Transition system (TS)

A transition system is a tuple

$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,

Transition system (TS)

A transition system is a tuple

$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,

Transition system (TS)

A transition system is a tuple

$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is the transition relation,

Transition system (TS)

A transition system is a tuple

$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is the transition relation,

i.e., transitions have the form $s \xrightarrow{\alpha} s'$
where $s, s' \in S$ and $\alpha \in Act$

Transition system (TS)

A transition system is a tuple

$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is the transition relation,

  i.e., transitions have the form $s \xrightarrow{\alpha} s'$ where $s, s' \in S$ and $\alpha \in Act$

- $S_0 \subseteq S$ the set of initial states,

Transition system (TS)

A transition system is a tuple

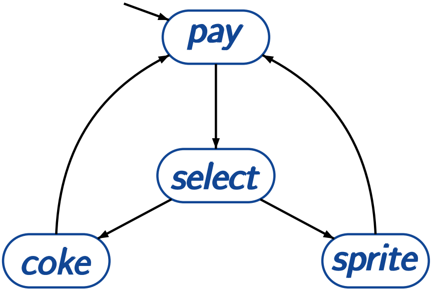
$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is the transition relation,

  i.e., transitions have the form $s \xrightarrow{\alpha} s'$ where $s, s' \in S$ and $\alpha \in Act$

- $S_0 \subseteq S$ the set of initial states,
- AP a set of atomic propositions,
- $L : S \to 2^{AP}$ the labeling function

Transition system for beverage machine



Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

Program graph (PG)

Let ***Var*** be a set of typed variables.

A *program graph* over ***Var*** is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- ***Loc*** is a (finite) set of locations, i.e., control states,

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $Effect : Act \times Eval(Var) \to Eval(Var)$

$\uparrow$

function that formalizes the effect of the actions

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$

$\uparrow$

function that formalizes the effect of the actions

example: if α is the assignment $x:=x+y$ then

$$Effect(\alpha, [x=1, y=7]) = [x=8, y=7]$$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \rightarrow \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \;\subseteq\; \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \subseteq \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

  specifies conditional transitions of the form $\ell \xhookrightarrow{g:\alpha} \ell'$

ℓ, ℓ' are locations, $g \in \mathit{Cond}(\mathit{Var})$, $\alpha \in \mathit{Act}$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \subseteq \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$
  specifies conditional transitions of the form $\ell \xrightarrow{g:\alpha} \ell'$
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is the set of initial locations,

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \;\subseteq\; \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

  specifies conditional transitions of the form $\ell \xhookrightarrow{g:\alpha} \ell'$
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is the set of initial locations,
- $g_0 \in \mathit{Cond}(\mathit{Var})$ initial condition on the variables

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$

  specifies conditional transitions of the form $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$
- $Loc_0 \subseteq Loc$ is the set of initial locations,
- $g_0 \in Cond(Var)$ initial condition on the variables.

TS-semantics of a program graph

TS-semantics of a program graph

$$\boxed{\begin{array}{c}\text{program graph } \mathcal{P} \text{ over } \textit{Var} \\ \Downarrow \\ \text{transition system } \mathcal{T}_{\mathcal{P}}\end{array}}$$

TS-semantics of a program graph

program graph $\mathcal{P}$ over $\boldsymbol{Var}$

$$\Downarrow$$

transition system $\mathcal{T}_{\mathcal{P}}$

states in $\mathcal{T}_{\mathcal{P}}$ have the form

$$\langle \ell, \eta \rangle$$

location   variable evaluation

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \; = \; (S, Act, \longrightarrow, S_0, AP, L)$$

TS-semantics of a program graph

Let $\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \; = \; (S, \mathit{Act}, \longrightarrow, S_0, AP, L)$$

- state space: $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} = (S, Act, \longrightarrow, S_0, AP, L)$$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0\}$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \; = \; (S, Act, \longrightarrow, S_0, AP, L)$$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \big\{ \langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0 \big\}$

The transition relation $\longrightarrow$ is given by the following rule:

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

Structured operational semantics (SOS)

The transition system of a program graph $\mathcal{P}$ is

$$\mathcal{T}_{\mathcal{P}} = (S, Act, \longrightarrow, S_0, AP, L) \text{ where}$$

the transition relation $\longrightarrow$ is given by the following rule

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

is a shortform notation in **SOS**-style.

$$\frac{premise}{conclusion}$$

The transition system of a program graph $\mathcal{P}$ is

$$\mathcal{T}_{\mathcal{P}} = (S, Act, \longrightarrow, S_0, AP, L) \text{ where}$$

the transition relation $\longrightarrow$ is given by the following rule

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

is a shortform notation in **SOS**-style.

It means that $\longrightarrow$ is the smallest relation such that:

$$\boxed{\text{if } \ell \overset{g:\alpha}{\hookrightarrow} \ell' \ \wedge\ \eta \models g \text{ then } \langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} = (S, Act, \longrightarrow, S_0, AP, L)$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0\}$
- $\longrightarrow$ is given by the following rule:

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

Let $\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} = (S, \mathit{Act}, \longrightarrow, S_0, \mathit{AP}, L)$

- state space: $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in \mathit{Loc}_0, \eta \models g_0\}$
- $\longrightarrow$ is given by the following rule:

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

- atomic propositions: $\mathit{AP} = \mathit{Loc} \cup \mathit{Cond}(\mathit{Var})$

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} = (S, Act, \longrightarrow, S_0, AP, L)$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0\}$
- $\longrightarrow$ is given by the following rule:

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

- atomic propositions: $AP = Loc \cup Cond(Var)$
- labeling function:

$$L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{g \in Cond(Var) : \eta \models g\}$$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} \; = \; (S, Act, \longrightarrow, S_0, AP, L)$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0\}$
- $\longrightarrow$ is given by the following rule:

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

- atomic propositions: $AP = Loc \cup Cond(Var)$
- labeling function:

$$L(\langle \ell, \eta \rangle) \; = \; \{\ell\} \cup \{g \in Cond(Var) : \eta \models g\}$$

Parallelism and communication

"real" parallel system

$$P = P_1 \parallel \ldots \parallel P_n$$

$$\uparrow \quad \downarrow$$

transition system

$$\mathcal{T} = \mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$$

Parallelism and communication

"real" parallel system

$$P = P_1 \parallel \ldots \parallel P_n$$

compositional design $\uparrow$ $\downarrow$ analysis semantics

transition system

$$\mathcal{T} = \mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$$

Parallelism and communication

"real" parallel system
$P = P_1 \parallel \ldots \parallel P_n$

compositional design $\uparrow$ $\downarrow$ analysis semantics

transition system
$\mathcal{T} = \mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$

goal: define semantic parallel operators on transition systems or program graphs that model "real" parallel operators

Interleaving operator ||| for TS

- interleaving of concurrent, independent actions of parallel processes (modelled by TS)
- representation by nondeterministic choice:

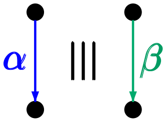
  "which subprocess performs the next step?"

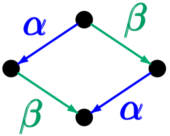
Interleaving operator ||| for TS

- interleaving of concurrent, independent actions of parallel processes (modelled by TS)
- representation by nondeterministic choice:

  "which subprocess performs the next step?"

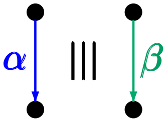
$$\textit{effect}(\alpha|||\beta) \;=\; \textit{effect}(\alpha;\beta+\beta;\alpha)$$

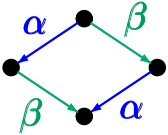




- interleaving of concurrent, independent actions of parallel processes (modelled by TS)
- representation by nondeterministic choice:
  "which subprocess performs the next step?"

$$\mathit{effect}(\alpha|||\beta) \;=\; \mathit{effect}(\alpha;\beta+\beta;\alpha)$$



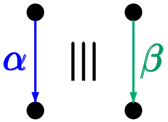


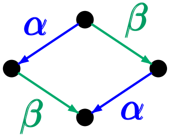
parallel execution of α and β on *two processors*

Interleaving operator ||| for TS

- interleaving of concurrent, independent actions of parallel processes (modelled by TS)
- representation by nondeterministic choice:
  "which subprocess performs the next step?"

$$\mathit{effect}(\alpha|||\beta) \;=\; \mathit{effect}(\alpha;\beta+\beta;\alpha)$$





parallel execution of α and β on *two processors* $\widehat{=}$ serial execution on a *single processor* in arbitrary order

Interleaving operator $|||$ for TS

$$\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, S_{0,1}, AP_1, L_1)$$
$$\mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, S_{0,2}, AP_2, L_2)$$

Interleaving operator ||| for TS

PC2.2-DEF-INTERLEAVING-TS

$$\mathcal{T}_1 \;=\; (S_1, Act_1, \longrightarrow_1, S_{0,1}, AP_1, L_1)$$

$$\mathcal{T}_2 \;=\; (S_2, Act_2, \longrightarrow_2, S_{0,2}, AP_2, L_2)$$

The transition system $\mathcal{T}_1 \,|||\, \mathcal{T}_2$ is defined by:

$$\mathcal{T}_1 \,|||\, \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, S_{0,1} \times S_{0,2}, AP, L)$$

where the transition relation $\longrightarrow$ is given by:

Interleaving operator $|||$ for TS

$$\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, S_{0,1}, AP_1, L_1)$$

$$\mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, S_{0,2}, AP_2, L_2)$$

The transition system $\mathcal{T}_1 \,|||\, \mathcal{T}_2$ is defined by:

$$\mathcal{T}_1 \,|||\, \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, S_{0,1} \times S_{0,2}, AP, L)$$

where the transition relation $\longrightarrow$ is given by:

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

Interleaving operator ||| for TS

$$\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, S_{0,1}, AP_1, L_1)$$

$$\mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, S_{0,2}, AP_2, L_2)$$

The transition system $\mathcal{T}_1 \,|||\, \mathcal{T}_2$ is defined by:

$$\mathcal{T}_1 \,|||\, \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, S_{0,1} \times S_{0,2}, AP, L)$$

where the transition relation $\longrightarrow$ is given by:

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

atomic propositions: $AP = AP_1 \uplus AP_2$

labeling function: $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$

SOS-notation (structured operational semantics)

just a simple notation for operational semantics

$$\frac{\textbf{premise}}{\textbf{conclusion}}$$

SOS-notation (structured operational semantics)

just a simple notation for operational semantics

$$\frac{\textbf{premise}}{\textbf{conclusion}}$$

E.g., *"the relation $\longrightarrow$ is given by ..."*

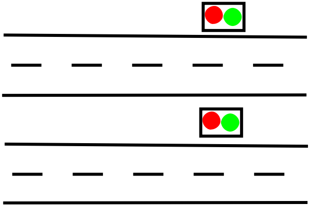
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

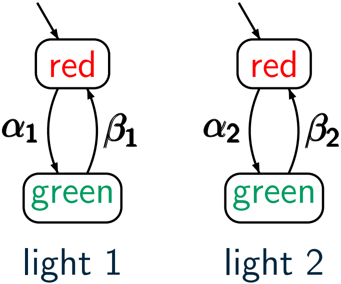
means that $\longrightarrow$ is the smallest relation such that:

(1) If $s_1 \xrightarrow{\alpha}_1 s_1'$, then $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle$

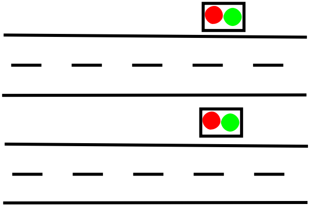
(2) If $s_2 \xrightarrow{\alpha}_2 s_2'$, then $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle$

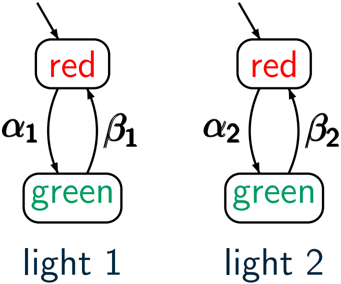
Useless lights for non-crossing streets



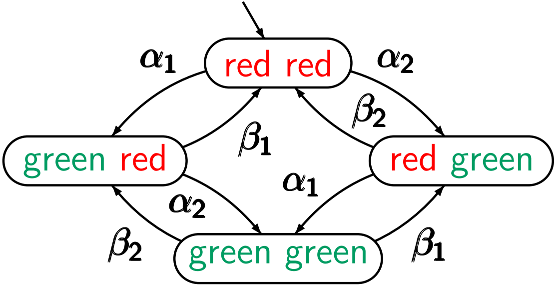


Useless lights for non-crossing streets





light 1 ||| light 2



Dependent actions

dependent actions $\alpha \mathrel{\widehat{=}} x{:=}2x$ and $\beta \mathrel{\widehat{=}} x{:=}x{+}1$

representations in transition systems

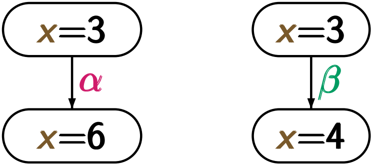
$$\begin{array}{c} x{=}3 \\ \downarrow \alpha \\ x{=}6 \end{array} \qquad\qquad \begin{array}{c} x{=}3 \\ \downarrow \beta \\ x{=}4 \end{array}$$

Dependent actions

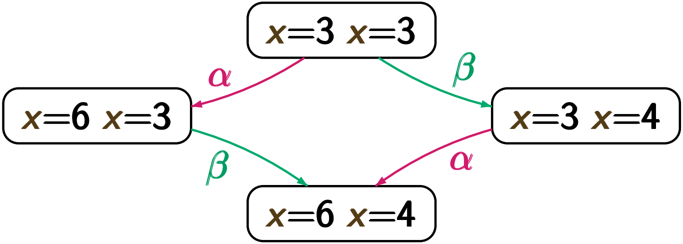
PC2.2-5

dependent actions $\alpha \mathrel{\widehat{=}} x{:=}2x$ and $\beta \mathrel{\widehat{=}} x{:=}x{+}1$

representations in transition systems



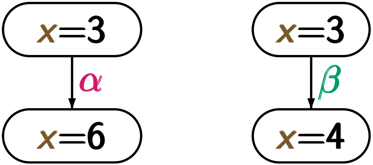
interleaving operator $|||$



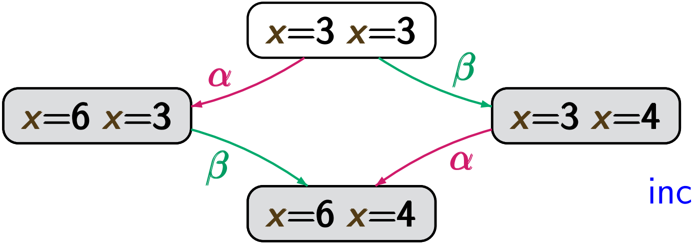
Interleaving fails for dependent actions

dependent actions $\alpha \mathrel{\widehat{=}} x{:=}2x$ and $\beta \mathrel{\widehat{=}} x{:=}x{+}1$

representations in transition systems



interleaving operator $|||$ for transition systems "fails"



inconsistent states

Interleaving for program graphs

. . . for modeling parallel systems with subprocesses communicating via shared variables

Interleaving for program graphs

program graph $\mathcal{P}_1$
$(Loc_1, \ldots, \hookrightarrow_1, \ldots)$

program graph $\mathcal{P}_2$
$(Loc_2, \ldots, \hookrightarrow_2, \ldots)$

interleaving operator

$$\mathcal{P}_1 \,|||\, \mathcal{P}_2 \;=\; (Loc_1 \times Loc_2, \ldots, \hookrightarrow, \ldots)$$

Interleaving for program graphs

program graph $\mathcal{P}_1$
$(Loc_1, \ldots, \hookrightarrow_1, \ldots)$

program graph $\mathcal{P}_2$
$(Loc_2, \ldots, \hookrightarrow_2, \ldots)$

interleaving operator

$$\mathcal{P}_1 \,|||\, \mathcal{P}_2 \;=\; (Loc_1 \times Loc_2, \ldots, \hookrightarrow, \ldots)$$

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

Example: interleaving for PG

$\mathcal{P}_1$

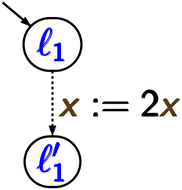
$$\begin{tikzcd} \ell_1 \arrow[d, dotted, "x := 2x"] \\ \ell_1' \end{tikzcd}$$

$\mathcal{P}_2$

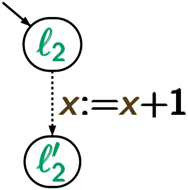
$$\begin{tikzcd} \ell_2 \arrow[d, dotted, "x := x+1"] \\ \ell_2' \end{tikzcd}$$

Example: interleaving for PG

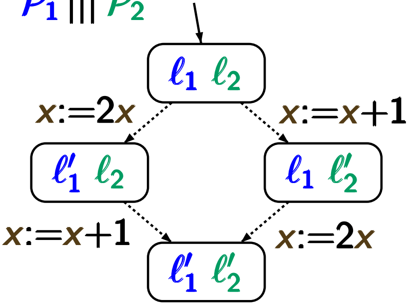
$\mathcal{P}_1$



$\mathcal{P}_2$

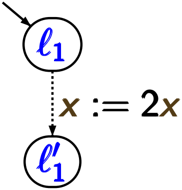


$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

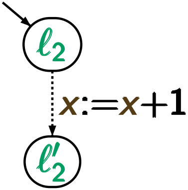


Example: interleaving for PG

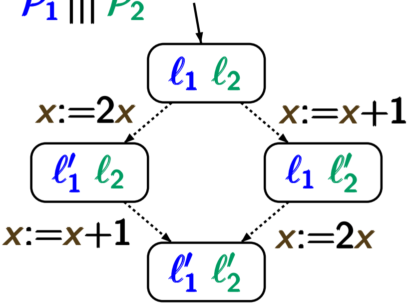
$\mathcal{P}_1$



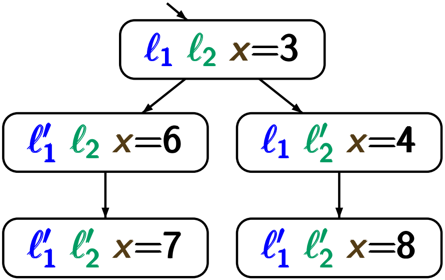
$\mathcal{P}_2$



$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

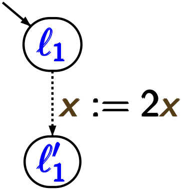


transition system $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2}$

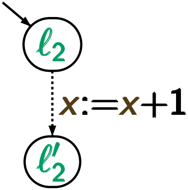


Example: interleaving for PG

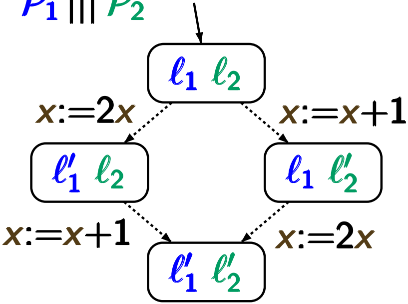
$\mathcal{P}_1$



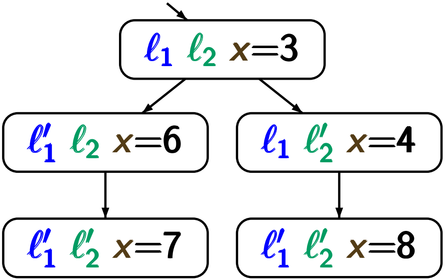
$\mathcal{P}_2$



$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

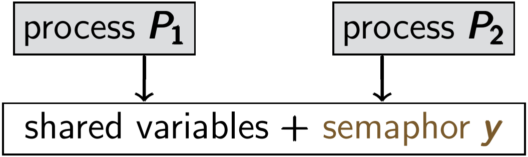


transition system $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2}$



note: $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2} \neq \mathcal{T}_{\mathcal{P}_1} \,|||\, \mathcal{T}_{\mathcal{P}_2}$

