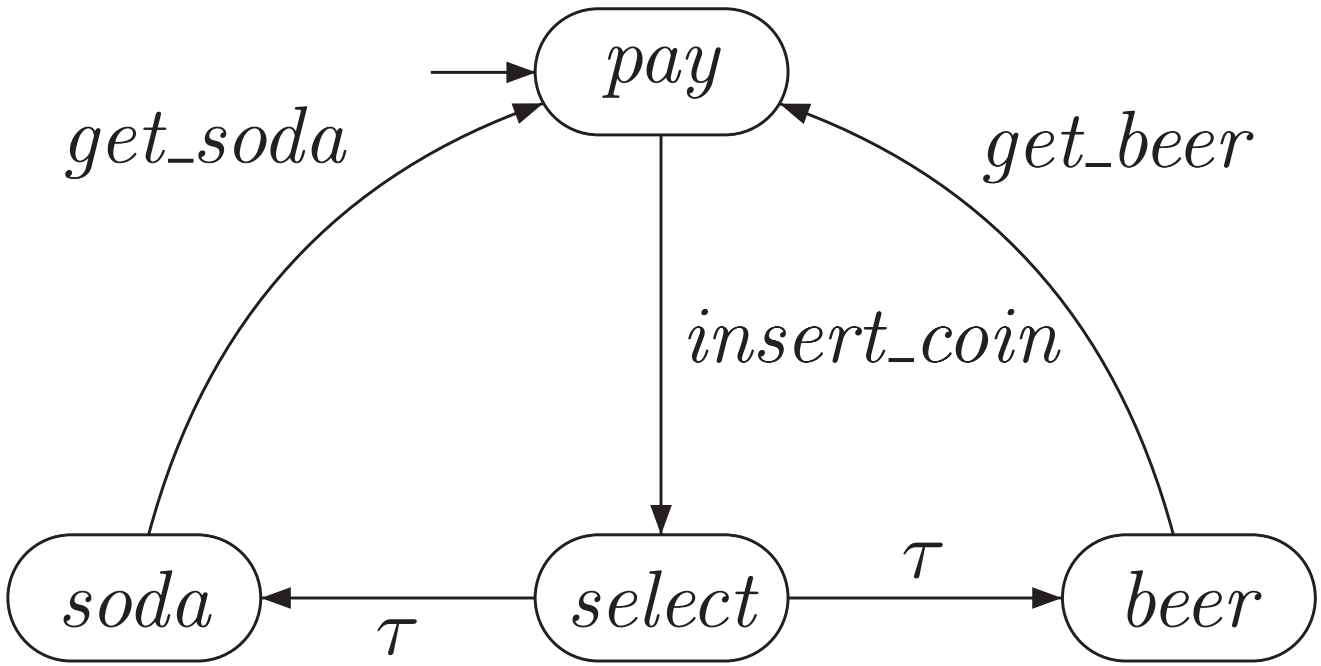
Satisfaction relation $|=$

- Evaluation for AP is function μ:AP$\rightarrow$\{0,1\}.
- Eval(AP) is the set of all evaluations for AP.
- Satisfaction relation $|=$ is a set of pair (μ,ϕ), indicating the evaluations μ for which a formula ϕ is true.
  - $\mu \;|= true$
  - $\mu \;|= a$ iff $\mu(a)=1$
  - $\mu \;|= \neg\, \phi$ iff $\mu \;|\neq \phi$
  - $\mu \;|= \phi \wedge \psi$ iff $\mu |= \phi \wedge \mu |= \psi$

Application of nondeterministic

- to model the parallel execution of independent activities by interleaving
- to model the conflict situations that arise if two processes aim to access a shared resource.
- To be so important for abstraction purposes, for underspecification
- to model the interface with an unknown or unpredictable environment.

Example: Beverage Vending



- S = { pay , select, soda, beer }
- I = { pay }
- Act = { insert_coin, get_soda, get_beer, τ }

Example(cont)

- After the insertion of a coin, the vending machine nondeterministically can choose to provide either beer or soda.
- $AP = \{ paid , drink \}$ *–property under consideration*
- Labeling function:

$$L(pay) = \varnothing,$$

$$L(soda) = L(beer) = \{ paid , drink \},$$

$$L(select) = \{ paid \}.$$

Actions and atomic propositions

- Actions are only necessary for modeling communication mechanisms
- The set of propositions AP is always chosen depending on the characteristics of interest.

Direct Successors

- Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system. For $s \in S$ and $\alpha \in Act$, the set of direct *α-successors* of s is defined as:

$$Post(s, \alpha) \quad = \quad \left\{ \, s' \in S \;\; \middle| \;\; s \xrightarrow{\alpha} s' \, \right\}$$

$$Post(s) \; = \; \bigcup_{\alpha \in Act} \; Post(s, \alpha)$$

Direct Predecessors

- The set of *α-predecessors* of *s* is defined by:

$$Pre(s,\alpha) \;=\; \left\{\, s' \in S \;\mid\; s' \xrightarrow{\alpha} s \,\right\}$$

$$Pre(s) \;=\; \bigcup_{\alpha\in Act} Pre(s,\alpha)$$

Direct Predecessors and Successors expanded to subset

$$Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), \quad Post(C) = \bigcup_{s \in C} Post(s)$$

$$Pre(C, \alpha) = \bigcup_{s \in C} Pre(s, \alpha), \quad Pre(C) = \bigcup_{s \in C} Pre(s)$$

Terminal State of a TS

- State s in TS is called *terminal* iff:
  - $Post(s) = \emptyset$.
- For a sequential computer program, terminal states represents the termination of the program.
- For parallel systems, terminal states are usually considered to be undesired

Deterministic Transition System

- Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system.
  - TS is called *action-deterministic* if for all states s and actions α

    $| I | \leq 1$ and $| Post(s, \alpha) | \leq 1$.
  - TS is called *AP-deterministic* if for all states s and $A \in 2^{AP}$

    $| I | \leq 1$ and

    $|Post(s) \cap \{s' \in S \mid L(s') = A\}| \leq 1$

Execution (also called run)

- Execution fragment
  - Let $TS = (S, Act, \to, I, AP, L)$. A *finite* execution fragment of TS:

$$\varrho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\ldots\alpha_n\,s_n$$

$$\text{such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i < n$$

Where n≥0 is the length of the finite execution fragment.

Infinite execution fragment

- An *infinite* execution fragment ρ of *TS* :

$$\rho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\,s_2\,\alpha_3\ldots$$

$$\text{such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i.$$

Maximal and Initial Execution Fragment

- A *maximal* execution fragment is either a finite execution fragment that ends in a terminal state, or an infinite execution fragment.
- An execution fragment is called *initial* if it starts in an initial state.

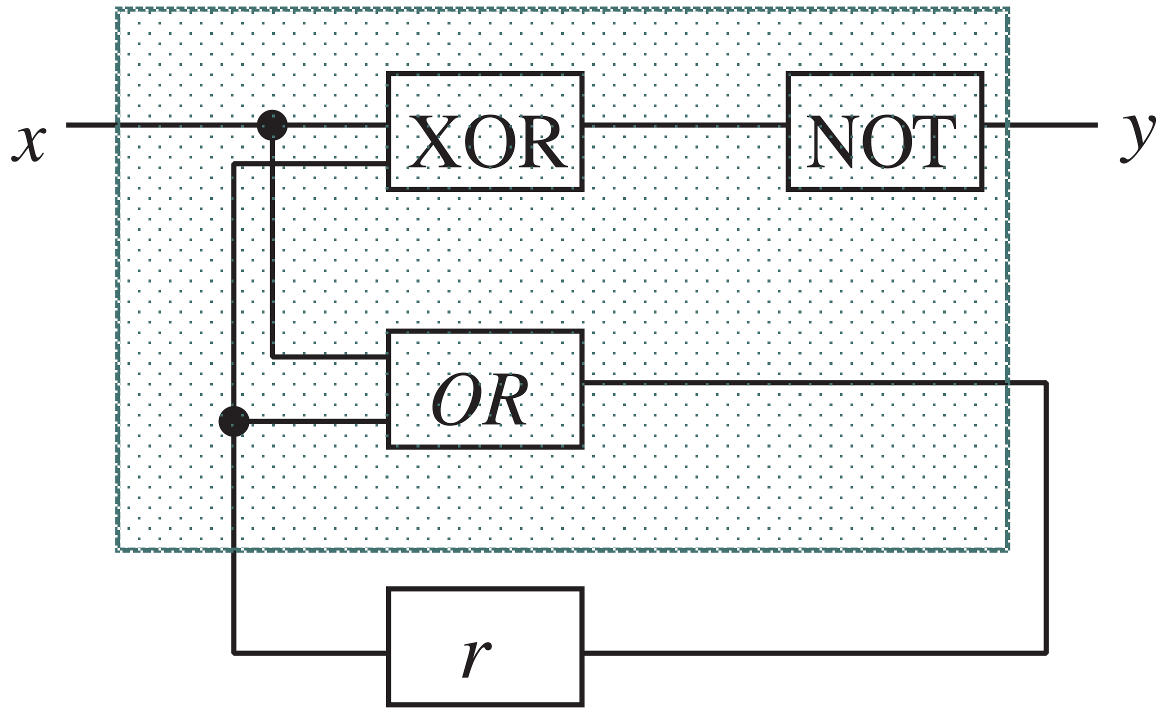
Execution and Reachable states

- An *execution* of *TS* is an initial, maximal execution fragment.
- Let $TS = (S, Act, \rightarrow, I, AP, L)$, a state $s \in S$ is called *reachable* in *TS* if there exists an initial, finite execution fragment

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n = s$$

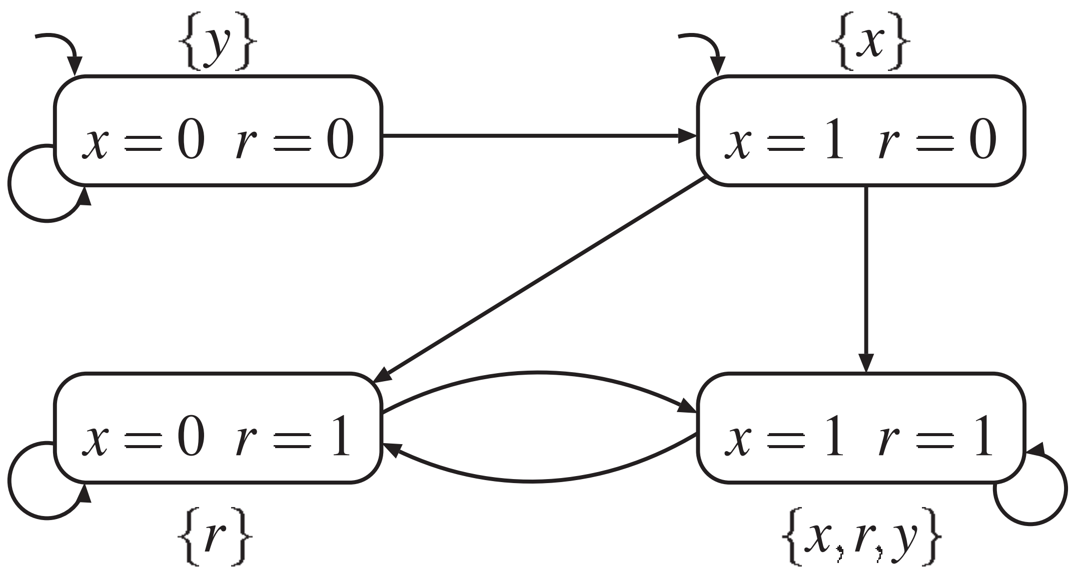
- *Reach*(*TS*) denotes the set of all reachable states in *TS*.

Sequential Hardware Circuits



$$\lambda_y = \neg(x \oplus r) \qquad\qquad \delta_r = x \vee r\ .$$

Sequential Hardware Circuits(con’t)



- state space: S = Eval(x, r)----the set of evaluation of x and r
- AP={x,y,r}
- Label function

Sequential Hardware Circuits(con’t)

- property: the output y is set infinitely often.
- AP’={x,y}
- r is invisible.

Sequential Hardware Circuits(con't)

- the general approach for sequential hardware circuits
  - state: $Eval(x_1,\ldots,x_n,r_1,\ldots,r_k)$
  - action: $\{\tau\}$
  - $AP=\{x_1,\ldots,x_n,y_1,\ldots,y_m,r_1,\ldots,r_k)$
  - transition:
    - $(a_1,\ldots,a_n,c_1,\ldots,c_k)\rightarrow(a'_1,\ldots,a'_n,c'_1,\ldots,c'_k)$

Data-Dependent Systems

- modeling conditional branching as a TS, the conditions of transitions could be omitted and could be replaced by nondeterminism;
  - if x%2=1 then x:=x+1; else x:=2x; if
- Result
  - very abstract transition system
  - a few relevant properties can be verified.
- New method
  - Conditional transition---program graph
  - Unfolded into a TS

Beverage Vending Machine

- counts the number of soda and beer bottles and returns inserted coins if the vending machine is empty.
- Location: start, select
- Conditional transition----- g:α

$$start \xhookrightarrow{true\ :\ coin} select$$

$$start \xhookrightarrow{true\ :\ refill} start$$

Beverage Vending Machine (cont)

- Conditional transition----- g:α
  - g is a Boolean condition (guard)
  - α is an action that is possible once g holds. If g does not hold, nothing to do.

$$select \xrightarrow{nsoda > 0 \ : \ sget} start$$

$$select \xrightarrow{nbeer > 0 \ : \ bget} start$$

$$select \xrightarrow{nsoda = 0 \wedge nbeer = 0: \ ret\_coin} start$$

Beverage Vending Machine (cont)

- coin: insert coin
- ret_coin:return coin
- Other actions

Action	Effect
$refill$	$nsoda := max;\ nbeer := max$
$sget$	$nsoda := nsoda - 1$
$bget$	$nbeer := nbeer - 1$

- Variables: $nsode$, $nbeer$

Typed variables

- *nsoda*, *nbeer* are typed variables: Var
- The domain of x: dom(x)
- The set of evaluation: Eval(Var)
- The set of Boolean condition over Var:Cond(Var)
- Propositional symbol: $x \in D$, where
  - $x=(x_1,x_2,\ldots x_n)$
  - $D=D_1 \times D_2 \times \ldots \times D_n$
  - Condition: propositional symbol
    - $-3<x-x'<=5$ and $x<=2*x'$

effect

- The effect of the actions is formalized by means of a mapping:
  - Effect:Act x Eval(Var)$\rightarrow$ Eval(Var)
  - How the evaluation η of variables is changed by performing an action.

Program Graph (PG)

- a digraph whose edges are labeled with conditions on these variables and actions.
- The effect of the actions is formalized by means of a mapping
  - $\text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})$
  - α action $x := y+5$,
  - η evaluation $\eta(x) = 17$,$\eta(y) = -2$,
  - $\text{Effect}(\alpha,\eta)(x)=\eta(y) + 5 =-2 + 5 = 3$
  - $\text{Effect}(\alpha, \eta)(y) =\eta(y) = -2$

Program Graph (PG)

- A PG over set Var of typed variables is a tuple $(Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$
- Loc is a set of locations
- Act is a set of actions,
- $Effect : Act \times \mathrm{Eval(Var)} \rightarrow \mathrm{Eval(Var)}$
- $\hookrightarrow \subseteq Loc \times \mathrm{Cond(Var)} \times Act \times Loc$ is the conditional transition relation,
- $Loc_0 \subseteq Loc$ is a set of initial locations,
- $g_0 \in \mathrm{Cond(Var)}$ is the initial condition.

Beverage Vending Machine

- $Loc = \{start, select\}$
- $Loc_0 = \{start\}$ Var$=\{nsoda, nbeer\}$
- $Act = \{ bget, sget, coin, ret\_coin, refill \}$
- $g_0 = (nsoda = max \wedge nbeer = max)$
- Effect
  - Effect$(coin, \eta) = \eta$
  - Effect$(ret_coin, \eta) = \eta$
  - Effect$(sget, \eta) = \eta[nsoda := nsoda - 1]$
  - Effect$(bget, \eta) = \eta[nbeer := nbeer - 1]$
  - Effect$(refill, \eta) = [nsoda := max, nbeer := max]$

Translation from PG into TS

Definition 2.15. Transition System Semantics of a Program Graph

The transition system $TS(PG)$ of program graph

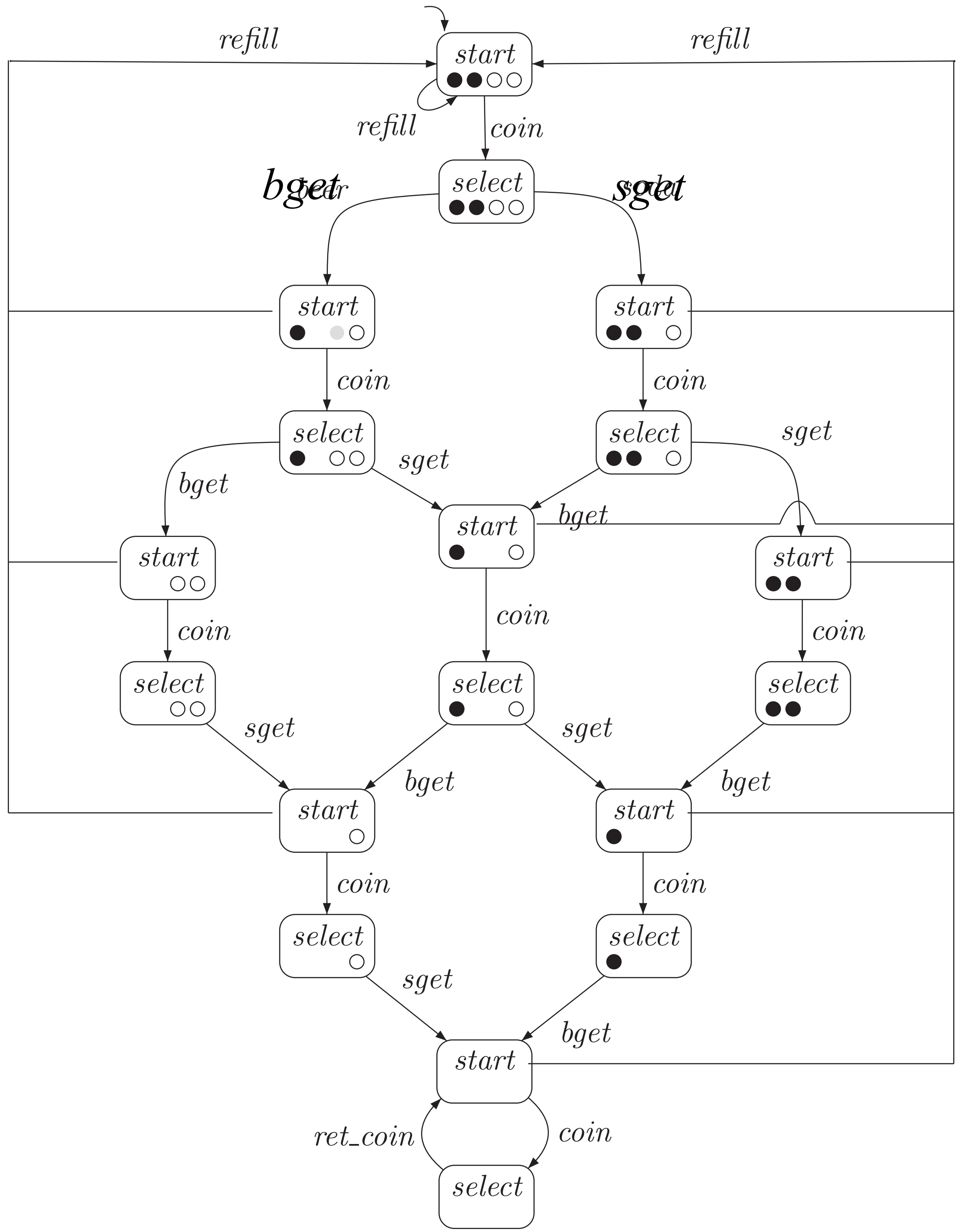
$$PG = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$$

over set Var of variables is the tuple $(S, Act, \longrightarrow, I, AP, L)$ where

- $S = Loc \times Eval(Var)$
- $\longrightarrow \subseteq S \times Act \times S$ is defined by the following rule (see remark below):

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \quad \wedge \quad \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

- $I = \{\langle \ell, \eta \rangle \mid \ell \in Loc_0, \eta \models g_0\}$
- $AP = Loc \cup Cond(Var)$
- $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{g \in Cond(Var) \mid \eta \models g\}$.

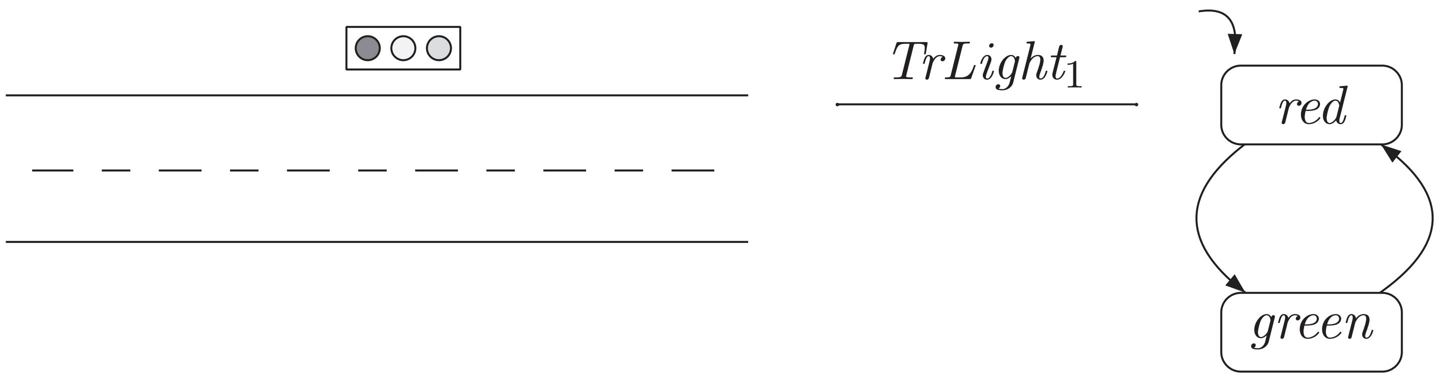

refill
start
refill
refill
coin
bget
select
sget
start
start
coin
coin
select
select
bget
sget
sget
start
bget
start
start
coin
coin
coin
select
select
select
sget
sget
bget
bget
start
start
coin
coin
select
select
sget
bget
start
ret_coin
coin
select

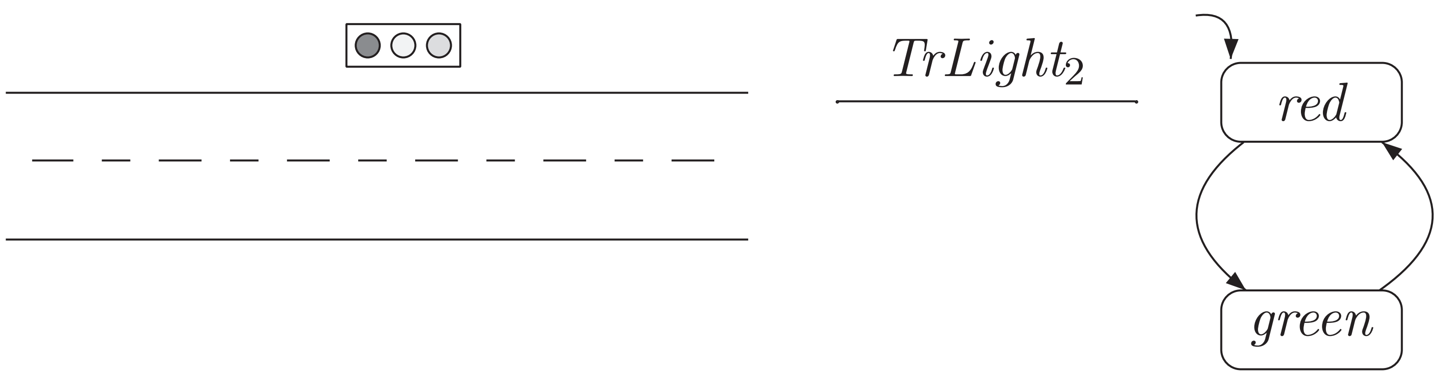
Parallelism and Communication

- define an operator, such that:
- $TS = TS_1 \mid\mid TS_2 \mid\mid \ \ldots \mid\mid TS_n$
- is a transition system that specifies the behavior of the parallel composition of transition systems TS_1 through TS_n.
- interleaving: the system is composed of a set of independent components.
  - example: two processes, P || Q
  - the order: PQPPQP..., or PPPQQQ, or PQPQPQ....

Concurrency and Interleaving

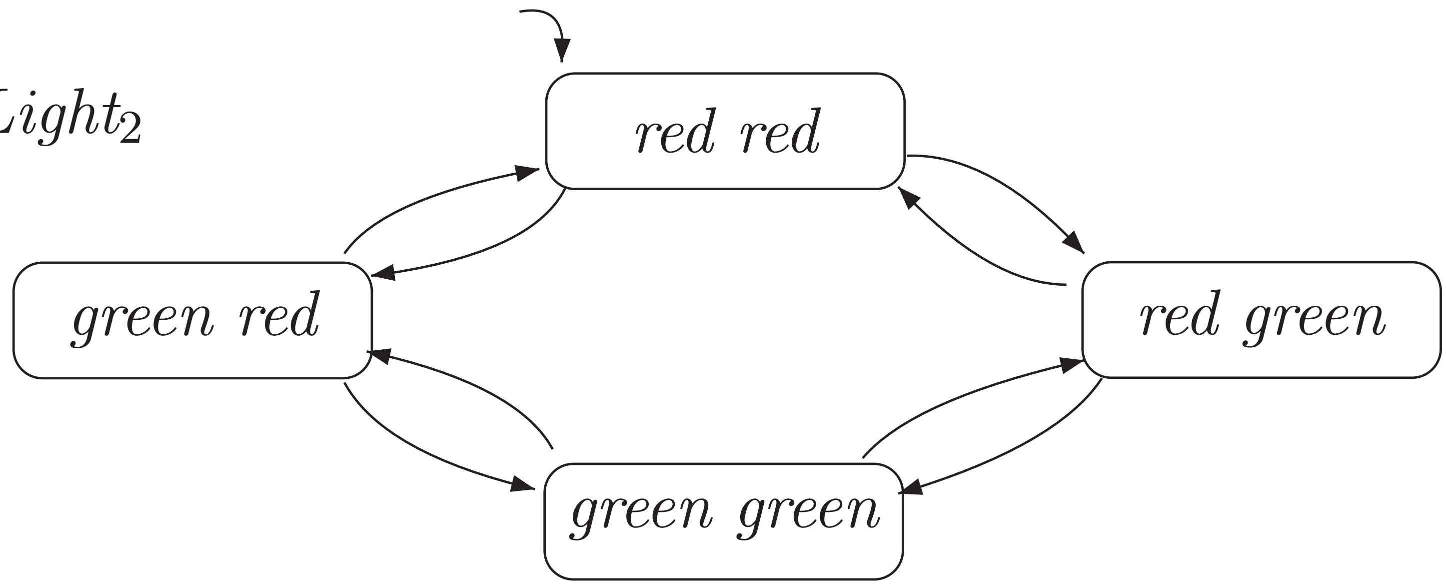
- Two Independent Traffic Lights----the transition systems of two traffic lights are nonintersecting (i.e., parallel) roads.