Mutual exclusion with semaphore



Mutual exclusion with semaphore

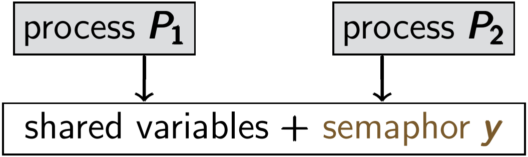
process P_1 process P_2

shared variables + semaphor y

protocol for process P_i

```
LOOP FOREVER
  noncritical actions;
  AWAIT y > 0 DO
          y:=y−1
  OD
  critical actions;
  y:=y+1
END LOOP
```

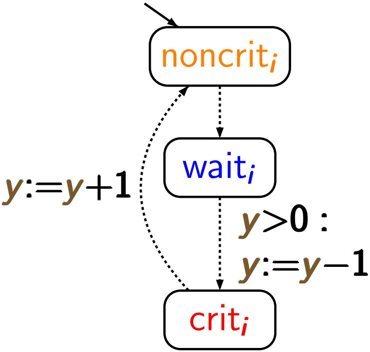
Mutual exclusion with semaphore



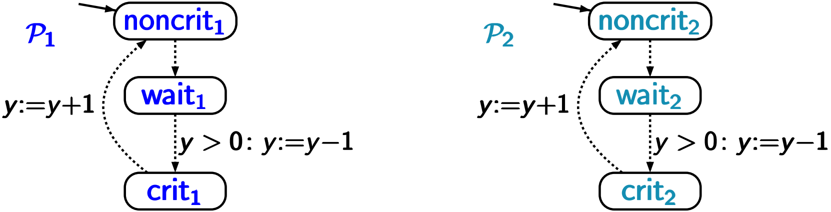
protocol for process P_i

```
LOOP FOREVER
  noncritical actions;
  AWAIT y > 0 DO
          y:=y−1
  OD
  critical actions;
  y:=y+1
END LOOP
```

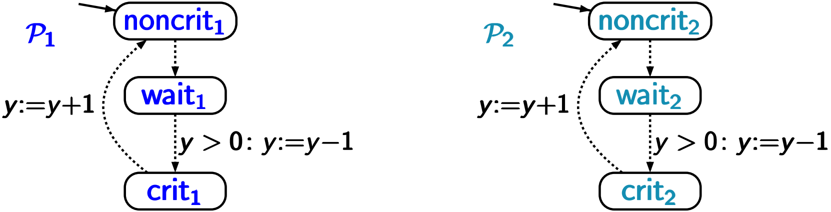
program graph $\mathcal{P}_i$



Mutual exclusion with semaphore

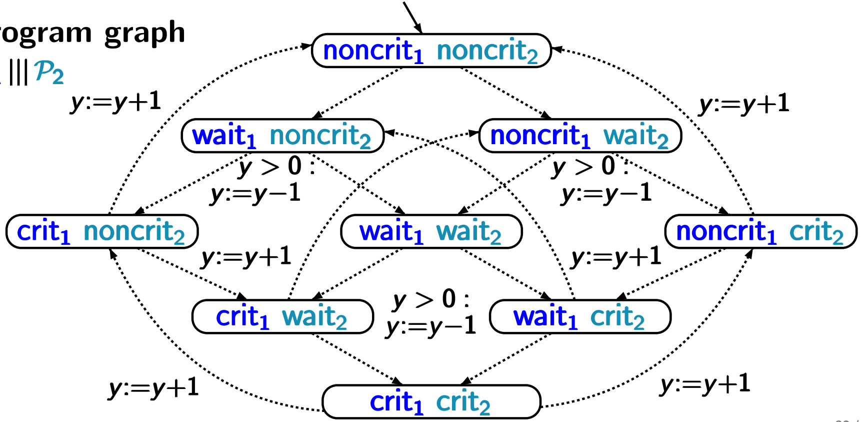


Mutual exclusion with semaphore

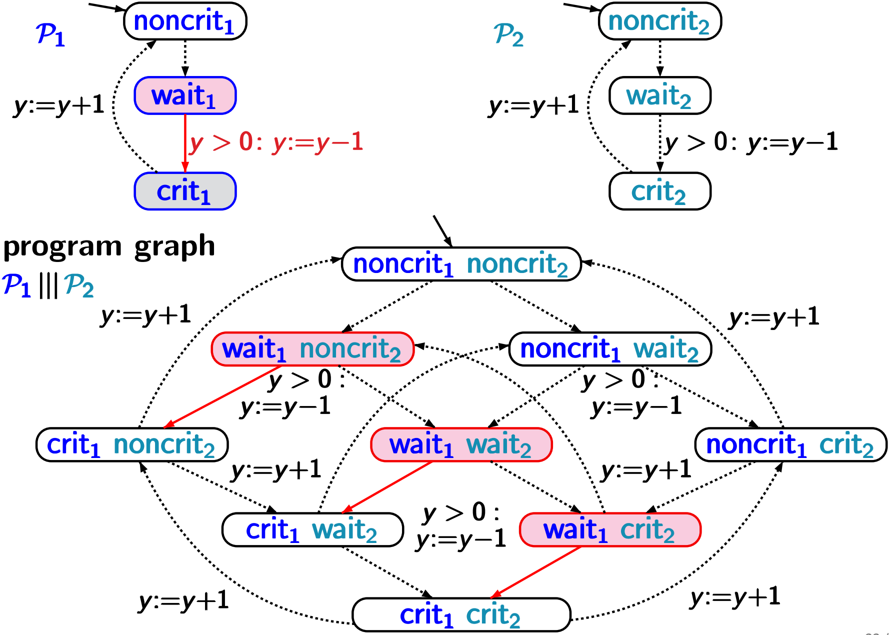


program graph

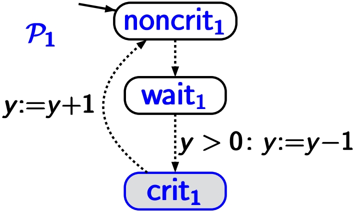
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

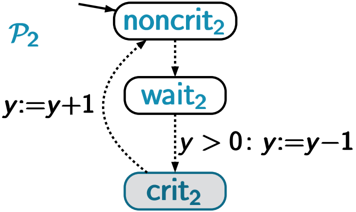


Mutual exclusion with semaphore



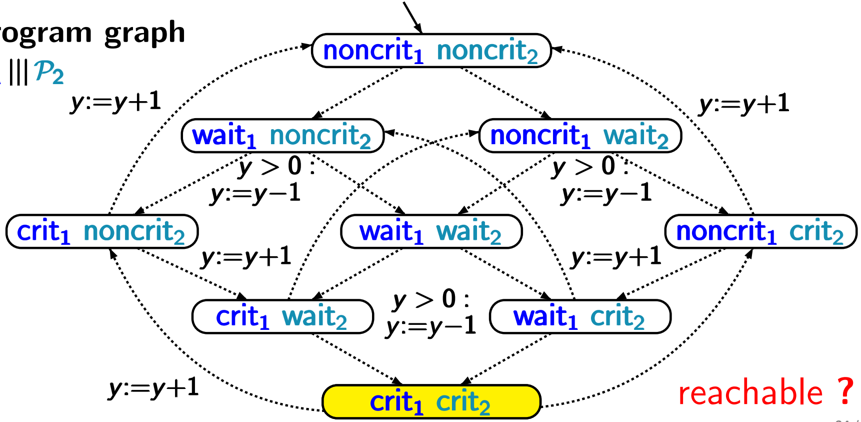
Mutual exclusion with semaphore





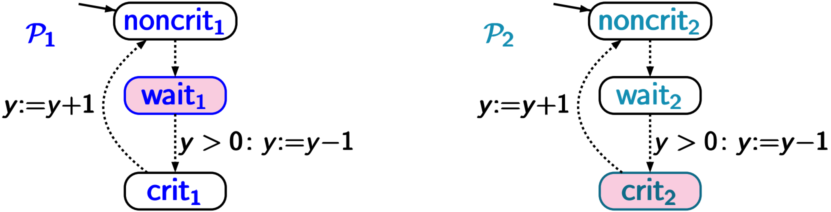
program graph

$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

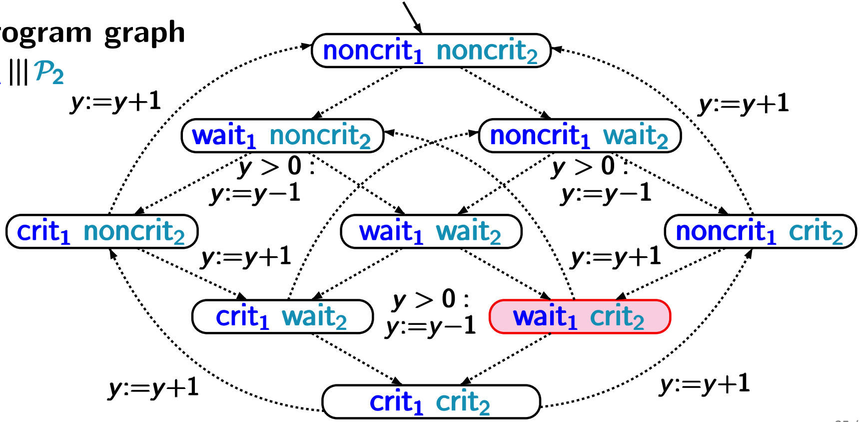


reachable **?**

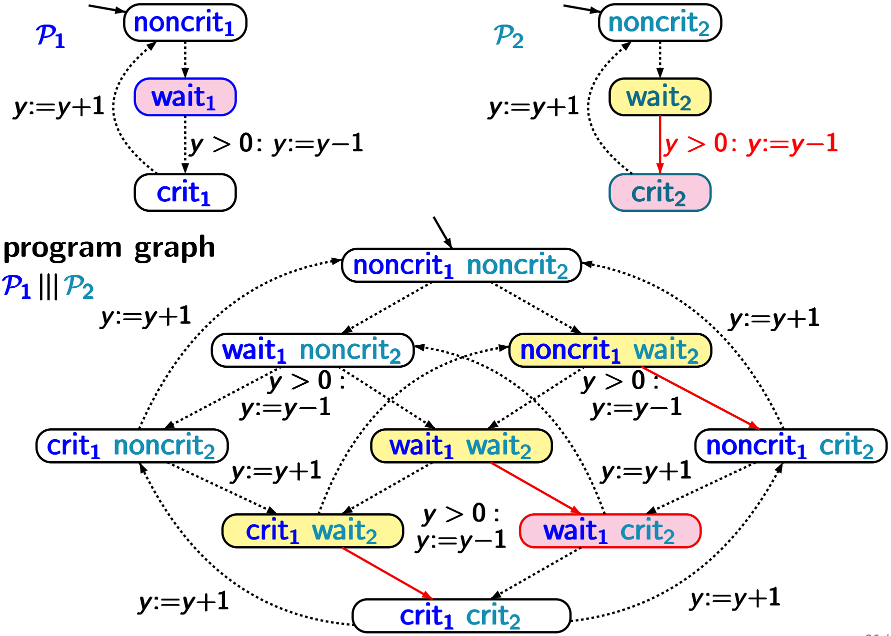
Mutual exclusion with semaphore



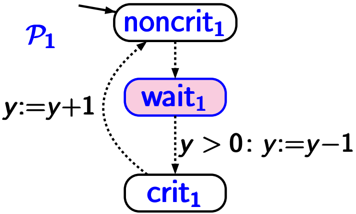
program graph
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

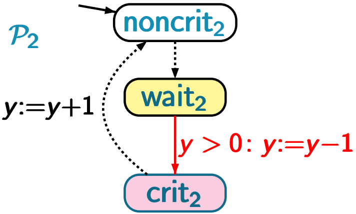


Mutual exclusion with semaphore

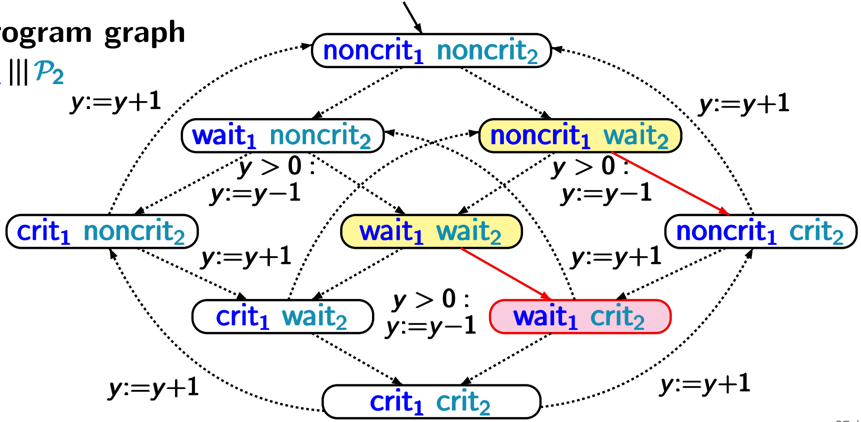


Mutual exclusion with semaphore

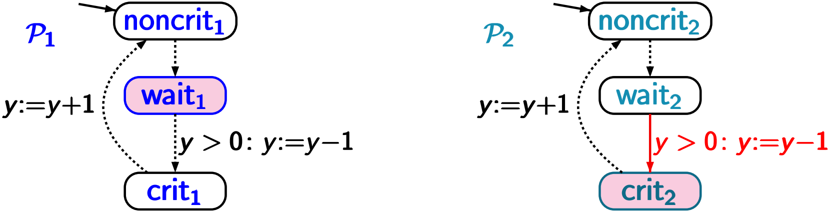




program graph
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$

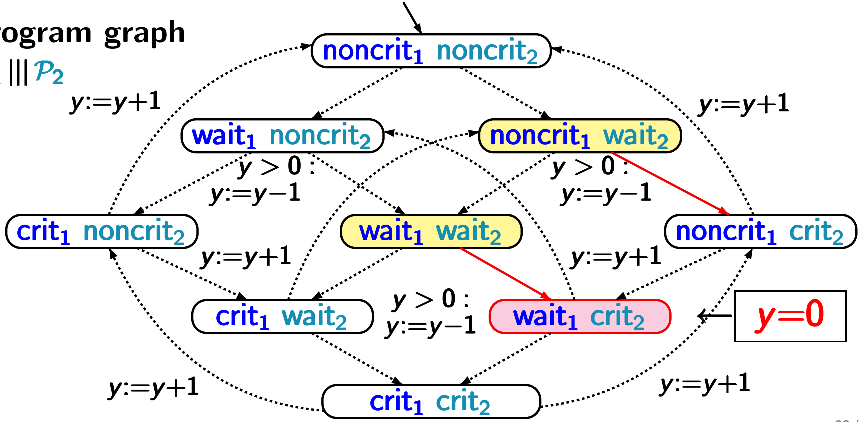


Mutual exclusion with semaphore

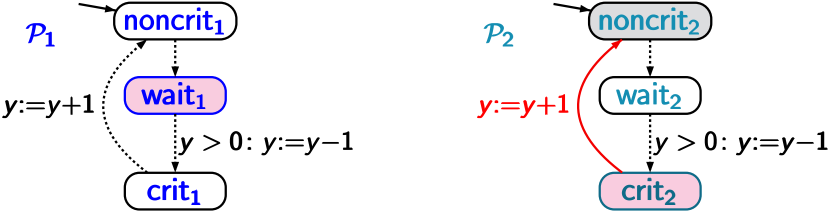


program graph

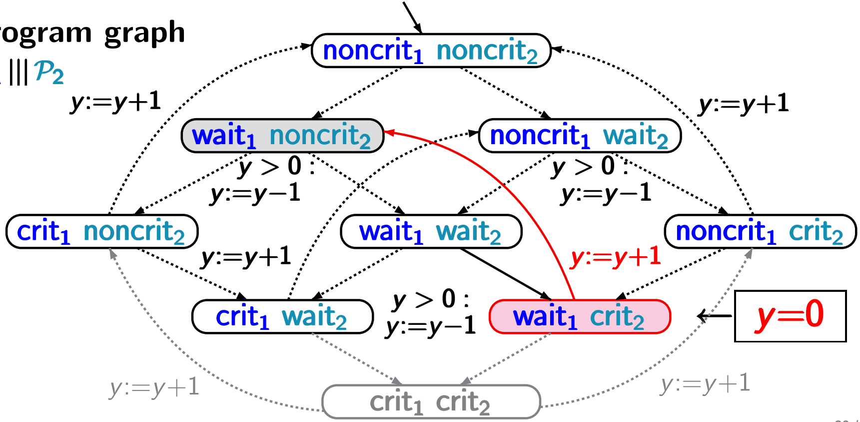
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$



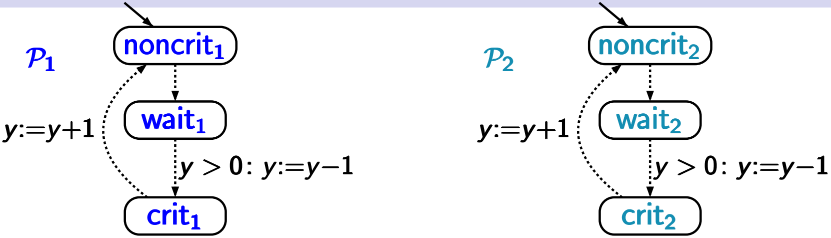
Mutual exclusion with semaphore



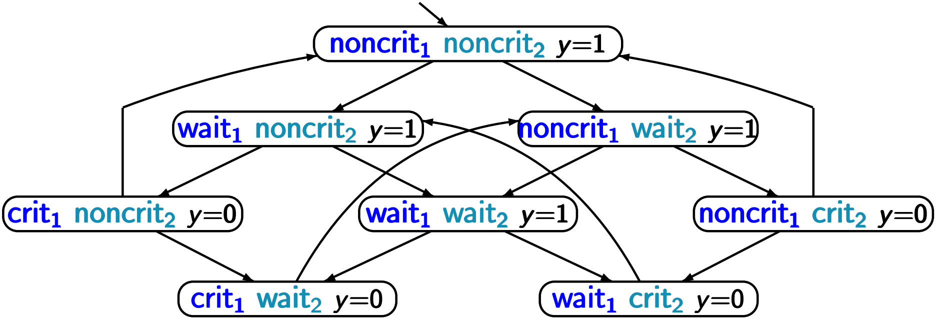
program graph
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$



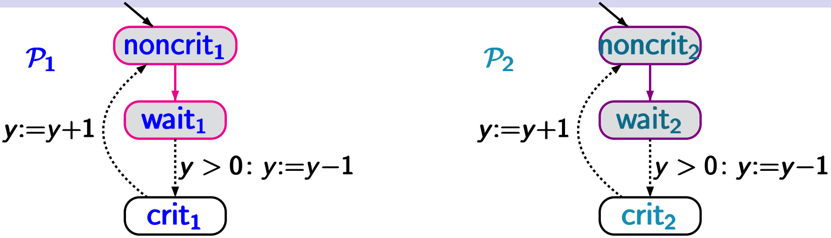
TS for mutual exclusion with semaphore



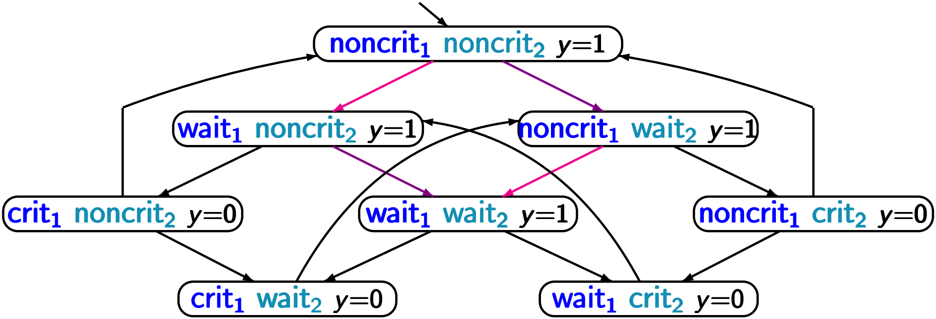
reachable fragment of the transition system $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2}$