Two Independent Traffic Lights(cont)

$TrLight_1 \; ||| \; TrLight_2$



where ||| denotes the interleaving operator.

Interleaving of Transition Systems

- Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$ i=1, 2,

$TS_1 \mid\mid\mid TS_2$

$= (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$,

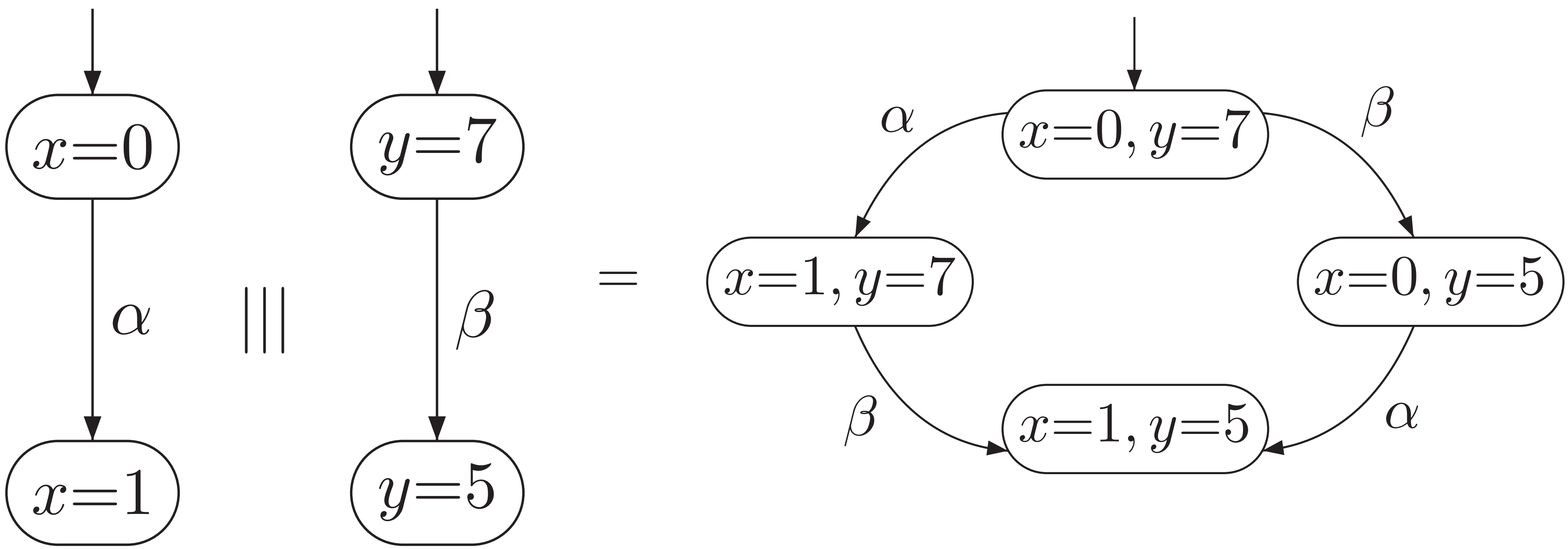
where $\rightarrow$

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \quad \text{and} \quad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

- the labeling function is defined by $L(<s_1, s_2>) = L(s_1) \cup L(s_2)$.

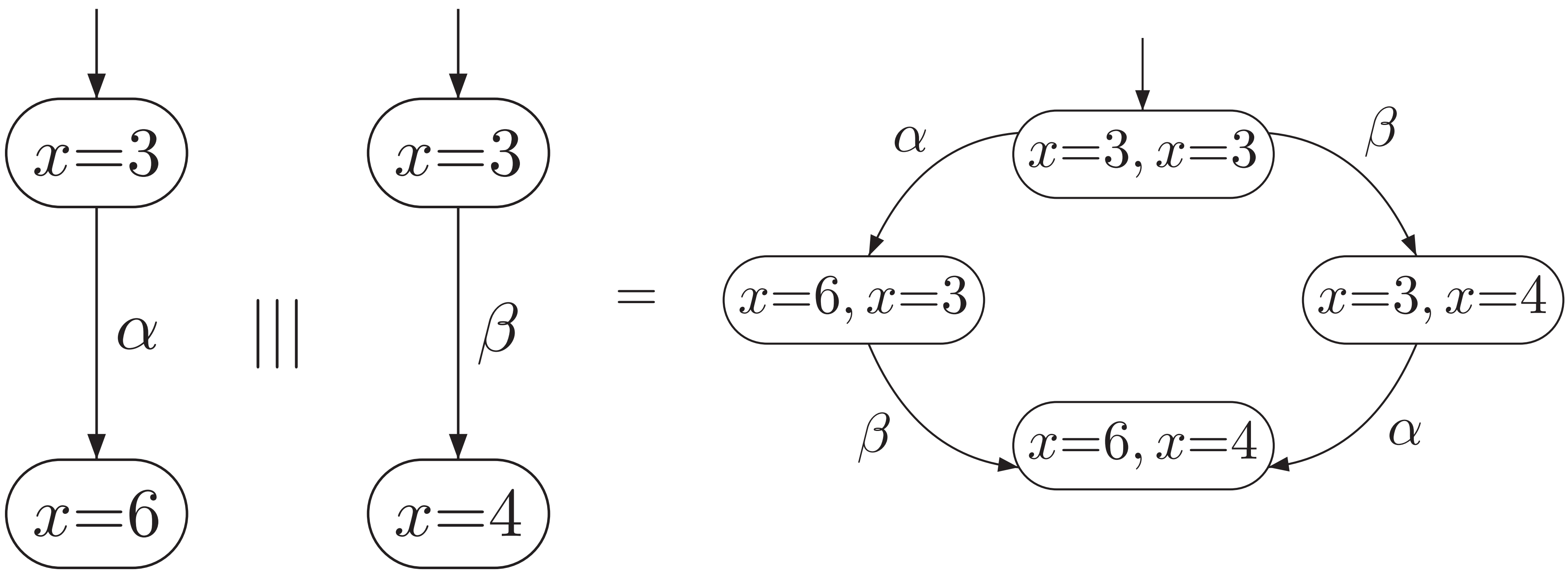
concurrent execution of independent activities.

$\underbrace{x := x + 1}_{=\alpha} \;|||\; \underbrace{y := y - 2}_{=\beta}.$ initially x = 0 and y = 7,



Communication via Shared variables

- Example: $\underbrace{x := 2 \cdot x}_{\text{action } \alpha} \;|||\; \underbrace{x := x + 1}_{\text{action } \beta}$
- Initial x=3
- TS(PG1) ||| TS(PG2)



problem

- the actions α and β access the shared variable x and therefore are competing.
- Solution: an interleaving operator will be defined on the level of program graphs
- The underlying transition system of the resulting program graph PG1 ||| PG2, i.e., TS(PG1 |||PG2) faithfully describes a parallel system whose components communicate via shared variables.

Interleaving of Program Graphs

Let $PG_i = (Loc_i, Act_i, \mathit{Effect}_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$, for i=1, 2 be two program graphs over the variables Var_i. Program graph $PG_1 \,|||\, PG_2$ over $Var_1 \cup Var_2$ is defined by

$$PG_1 \,|||\, PG_2 \;=\; (Loc_1 \times Loc_2, Act_1 \uplus Act_2, \mathit{Effect}, \hookrightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \wedge g_{0,2})$$

where $\hookrightarrow$ is defined by the rules:

$$\frac{\ell_1 \overset{g:\alpha}{\hookrightarrow}_1 \ell_1'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha}{\hookrightarrow} \langle \ell_1', \ell_2 \rangle} \quad \text{and} \quad \frac{\ell_2 \overset{g:\alpha}{\hookrightarrow}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha}{\hookrightarrow} \langle \ell_1, \ell_2' \rangle}$$

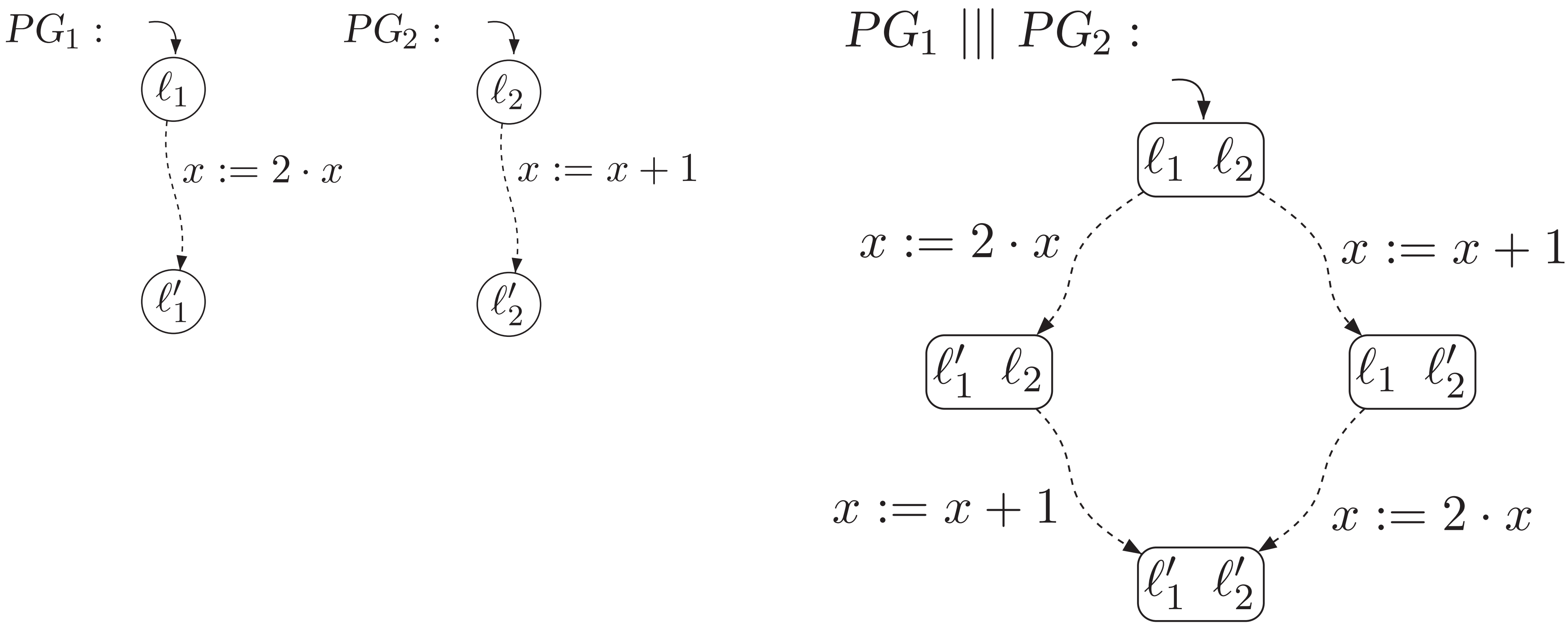
and $\mathit{Effect}(\alpha, \eta) \;=\; \mathit{Effect}_i(\alpha, \eta)$ if $\alpha \in Act_i$. ■

Interleaving of Program Graphs

- $Var_1 \cap Var_2 \neq \varnothing$------shared variables (sometimes called "global"), critical
- Var_1/Var_2 -----local variables of PG_1. noncritical
- Var_2/Var_1 -----local variables of PG_2. noncritical

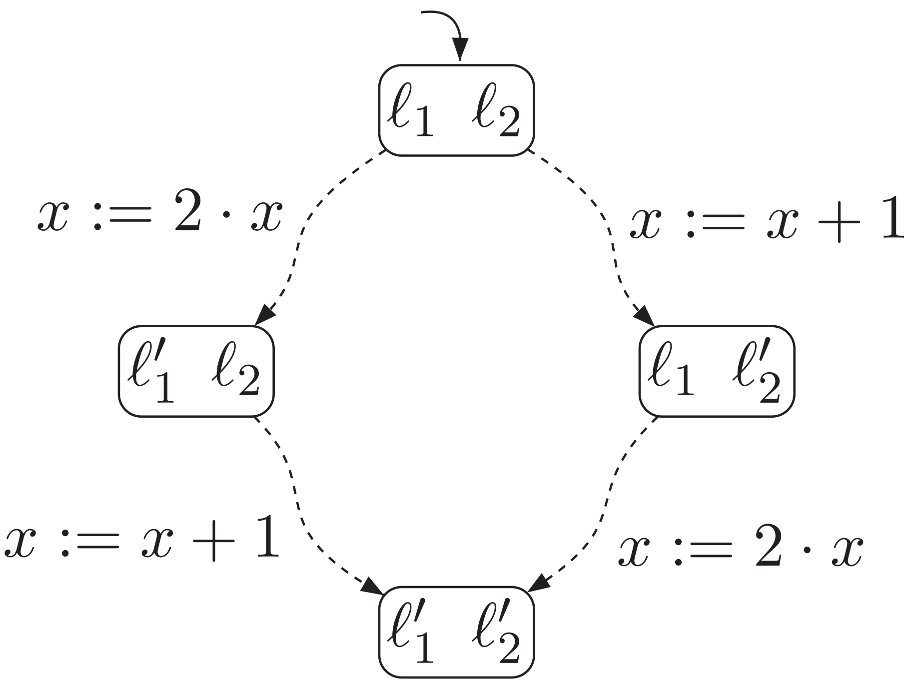
example

- x=x+1 ||| x:=2·x

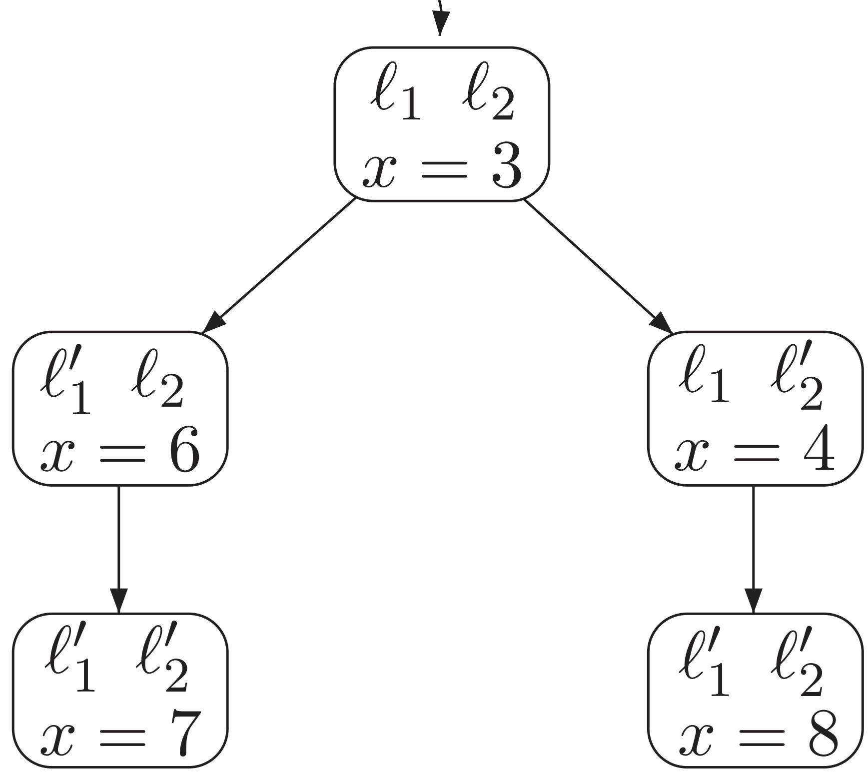


TS of example

$PG_1 \mid\mid\mid PG_2$:



$TS(PG_1 \mid\mid\mid PG_2)$



nondeterminism in a state of this transition system

- An “internal” nondeterministic choice within program graph PG_1 or PG_2,
- The interleaving of noncritical actions of PG_1 and PG_2, or
- The resolution of a contention between critical actions of PG_1 and PG_2 (concurrency).
  - The value of the shared variables depends on the order of executing Critical actions of PG_1 and PG_2.

Atomicity

- an action α with its effect being described by the statement sequence.
- declared *atomic* , surrounded by brackets<...>
- $x{:=}x+1; y=2x+1;$ if $x<12$ then $z{:=}(x-2)^2*y$;
  - $\mathrm{Effect}(\alpha,\eta)(x)= \eta(x)+1$
  - $\mathrm{Effect}(\alpha, \eta)(y)=2(\eta(x)+1)+1$
  - $\mathrm{Effect}(\alpha, \eta)(z)=\begin{cases} (\eta(x)+1-\eta(z))^2*2(\eta(x)+1)+1 & \text{if } \eta(x)+1<12 \\ \eta(z) & \text{otherwise} \end{cases}$

Mutual Exclusion with Semaphores

- share the binary semaphore y.
- y=0 indicates that the semaphore—the lock to get access to the critical section—is currently possessed by one of the processes.
- y=1, the semaphore is free.

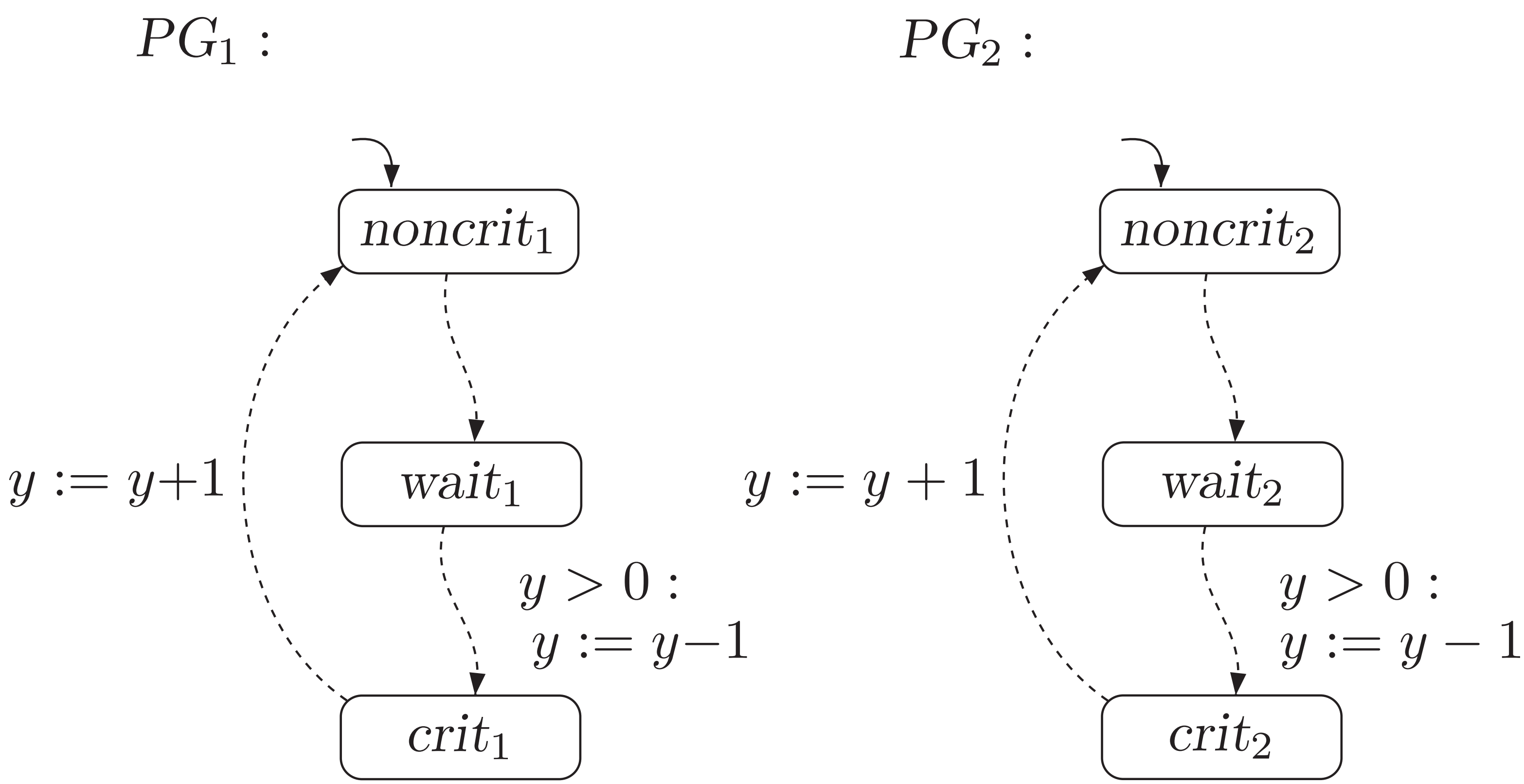
Mutual Exclusion with Semaphores (cont)

- Consider two simplified processes P_i, i=1,2 of the form:

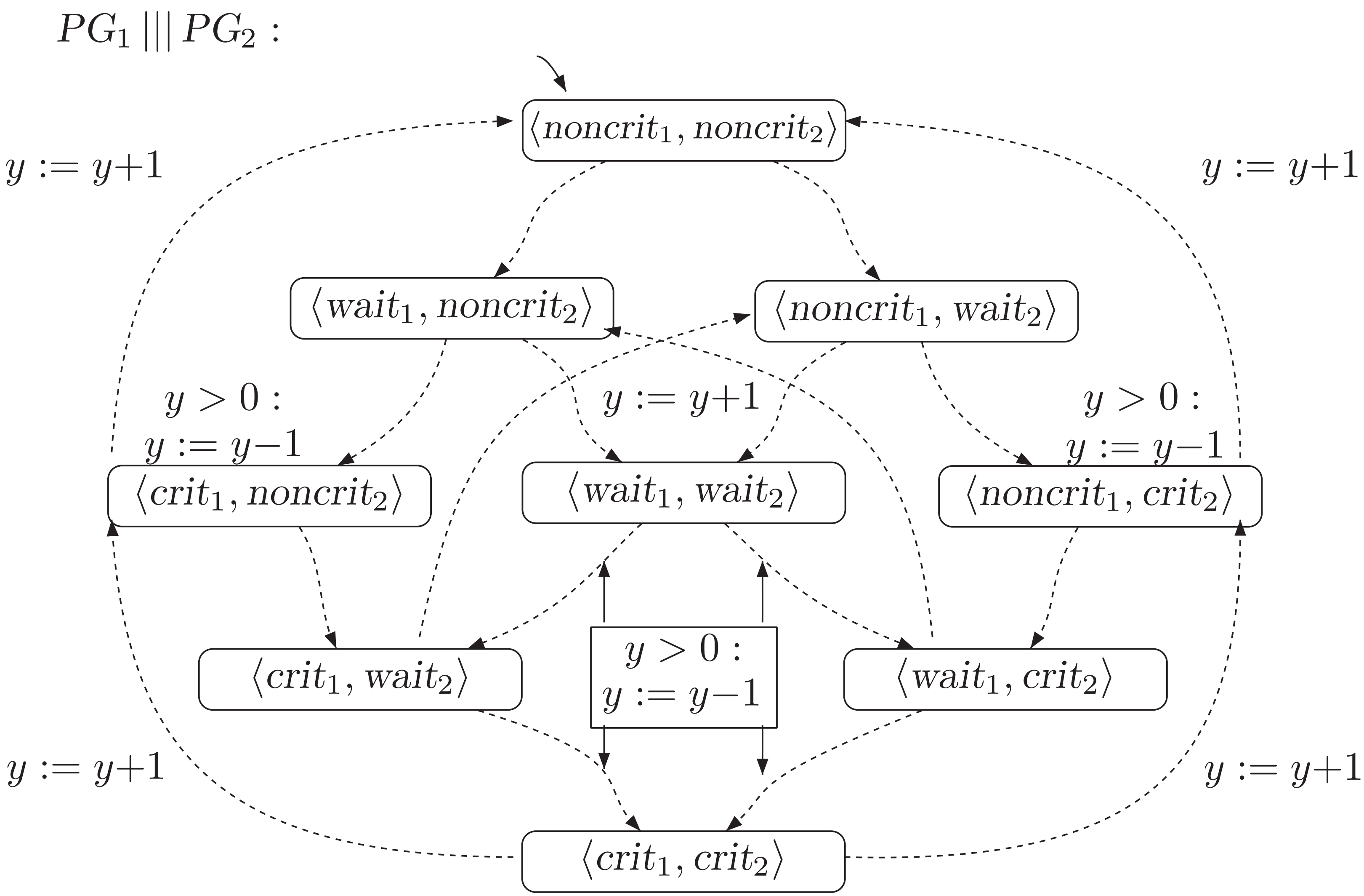
$$
\begin{array}{lll}
P_i & \textbf{loop forever} & \\
 & \vdots & \text{(* noncritical actions *)} \\
 & \textit{request} & \\
 & \text{critical section} & \\
 & \textit{release} & \\
 & \vdots & \text{(* noncritical actions *)} \\
 & \textbf{end loop} &
\end{array}
$$

Mutual Exclusion with Semaphores(cont)