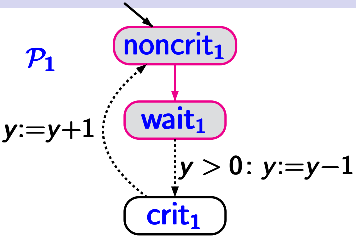
Concurrency of the request actions

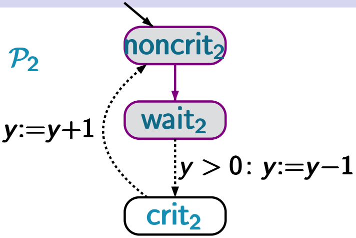


reachable fragment of the transition system $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2}$

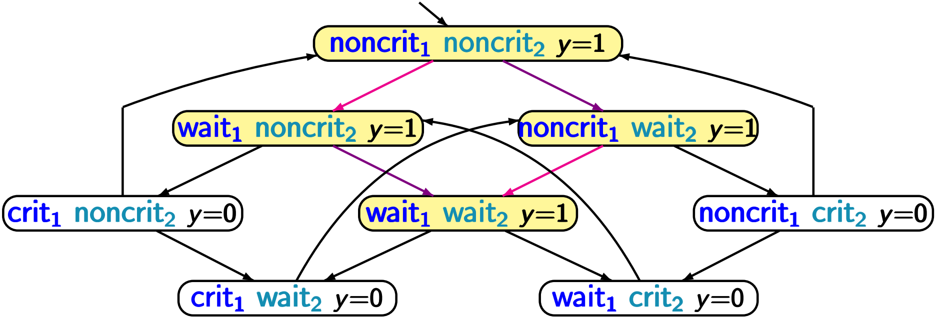


Concurrency of the request actions

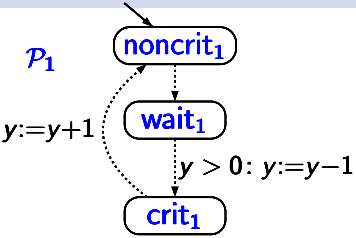


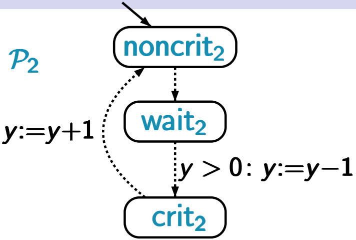


interleaving of the independent request actions

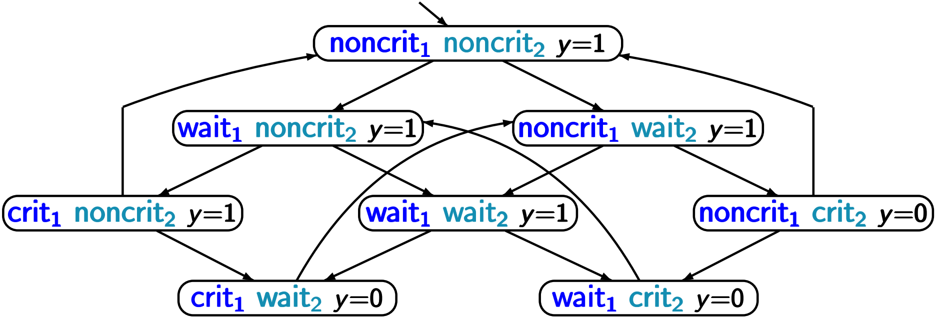


Competition

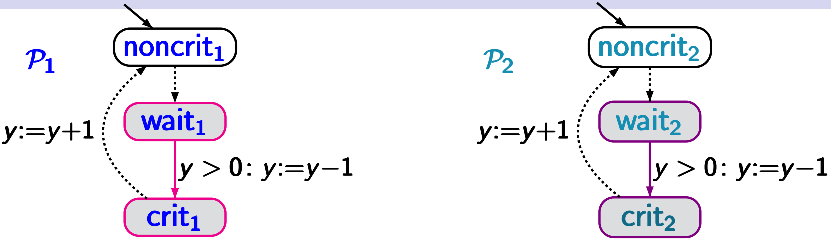




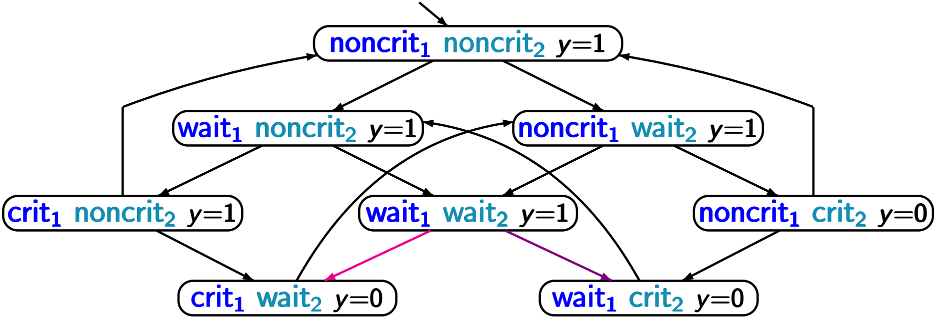
reachable fragment of the transition system $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2}$



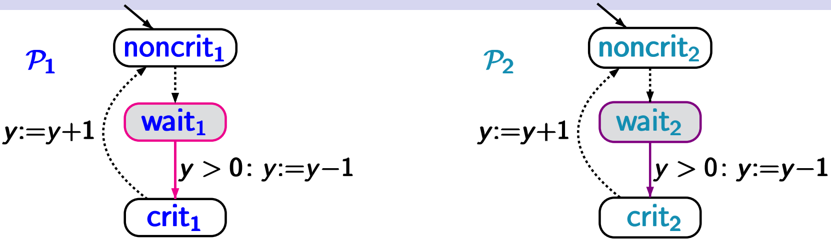
Competition



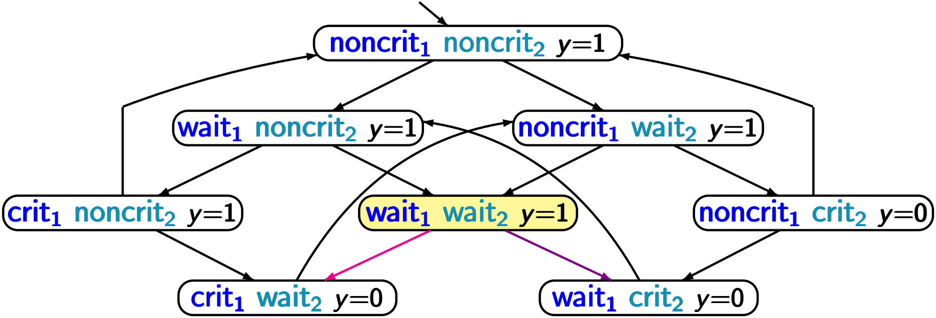
reachable fragment of the transition system $\mathcal{T}_{\mathcal{P}_1 \,|||\, \mathcal{P}_2}$



Competition

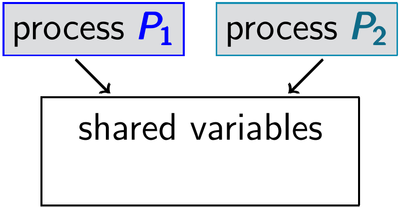


. . . competition between the waiting processes . . .

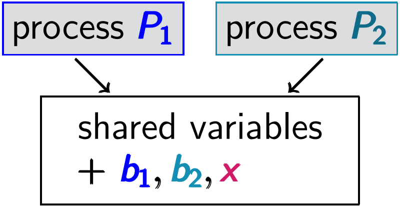


Peterson algorithm for mutual exclusion

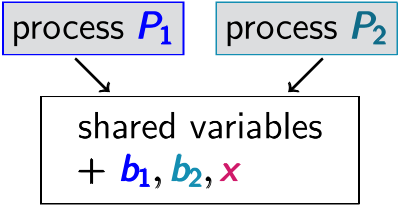
Peterson algorithm for mutual exclusion



Peterson algorithm for mutual exclusion

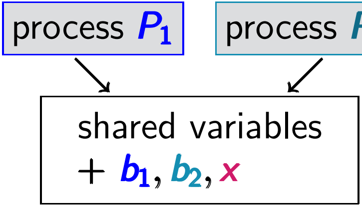


Peterson algorithm for mutual exclusion



b_1, b_2 Boolean variables, $x \in \{1, 2\}$

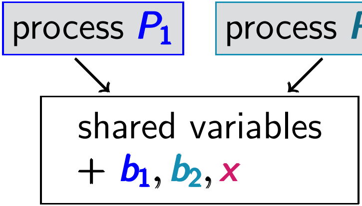
Peterson algorithm for mutual exclusion

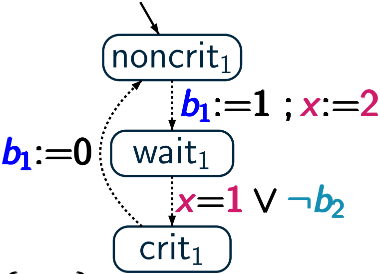


b_1, b_2 Boolean variables, $x \in \{1, 2\}$

```
LOOP FOREVER                          (* protocol for P1 *)
    noncritical actions;
    b1:=1 ; x:=2;
    AWAIT  x=1 ∨ ¬b2  DO  critical section OD
    b1:=0
END LOOP
```

Peterson algorithm for mutual exclusion

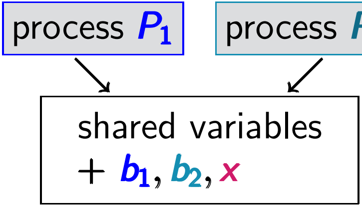


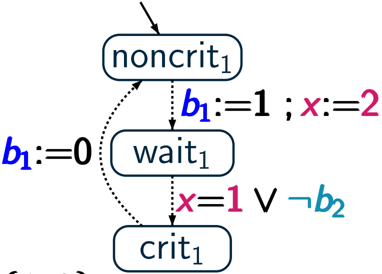


b_1, b_2 Boolean variables, $x \in \{1, 2\}$

```
LOOP FOREVER                        (* protocol for P1 *)
    noncritical actions;
    b1:=1 ; x:=2;
    AWAIT x=1 ∨ ¬b2 DO critical section OD
    b1:=0
END LOOP
```

Peterson algorithm for mutual exclusion





b_1, b_2 Boolean variables, $x \in \{1, 2\}$

LOOP FOREVER (* protocol for P_1 *)

  noncritical actions;