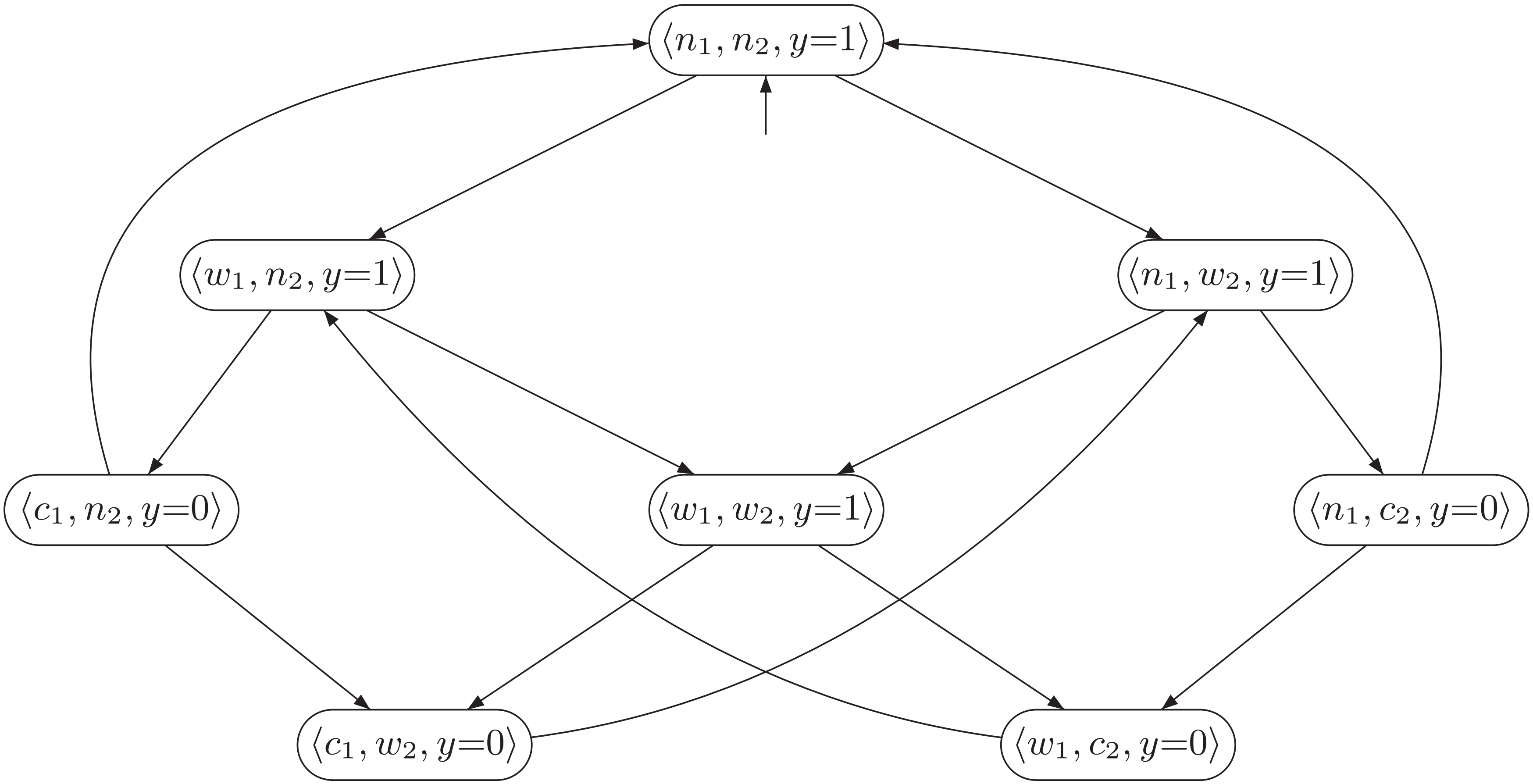
- PG_1 and PG_2.



Mutual Exclusion with Semaphores(cont)



Translation from PG into TS



Peterson's Mutual exclusion

- Shared variables b_1,b_2 and x.
- Initial b_1=b_2=false

P_1 **loop forever**

…

<b_1 := true; x := 2;>

wait until ($x = 1 \vee \neg b_2$)

do critical section **od**

b_1 := false

…

end loop

P_2 **loop forever**

…

<b_2 := true; x := 1;>

wait until ($x = 2 \vee \neg b_1$)

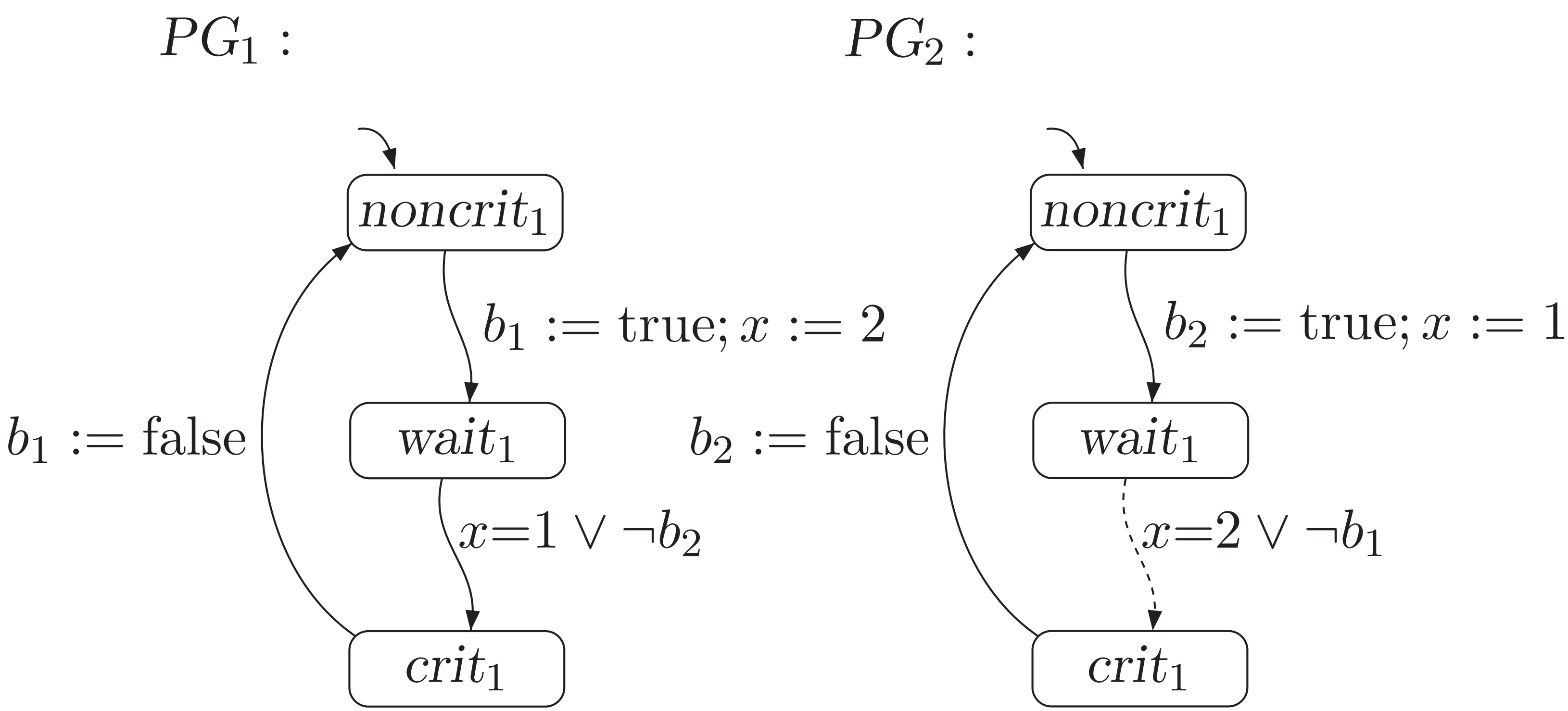
do critical section **od**

b_2 := false

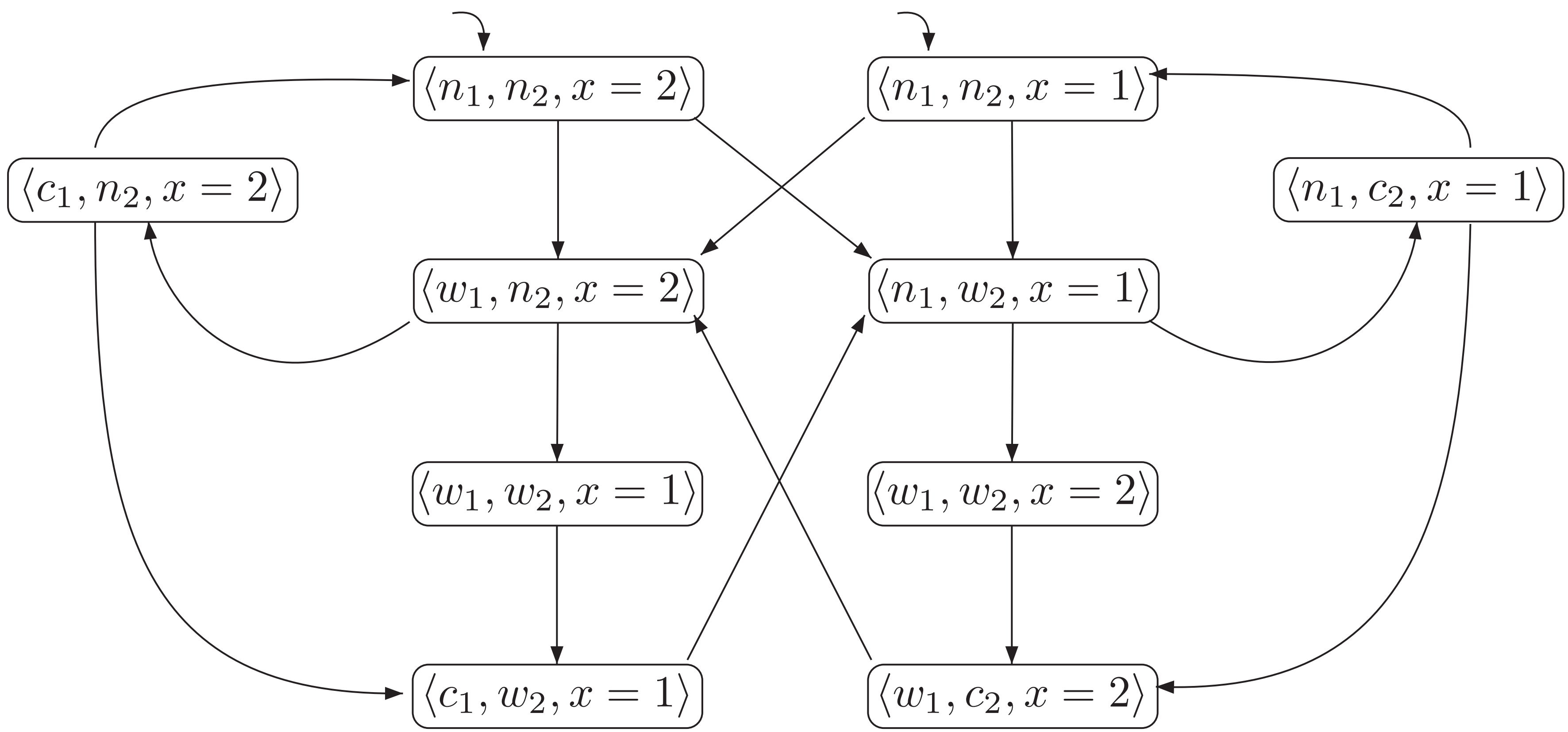
…

end loop

PG of these processes



TS of PG_1 ||| PG_2.



$\langle n_1, n_2, x = 2\rangle$
$\langle n_1, n_2, x = 1\rangle$
$\langle c_1, n_2, x = 2\rangle$
$\langle n_1, c_2, x = 1\rangle$
$\langle w_1, n_2, x = 2\rangle$
$\langle n_1, w_2, x = 1\rangle$
$\langle w_1, w_2, x = 1\rangle$
$\langle w_1, w_2, x = 2\rangle$
$\langle c_1, w_2, x = 1\rangle$
$\langle w_1, c_2, x = 2\rangle$

questions?

- if b_i:=true;x:=...in this order but in a nonatomic way, can this algorithm satisfy the mutual exclusion property?
  - » Yes
- if x:=...; b_i:=true, can this algorithm satisfy the mutual exclusion property?
  - » No.

handshaking

- Interleaving: completely autonomously, no communicative
- Shared-variable: communicates by shared variable, asynchronous
- Handshaking: synchronous fashion----they are both participating in this interaction at the same time.
  - Exchanged information can: integer, complex data structure.
  - Abstract: only communication (also called synchronization) actions are considered that represent the occurrence of a handshake
- Channel: first-in,first-out buffers that may contain messages.

Handshake (cont')

- H: handshake actions is distinguished with $\tau \notin H$
- Only if both participating processes are ready to execute the same handshake action, can message passing take place.
- actions in Act/ H are independent

Definition of handshaking
(Synchronous Message Passing)

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, i=1, 2 TSs, and $H \subseteq Act_1 \cap Act_2$ with $\tau \notin H$.

$TS_1 ||_H TS_2$ is defined :

$$TS_1 ||_H TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

where $L(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$,

$\rightarrow$ is defined:

Definition of handshaking(cont')

(Synchronous Message Passing)

- interleaving for $\alpha \notin H$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

- handshaking for $\alpha \in H$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle}$$

$TS_1 \,||\, TS_2 = TS_1 \,||_H\, TS_2$ for $H = Act_1 \cap Act_2$

- **Empty Set of Handshake Actions**
  - $TS_1 \,||_{\varnothing}\, TS_2 = TS_1 \,|||\, TS_2$.
- Handshaking is commutative
  - $TS_1 \,||_H\, TS_2 = TS_2 \,||_H\, TS_1$
- if $H \subseteq Act_1 \cap \ldots \cap Act_n$,
  - $TS = TS_1 \,||_H\, TS_2 \,||_H \ldots ||_H\, TS_n$, TS is associative

processes communicate in a pairwise

- $TS_1 \| \ldots \| TS_n$ denote the parallel
- TS_i and TS_j $(0 < i \neq j \leq n)$ synchronize , $H_{i,j} = Act_i \cap Act_j$ such that $H_{i,j} \cap Act_k = \varnothing$ for $k \notin \{i, j\}$. $\tau \notin H_{i,j}$.

- for $\alpha \in Act_i \setminus (\bigcup\limits_{\substack{0<j\leqslant n \\ i\neq j}} H_{i,j})$ and $0 < i \leqslant n$:

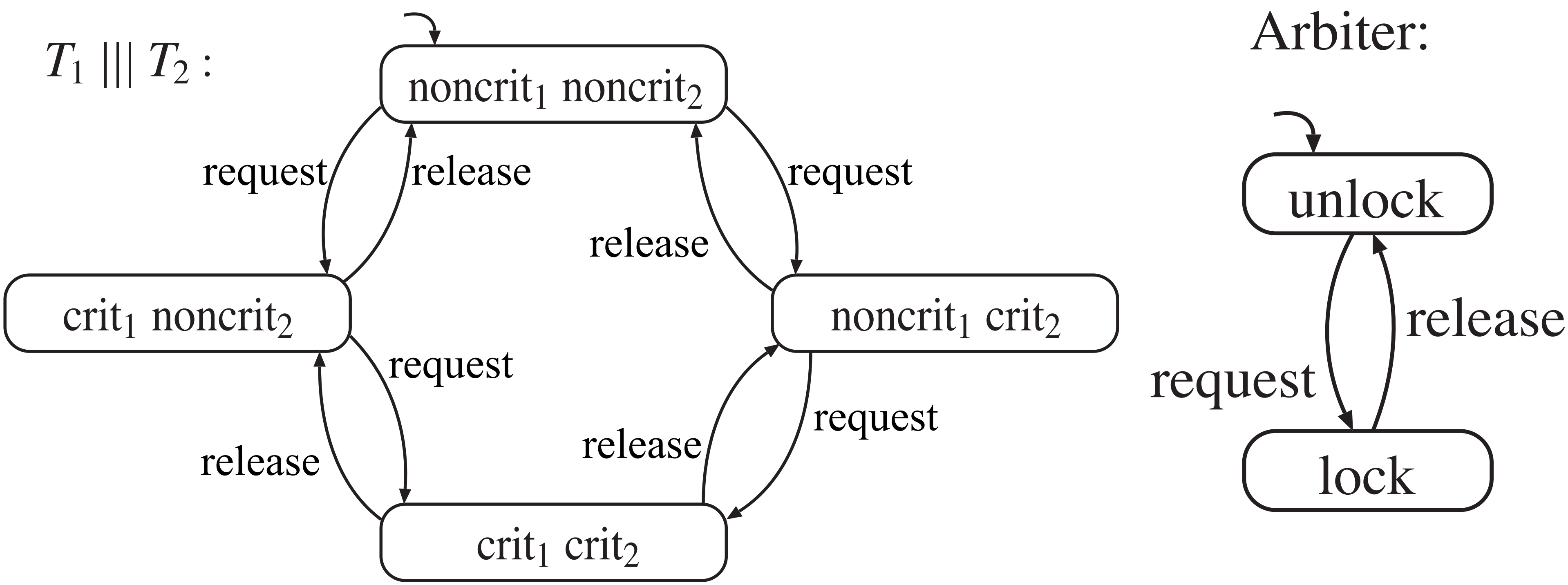
$$\frac{s_i \xrightarrow{\alpha}_i s_i'}{\langle s_1, \ldots, s_i, \ldots, s_n \rangle \xrightarrow{\alpha} \langle s_1, \ldots, s_i', \ldots s_n \rangle}$$

- for $\alpha \in H_{i,j}$ and $0 < i < j \leqslant n$:

$$\frac{s_i \xrightarrow{\alpha}_i s_i' \quad \wedge \quad s_j \xrightarrow{\alpha}_j s_j'}{\langle s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n \rangle \xrightarrow{\alpha} \langle s_1, \ldots, s_i', \ldots, s_j', \ldots, s_n \rangle}$$

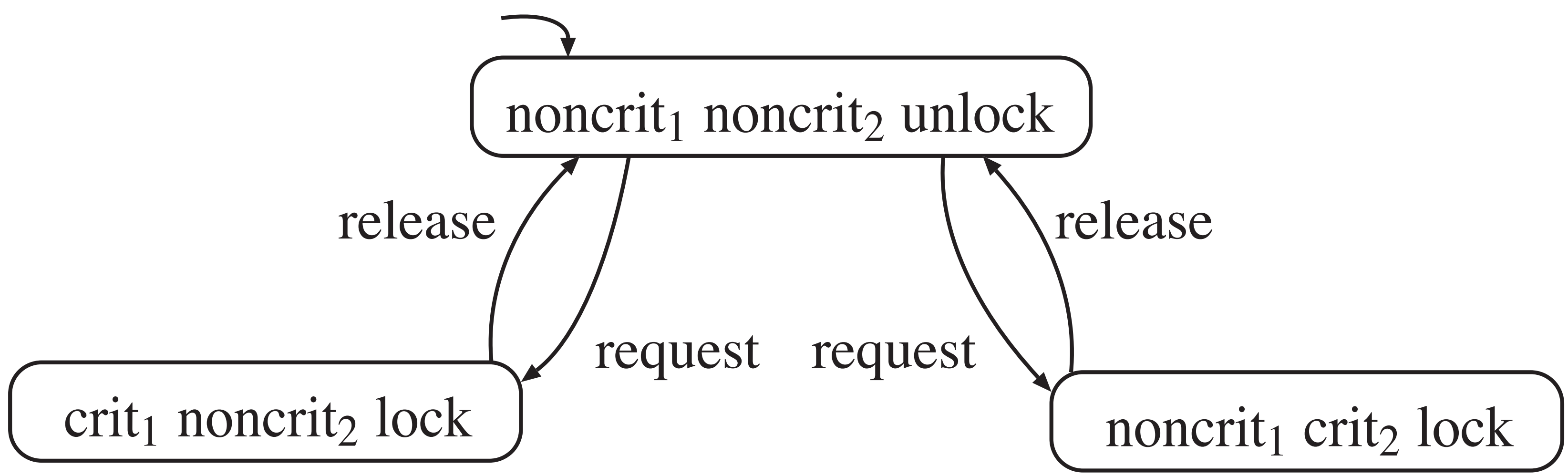
Example: Mutual exclusion by handshaking

- Add a arbiter
  - $TS_{Arb} = (TS_1 \mid\mid\mid TS_2) \mid\mid Arbiter$



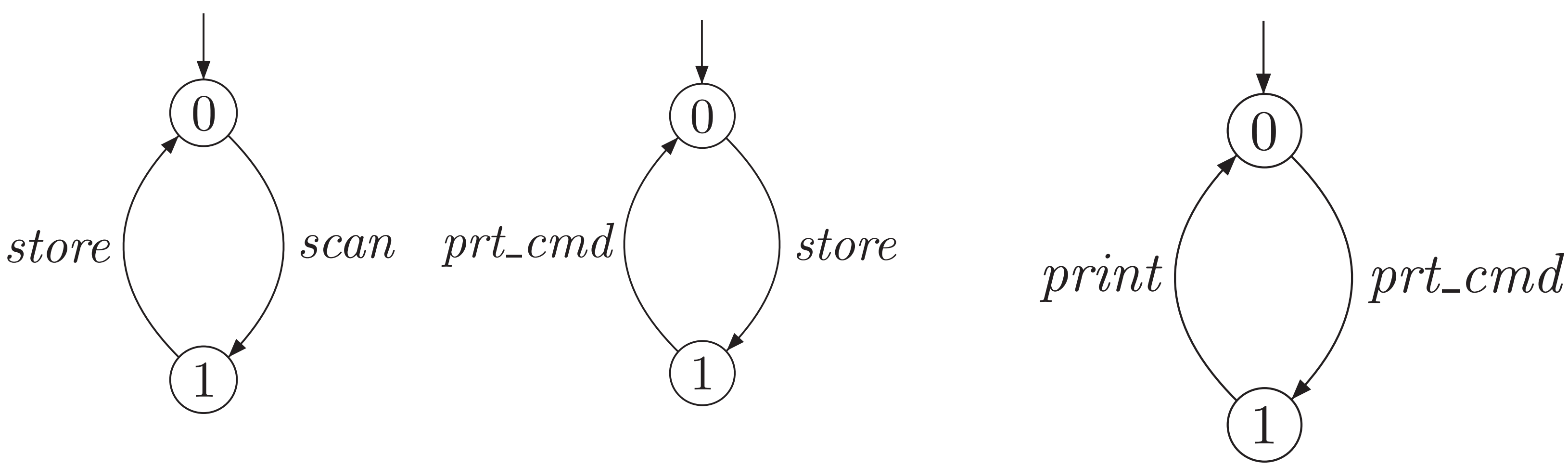
Example: Mutual exclusion by handshaking(cont')

$(T_1 \,|||\, T_2) \,||\, \text{Arbiter}:$



Example: booking system

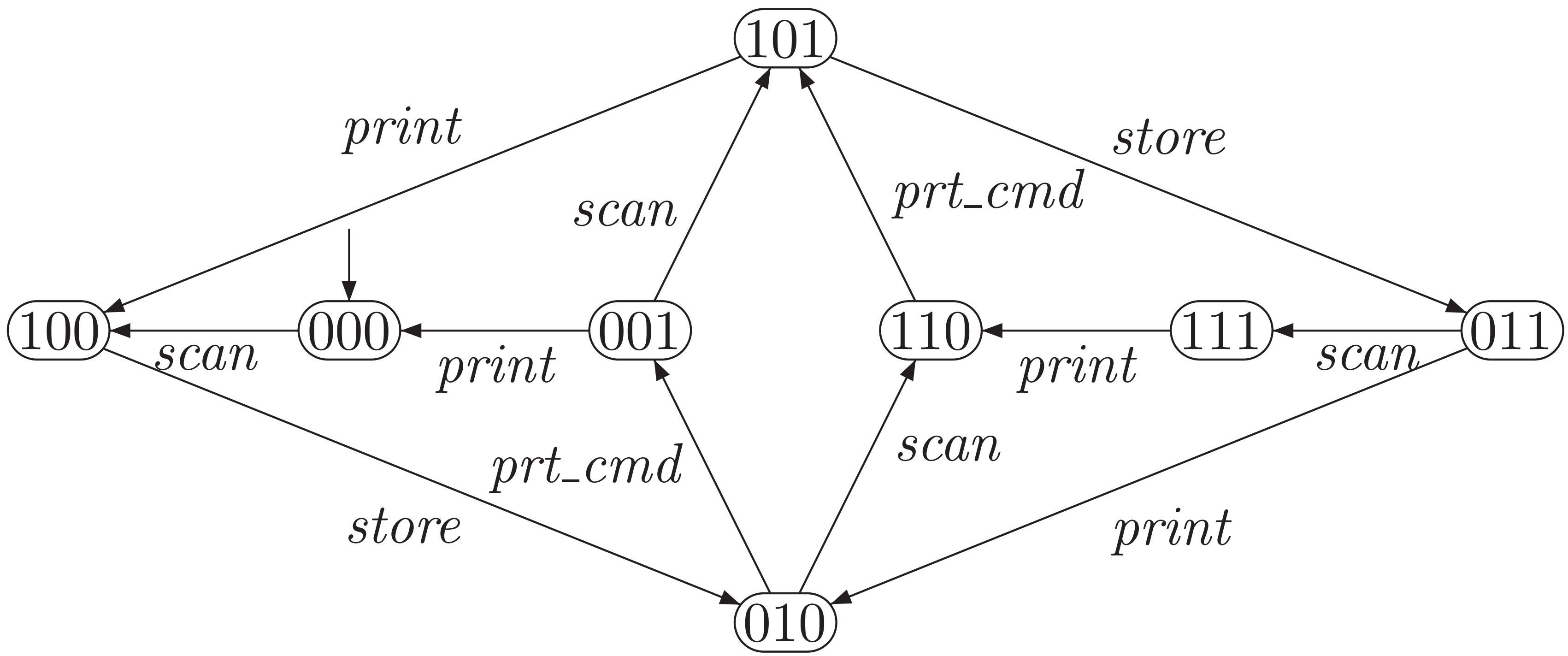
- Three processes:
  - The bar code reader :BCR
  - The actual booking program :BP
  - The printer: Printer



BCR BP Printer

Example: booking system(cont')

- The complete system is given:
  - BCR || BP || Printer

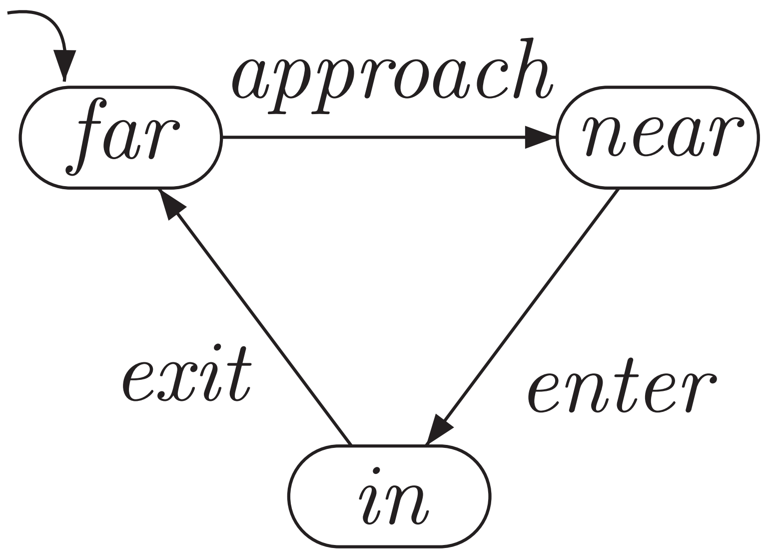


Example railroad crossing

- Requirement:
  - a train is approaching closes the gates, and only opens these gates after the train crossed the road.
  - The gates are always closed when the train is crossing the road.
- Three components: Train, Gate and Controller
  - Train || Gate || Controller

Example railroad crossing(cont')