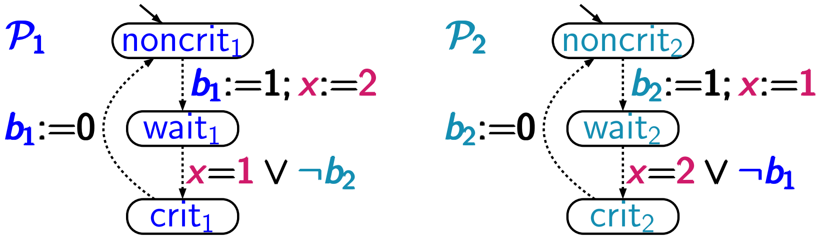
  atomic$\{b_1 := 1 \; ; \; x := 2\}$;

  AWAIT $x = 1 \vee \neg b_2$ DO critical section OD

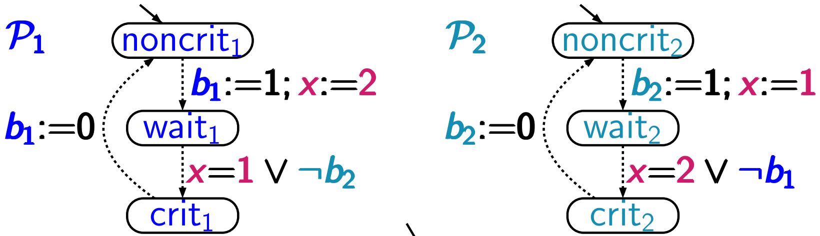
  $b_1 := 0$

END LOOP

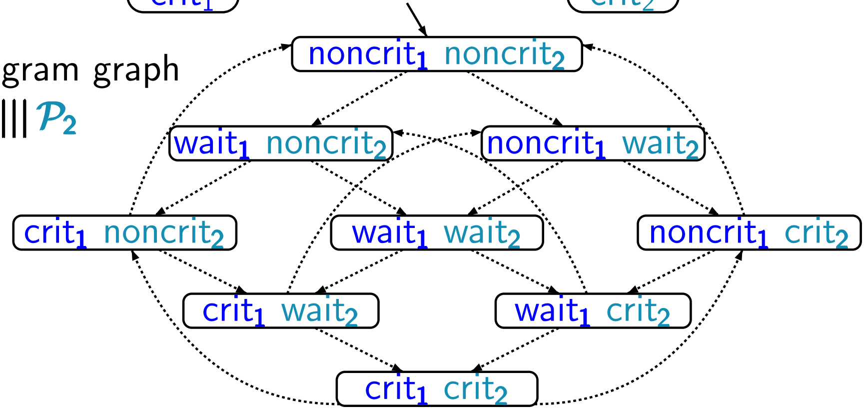
Program graphs for Peterson algorithm



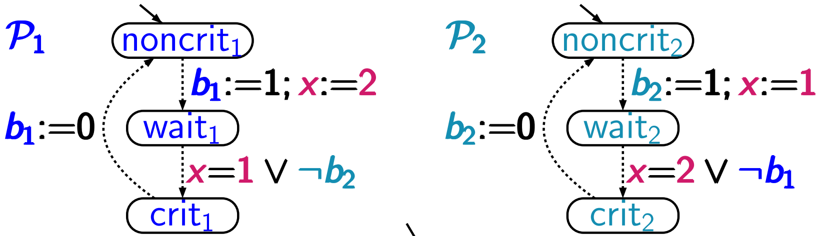
Program graphs for Peterson algorithm



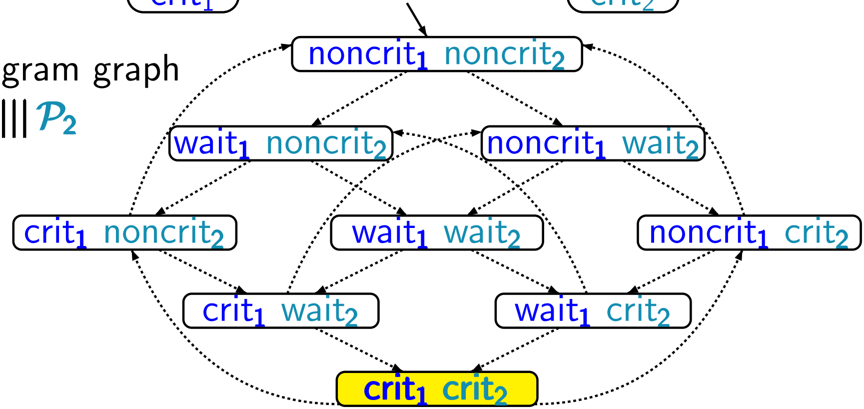
program graph
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$



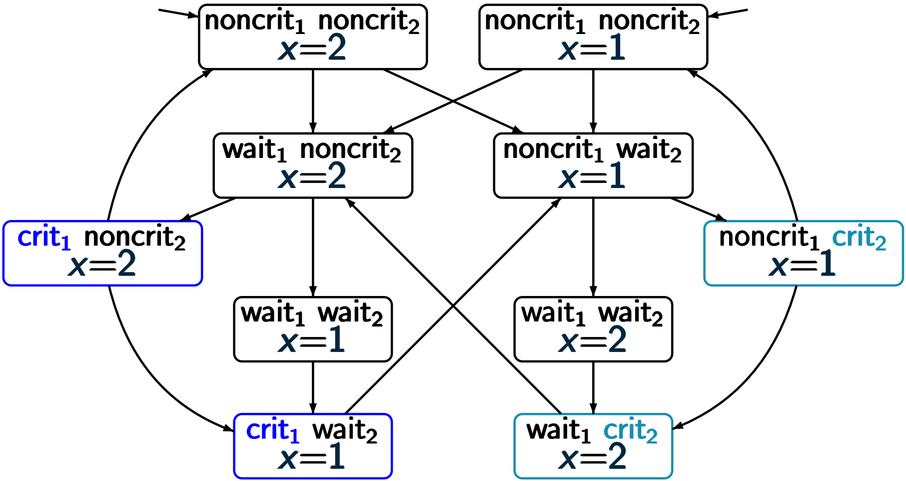
Program graphs for Peterson algorithm



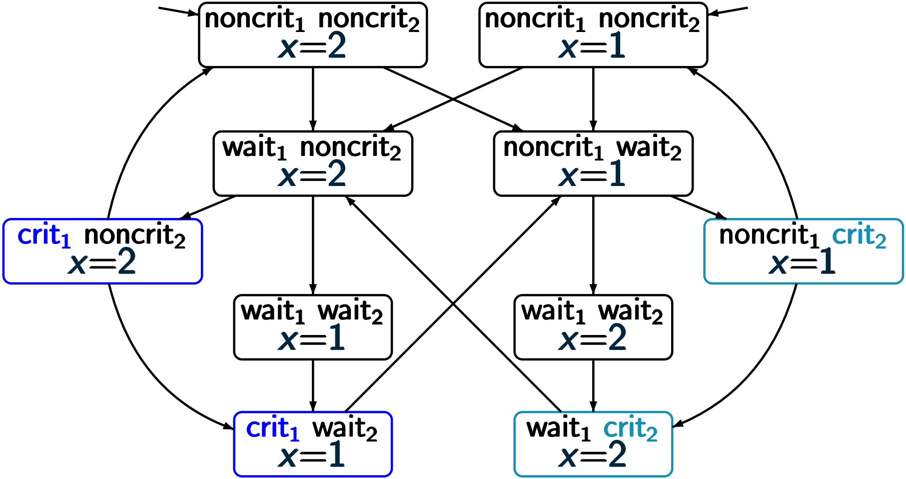
program graph
$\mathcal{P}_1 \,|||\, \mathcal{P}_2$