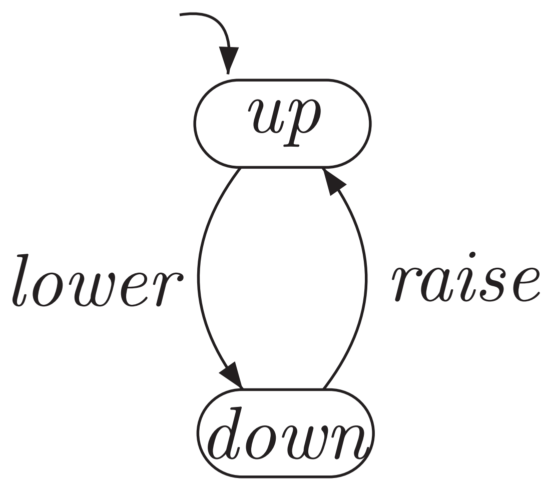
- Train: has the same direction



Train

Example railroad crossing(cont')

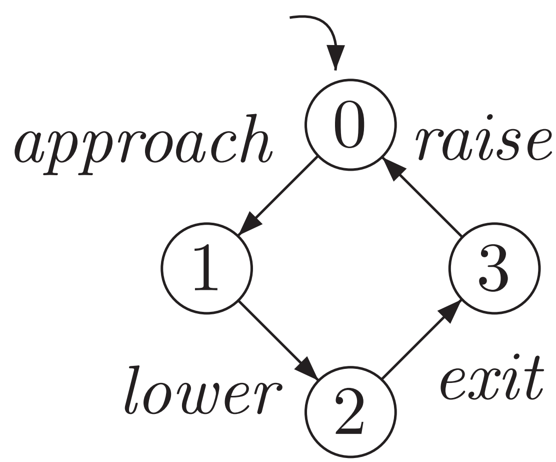
- Gate



Gate

Example railroad crossing(cont')

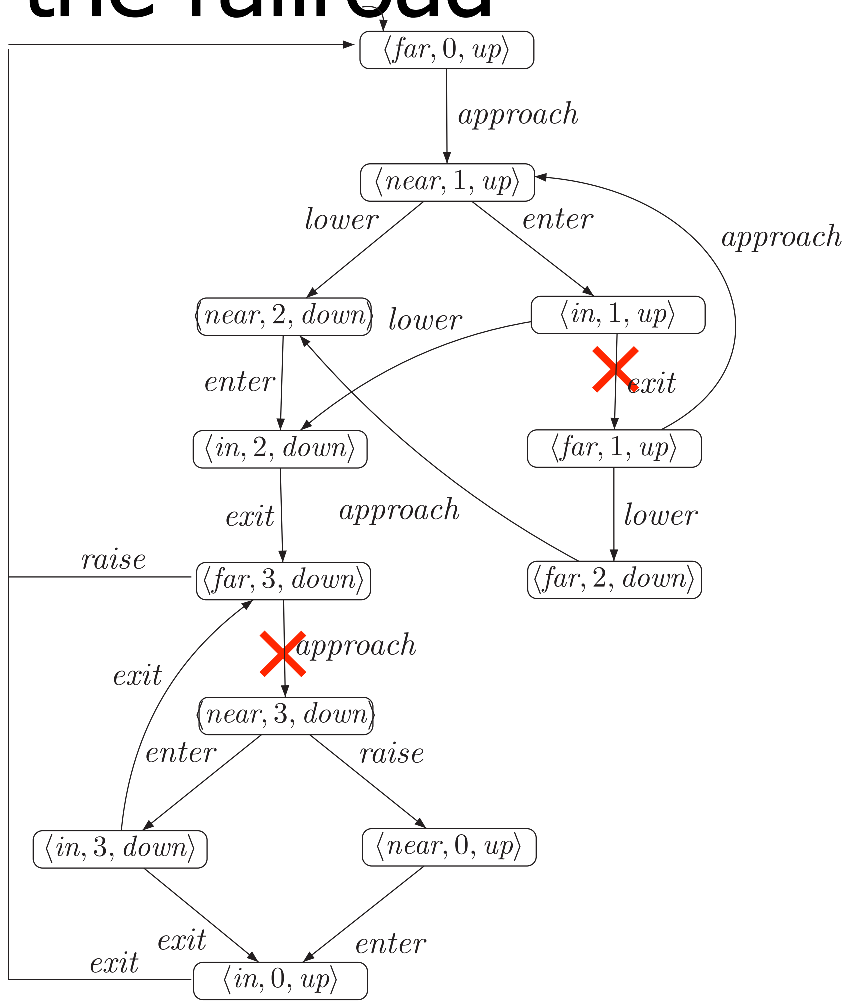
- Controller: The state changes stand for handshaking with the trains and the Gate (the Controller causes the gate to close or to open, respectively).



Controller

Example railroad crossing(cont')

- TS for the railroad



Channel systems

- Channels: first-in, first-out buffers that may contain messages.
- Constraint: channel systems that are closed.
- PG_i represents a processes from P_1 to P_n.
  - Transition:
    - conditional
    - communicative action’s transition.
      - c!v transmit the value v along channel c,
      - c?x receive a message via channel c and assign it to variable x.

Type

- Comm $=\{c!v, c?x \mid c \in$ Chan, $v \in$ dom(c), $x \in$ Var with dom(x) $\supseteq$ dom(c) $\}$
  - Where Chan is a finite set of channels with typical element c.

Capacity

- Channel has a (finite or infinite) capacity cap(c) $\in 2^{\mathbb{N}} \cup \{\infty\}$
  - cap(c)= ∞
  - cap(c)=0: handshaking (simultaneous transmission) plus the exchange of some data.
  - cap(c)>0:sending and reading of the same message take place at different moments. This is called asynchronous message passing.

Channel system (CS)

A program graph over $(Var, Chan)$ is a tuple

$$PG = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$$

$$\hookrightarrow \quad \subseteq \quad Loc \times (Cond(Var) \times (Act \cup Comm) \times Loc.$$

A channel system CS over (*Var*, *Chan*) consists of program graphs PG_i over (Var_i, *Chan*) (for $1 \leq i \leq n$) with $Var = \bigcup_{1 \leq i \leq n} Var_i$. We denote $CS = [PG_1 \mid \ldots \mid PG_n]$.

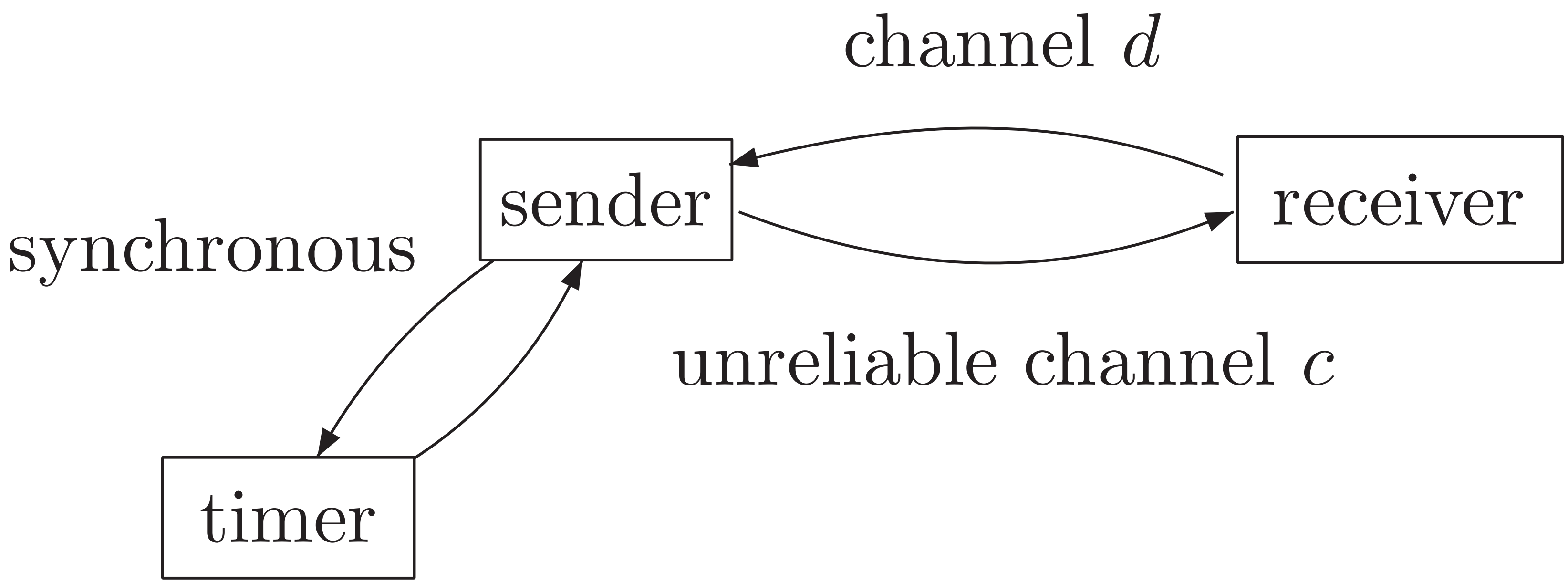
Transition of CS

- Conditional transition: $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$
- Conditional transition with communication actions
  - $\ell \overset{g:c!v}{\hookrightarrow} \ell'$:for sending v along c
  - $\ell \overset{g:c?x}{\hookrightarrow} \ell'$:for receiving a message along c
    - Handshaking :cap(c)=0
    - Asynchronous message passing: cap(c)≠0
      - Send a message to a channel iff c is not full
      - receiving a message from a channel iff c is not

Transition of CS(cont)

	executable if ...	effect
$c!v$	c is not "full"	$Enqueue(c, v)$
$c?x$	c is not empty	$\langle x := Front(c)\,;\; Dequeue(c)\rangle;$