TS for the Peterson algorithm



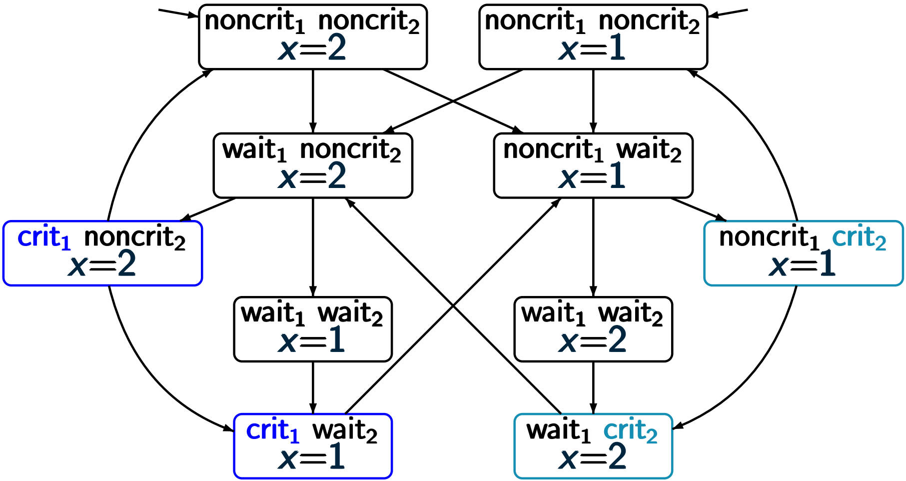
TS for the Peterson algorithm



value of b_1 is given by $\mathbf{wait_1} \vee \mathbf{crit_1}$

value of b_2 is given by $\mathbf{wait_2} \vee \mathbf{crit_2}$

TS for the Peterson algorithm

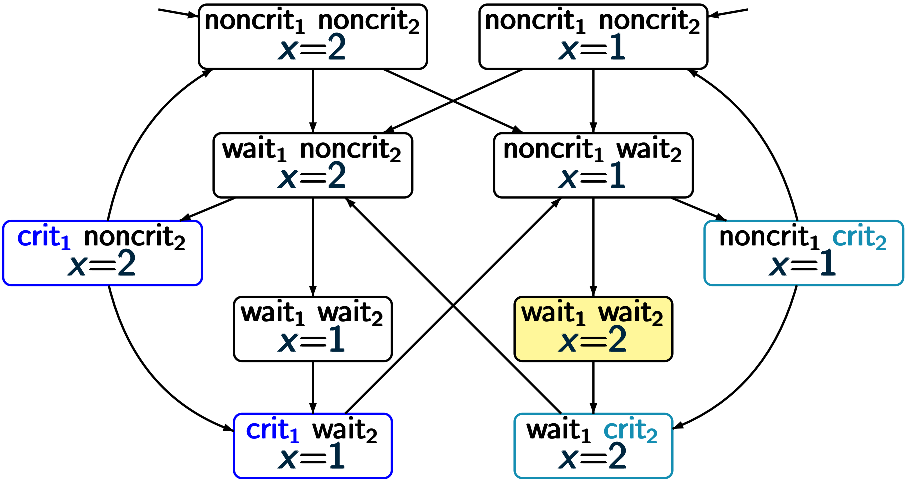


value of b_1 is given by **wait$_1$** $\vee$ **crit$_1$**

value of b_2 is given by **wait$_2$** $\vee$ **crit$_2$**

+ unreachable states

TS for the Peterson algorithm

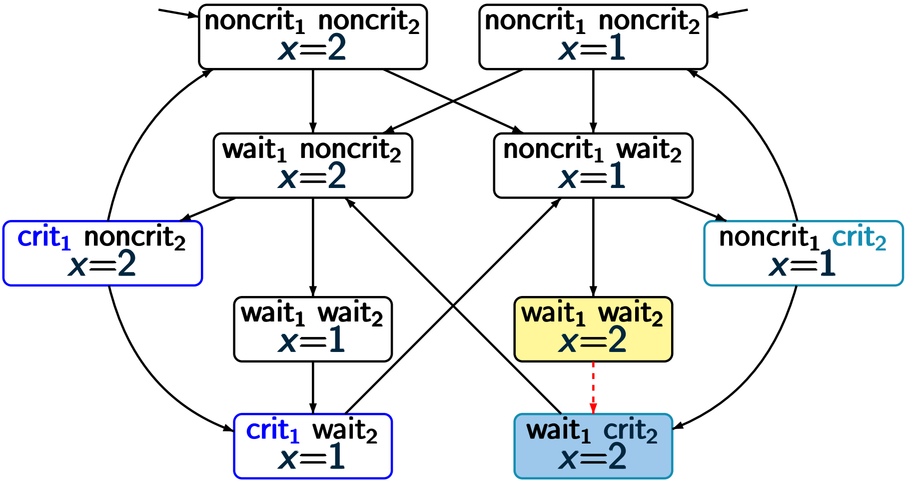


value of b_1 is given by **wait$_1$** $\vee$ **crit$_1$**

value of b_2 is given by **wait$_2$** $\vee$ **crit$_2$**

+ unreachable states

TS for the Peterson algorithm

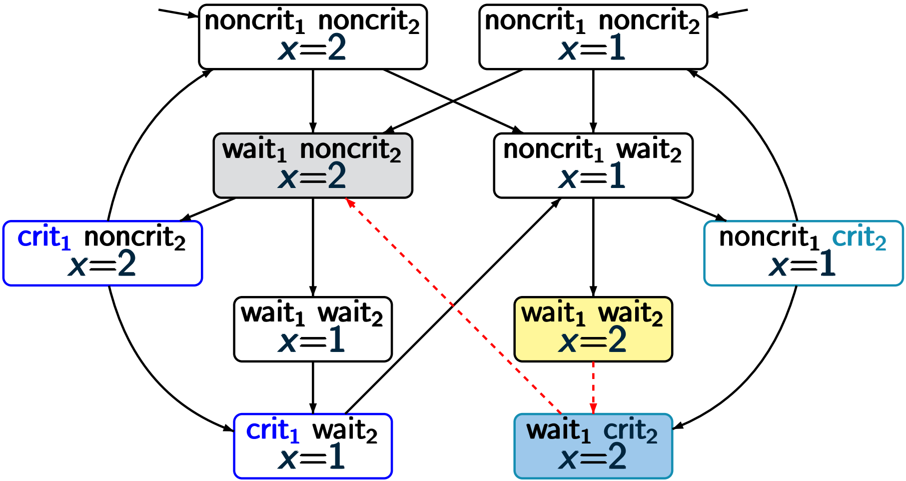


value of b_1 is given by **wait**$_1$ $\vee$ **crit**$_1$

value of b_2 is given by **wait**$_2$ $\vee$ **crit**$_2$

+ unreachable states

TS for the Peterson algorithm

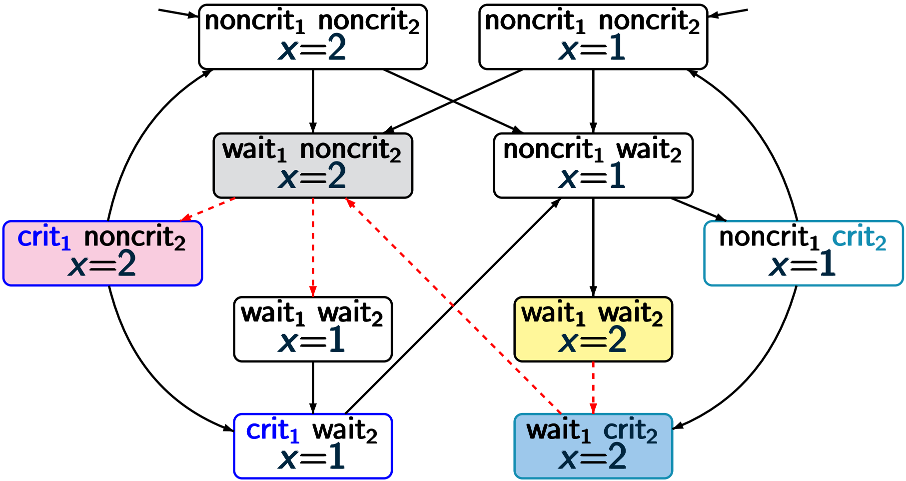


value of b_1 is given by **wait$_1$** $\vee$ **crit$_1$**

value of b_2 is given by **wait$_2$** $\vee$ **crit$_2$**

+ unreachable states

TS for the Peterson algorithm

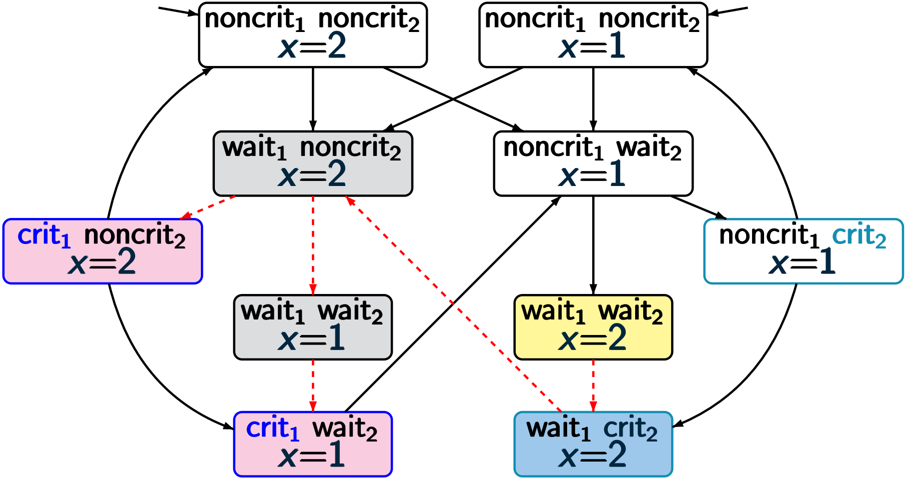


value of b_1 is given by **wait$_1$** $\vee$ **crit$_1$**

value of b_2 is given by **wait$_2$** $\vee$ **crit$_2$**

+ unreachable states

TS for the Peterson algorithm

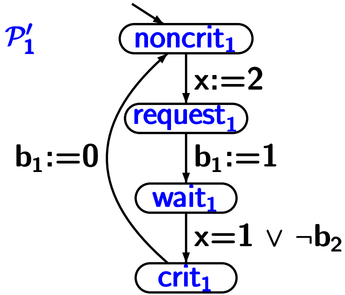


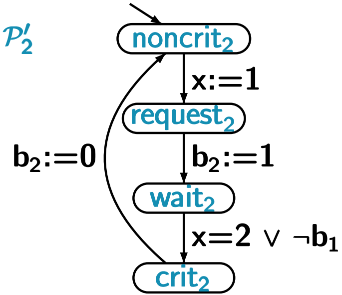
value of b_1 is given by **wait$_1$** $\vee$ crit$_1$

value of b_2 is given by **wait$_2$** $\vee$ crit$_2$

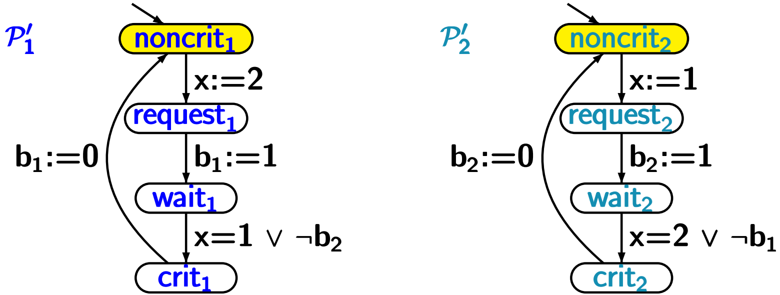
+ unreachable states

Variant of Peterson algorithm





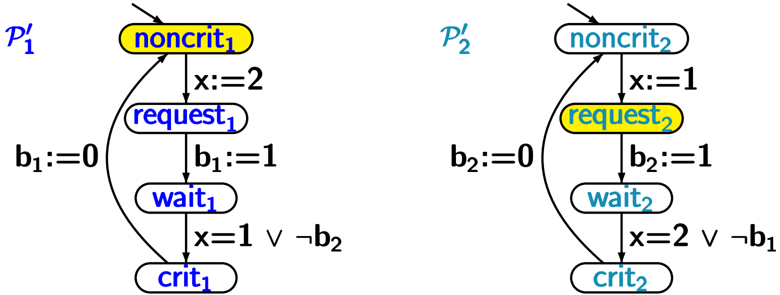
Variant of Peterson algorithm



possible executions

noncrit$_1$ noncrit$_2$ $x=1$ $\neg b_1$ $\neg b_2$

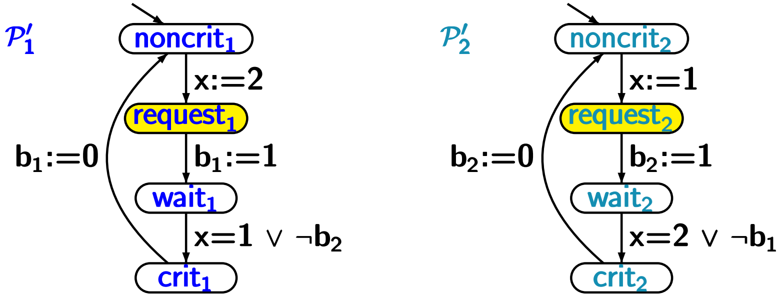
Variant of Peterson algorithm



possible executions

noncrit$_1$	noncrit$_2$	$x=1$	$\neg b_1$	$\neg b_2$
noncrit$_1$	request$_2$	$x=1$	$\neg b_1$	$\neg b_2$