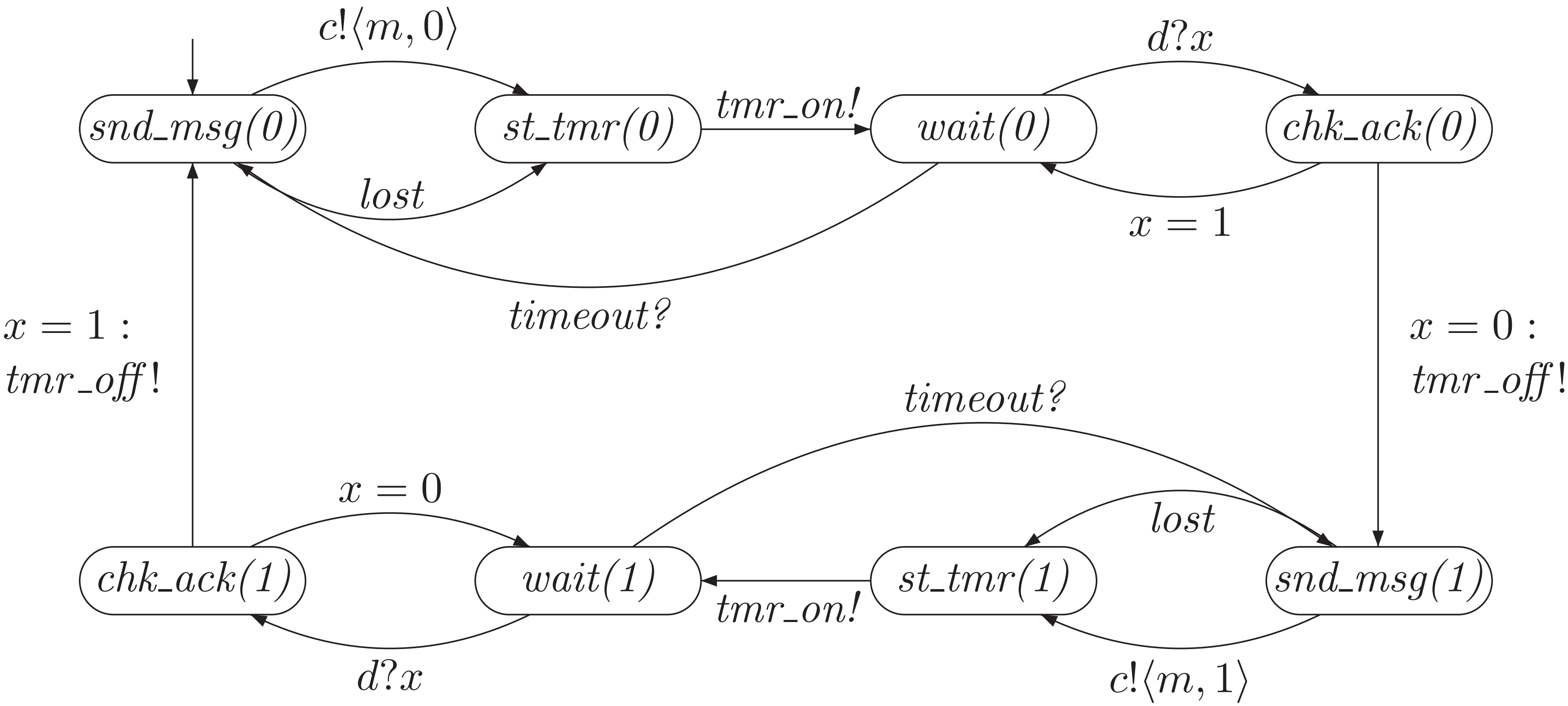
Example: Alternating Bit Protocol



Example: Alternating Bit Protocol (ABP) cont

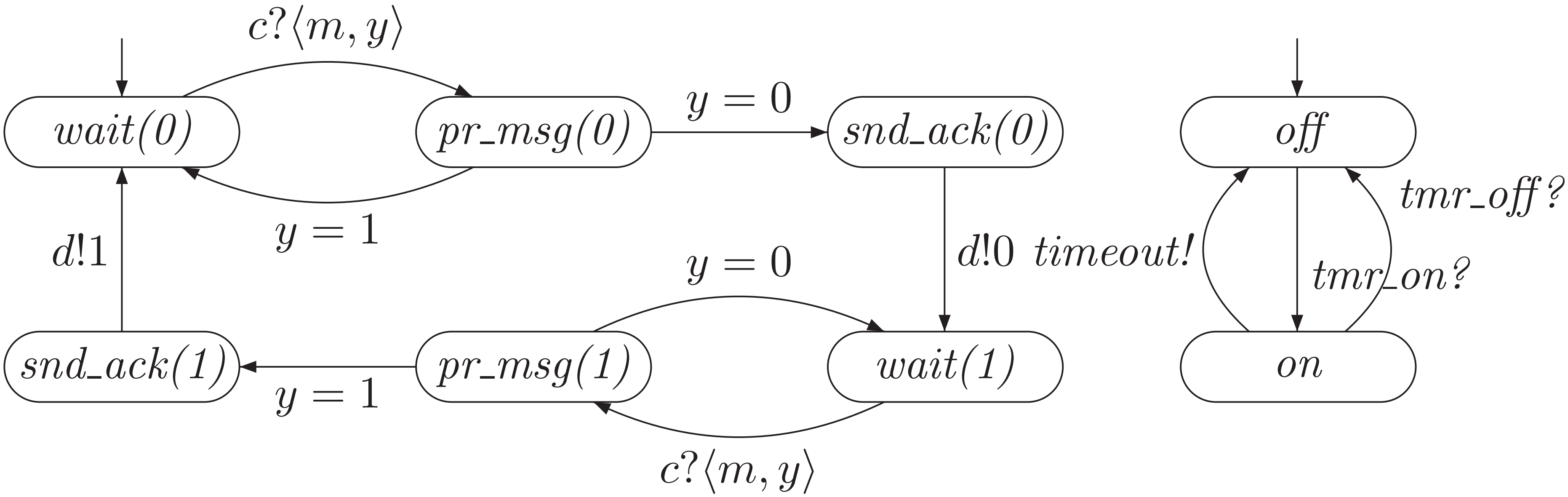
- S sends the successive messages m_0,m_1, . . . together with control bits b_0, b_1, . . . over channel c to R.
  - $<m_0, 0>$, $<m_1, 1>$, $<m_2, 0>$, $<m_3, 1>$, . . .

Example: Alternating Bit Protocol



Program graph of ABP sender S.

Example: Alternating Bit Protocol



Program graph of ABP receiver R

Timer.

Example: Alternating Bit Protocol (ABP) cont

- The complete system :
  - Chan={ c, d, tmr_on, tmr_off , timeout } and Var = { x, y,m_i }
  - ABP = [S | Timer | R]

TS of a channel system(cont')

- Term:
  - $\xi(c) = v_1\, v_2\, .\ .\ .\ v_k$ (where cap(c)≥k), len($\xi(c)$) = k
  - $\xi[c := v_1 \ldots v_k](c') = \begin{cases} \xi(c') & \text{if } c \neq c' \\ v_1 \ldots v_k & \text{if } c = c'. \end{cases}$
  - $\xi_0(c)$ =ε for any channel c. Let len(ε) = 0.
- Actions: consists of actions $\alpha \in Act_i$ of component PG_i and the distinguished symbol τ representing all communication actions in which data is exchanged.

TS of a channel system

- TS(CS)
  - States of TS(CS):
    - $<l_1, \ldots, l_n, \eta, \xi>$
    - η keeps track of the current values of the variables, and
    - ξ records the current content of the various channels
    - Initial $l_i \in Loc_{0,i}$ must be initial and variable evaluation η must satisfy the initial condition $g_{0,i}$.
    - every channel is initially assumed to be empty, denoted ε.

TS of a channel system(cont')

Let $CS = [PG_1 \mid \ldots \mid PG_n]$ be a channel system over $(Chan, Var)$ with

$$PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i}), \qquad \text{for } 0 < i \leqslant n.$$

The transition system of CS, denoted $TS(CS)$, is the tuple $(S, Act, \rightarrow, I, AP, L)$ where:

- $S \;=\; (Loc_1 \times \ldots \times Loc_n) \times Eval(Var) \times Eval(Chan)$,
- $Act \;=\; \biguplus_{0<i\leqslant n} Act_i \;\uplus\; \{\tau\}$,
- $\rightarrow$ is defined by the rules of Figure 2.20 (page 61),
- $I \;=\; \left\{ \langle \ell_1, \ldots, \ell_n, \eta, \xi_0 \rangle \;\middle|\; \forall 0 < i \leqslant n.\ (\ell_i \in Loc_{0,i} \;\wedge\; \eta \models g_{0,i}) \right\}$,
- $AP \;=\biguplus_{0<i\leqslant n} Loc_i \;\uplus\; Cond(Var)$,
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle) \;=\; \{\ell_1, \ldots, \ell_n\} \;\cup\; \{ g \in Cond(Var) \mid \eta \models g \}$.

TS of a channel system(cont')

- interleaving for $\alpha \in Act_i$:

$$\frac{\ell_i \overset{g:\alpha}{\hookrightarrow} \ell_i' \quad \wedge \quad \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi \rangle}$$

  where $\eta' = \mathit{Effect}(\alpha, \eta)$.

- asynchronous message passing for $c \in \mathit{Chan}$, $\mathit{cap}(c) > 0$:

  – receive a value along channel c and assign it to variable x:

$$\frac{\ell_i \overset{g:c?x}{\hookrightarrow} \ell_i' \ \wedge\ \eta \models g \ \wedge\ \mathit{len}(\xi(c)) = k > 0 \ \wedge\ \xi(c) \ = \ v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle}$$

  where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \ldots v_k]$.

  – transmit value $v \in \mathit{dom}(c)$ over channel c:

$$\frac{\ell_i \overset{g:c!v}{\hookrightarrow} \ell_i' \ \wedge\ \eta \models g \ \wedge\ \mathit{len}(\xi(c)) = k < \mathit{cap}(c) \ \wedge\ \xi(c) \ = \ v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle}$$

  where $\xi' = \xi[c := v_1\, v_2 \ldots v_k\, v]$.

- synchronous message passing over $c \in \mathit{Chan}$, $\mathit{cap}(c) = 0$:

$$\frac{\ell_i \overset{g_1:c?x}{\hookrightarrow} \ell_i' \ \wedge\ \eta \models g_1 \ \wedge\ \eta \models g_2 \ \wedge\ \ell_j \overset{g_2:c!v}{\hookrightarrow} \ell_j' \ \wedge\ i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle}$$

  where $\eta' = \eta[x := v]$.

NanoPromela

- the core input language for the SPIN
- support
  - communication by shared variables
  - message passing along either synchronous or buffer FIFO-channels
- the formal semantics can be provided by means of a *channel system*.
- channel system can be unfolded into transition system.
- no use action name,but specify the effect of action

syntax of nanoPromela

$$
\begin{array}{lll}
\mathsf{stmt} & ::= & \texttt{skip} \;\mid\; x := \mathsf{expr} \;\mid\; c?x \;\mid\; c!\mathsf{expr} \;\mid \\
& & \mathsf{stmt}_1 \,;\, \mathsf{stmt}_2 \;\mid\; \texttt{atomic}\{\mathsf{assignments}\} \;\mid \\
& & \mathbf{if} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \mathbf{fi} \quad \mid \\
& & \mathbf{do} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \mathbf{do}
\end{array}
$$

peterson's mutual exclusion algorithm

$$
\begin{array}{llll}
\textbf{do} & :: & \text{true} \Rightarrow & \texttt{skip}; \\
 & & & \texttt{atomic}\{b_1 := \text{true}; x := 2\}; \\
 & & & \textbf{if} \;\; :: \; (x = 1) \vee \neg b_2 \;\; \Rightarrow \;\; \mathit{crit}_1 := \text{true} \;\; \textbf{fi} \\
 & & & \texttt{atomic}\{\mathit{crit}_1 := \text{false}; b_1 := \text{false}\} \\
\textbf{od} & & &
\end{array}
$$

Vending Machine

$$
\begin{array}{lll}
\textbf{do} & :: & \text{true} \Rightarrow \\
& & \quad \texttt{skip}; \\
& & \quad \begin{array}{lllll}
\textbf{if} & :: & nsoda > 0 & \Rightarrow & nsoda := nsoda - 1 \\
& :: & nbeer > 0 & \Rightarrow & nbeer := nbeer - 1 \\
& :: & nsoda = nbeer = 0 & \Rightarrow & \texttt{skip}
\end{array} \\
& & \quad \textbf{fi} \\
& :: & \text{true} \Rightarrow \texttt{atomic}\{nbeer := max; nsoda := max\} \\
\textbf{od} & &
\end{array}
$$

Synchronous Parallelism

Let $TS_i = (S_i, Act, \rightarrow_i, I_i, AP_i, L_i)$, i=1, 2,

be transition systems with the same set of actions Act.

$Act \times Act \rightarrow Act, \quad (\alpha, \beta) \rightarrow \alpha * \beta$

synchronous product $TS_1 \otimes TS_2$ is given by:

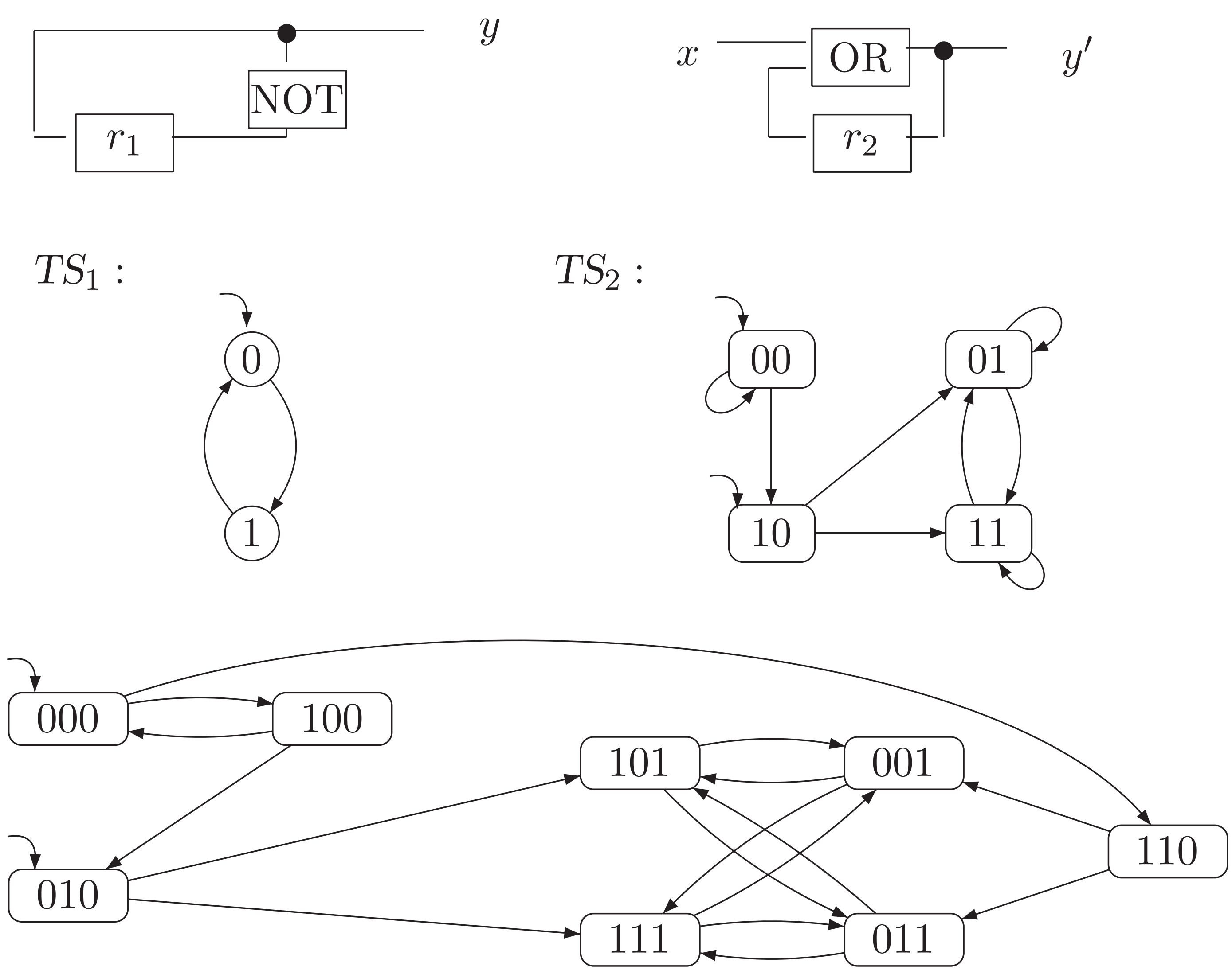
$$TS_1 \otimes TS_2 = (S_1 \times S_2, Act, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L),$$

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2' \rangle}$$

$$L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2).$$

example

- Synchronous Product of Two Circuits



the State-Space Explosion Problem

- Program Graph Representation

$$|\,Loc\,| \cdot \prod_{x \in Var} |\,dom(x)\,| \,.$$

- Channel System

$$\prod_{i=1}^{n} \left(|\,Loc_i\,| \cdot \prod_{x \in Var_i} |\,dom(x)\,| \right) \cdot \prod_{c \in Chan} |\,dom(c)\,|^{cp(c)}.$$

summary

- TS
- PG
- parallel,concurrent
- shared variable system
- handshake
- channel system
- transition to TS