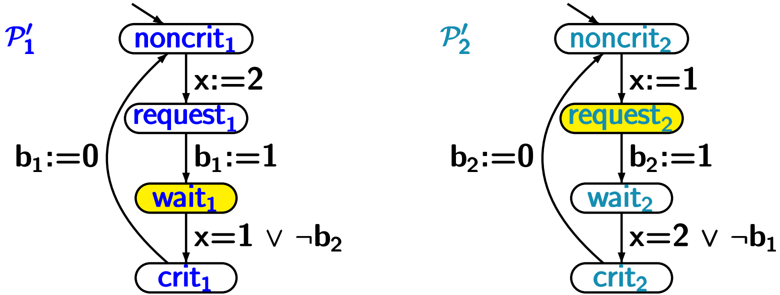
Variant of Peterson algorithm



possible executions

$noncrit_1$	$noncrit_2$	$x=1$	$\neg b_1$	$\neg b_2$
$noncrit_1$	$request_2$	$x=1$	$\neg\ b_1$	$\neg b_2$
$request_1$	$request_2$	$x=2$	$\neg b_1$	$\neg b_2$

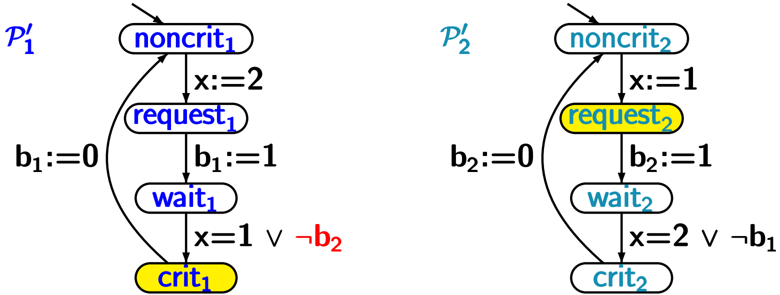
Variant of Peterson algorithm



possible executions

$noncrit_1$	$noncrit_2$	$x=1$	$\neg b_1$	$\neg b_2$
$noncrit_1$	$request_2$	$x=1$	$\neg b_1$	$\neg b_2$
$request_1$	$request_2$	$x=2$	$\neg b_1$	$\neg b_2$
$wait_1$	$request_2$	$x=2$	b_1	$\neg b_2$

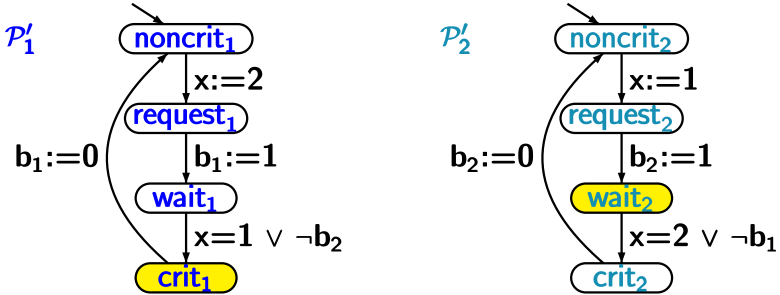
Variant of Peterson algorithm



possible executions

$noncrit_1$	$noncrit_2$	$x=1$	$\neg b_1$	$\neg b_2$
$noncrit_1$	$request_2$	$x=1$	$\neg b_1$	$\neg b_2$
$request_1$	$request_2$	$x=2$	$\neg b_1$	$\neg b_2$
$wait_1$	$request_2$	$x=2$	b_1	$\neg b_2$
$crit_1$	$request_2$	$x=2$	b_1	$\neg b_2$

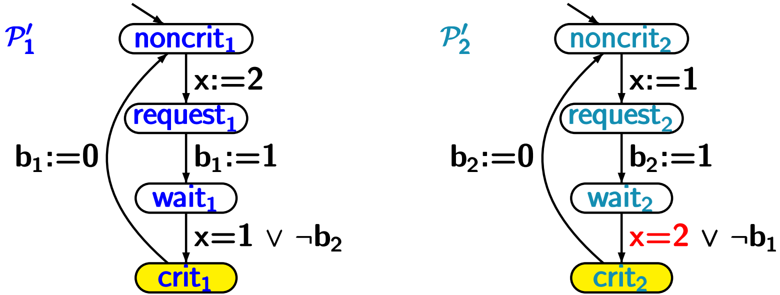
Variant of Peterson algorithm



possible executions

noncrit$_1$	noncrit$_2$	x=1	$\neg b_1$	$\neg b_2$
noncrit$_1$	request$_2$	x=1	$\neg b_1$	$\neg b_2$
request$_1$	request$_2$	x=2	$\neg b_1$	$\neg b_2$
wait$_1$	request$_2$	x=2	b_1	$\neg b_2$
crit$_1$	request$_2$	x=2	b_1	$\neg b_2$
crit$_1$	wait$_2$	x=2	b_1	b_2

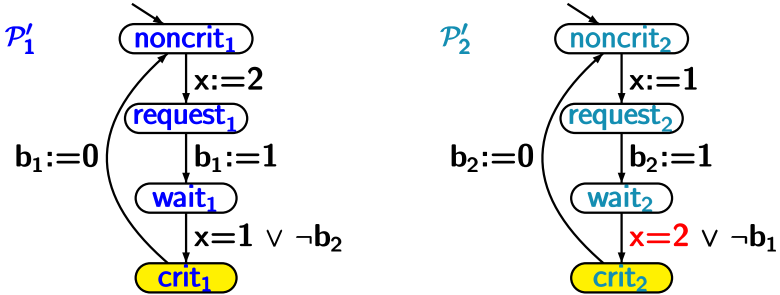
Variant of Peterson algorithm



possible executions

$noncrit_1$	$noncrit_2$	$x=1$	$\neg b_1$	$\neg b_2$
$noncrit_1$	$request_2$	$x=1$	$\neg b_1$	$\neg b_2$
$request_1$	$request_2$	$x=2$	$\neg b_1$	$\neg b_2$
$wait_1$	$request_2$	$x=2$	b_1	$\neg b_2$
$crit_1$	$request_2$	$x=2$	b_1	$\neg b_2$
$crit_1$	$wait_2$	$x=2$	b_1	b_2
$crit_1$	$crit_2$	$x=2$	b_1	b_2

Variant of Peterson algorithm incorrect!



possible executions

$noncrit_1$	$noncrit_2$	$x=1$	$\neg b_1$	$\neg b_2$
$noncrit_1$	$request_2$	$x=1$	$\neg b_1$	$\neg b_2$
$request_1$	$request_2$	$x=2$	$\neg b_1$	$\neg b_2$
$wait_1$	$request_2$	$x=2$	b_1	$\neg b_2$
$crit_1$	$request_2$	$x=2$	b_1	$\neg b_2$
$crit_1$	$wait_2$	$x=2$	b_1	b_2
$crit_1$	$crit_2$	$x=2$	b_1	b_2

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS (no communication, no dependencies)

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS (no communication, no dependencies)
- communication via shared variables
  - description of subsystems by program graphs
  - interleaving $|||$ for program graphs
  - TS is obtained by "unfolding"

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS (no communication, no dependencies)
- communication via shared variables
  - description of subsystems by program graphs
  - interleaving $|||$ for program graphs
  - TS is obtained by "unfolding"
- synchronous message passing

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS (no communication, no dependencies)
- communication via shared variables
  - description of subsystems by program graphs
  - interleaving $|||$ for program graphs
  - TS is obtained by "unfolding"
- synchronous message passing
  - operator $\|_{Syn}$ for TS
  - interleaving for independent actions
  - synchronization over actions in *Syn*

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS (no communication, no dependencies)
- communication via shared variables
  - description of subsystems by program graphs
  - interleaving $|||$ for program graphs
  - TS is obtained by "unfolding"
- synchronous message passing $\longleftarrow$ data abstract
  - operator $\|_{Syn}$ for TS
  - interleaving for independent actions
  - synchronization over actions in Syn

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS (no communication, no dependencies)
- communication via shared variables
  - ∗ description of subsystems by program graphs
  - ∗ interleaving $|||$ for program graphs
  - ∗ TS is obtained by "unfolding"
- synchronous message passing $\longleftarrow$ data abstract
  - ∗ operator $\|_{Syn}$ for TS
  - ∗ interleaving for independent actions
  - ∗ synchronization over actions in Syn
- channel systems
  communication via shared variables + via channels

Synchronous message passing

Synchronous message passing

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

Synchronous message passing

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

composite transition system:

$$\mathcal{T}_1 \,\|_{Syn}\, \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$$

for modeling the concurrent execution of $\mathcal{T}_1$ and $\mathcal{T}_2$ with synchronization over all actions in Syn

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

composite transition system:

$$\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$$

interleaving for all actions $\alpha \in Act_i \setminus Syn$:

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

Synchronous message passing

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

composite transition system:

$$\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$$

interleaving for all actions $\alpha \in Act_i \setminus Syn$:

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

handshaking (rendezvous) for all $\alpha \in Syn$:

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

composite transition system:

$$\mathcal{T}_1 \,\|_{Syn}\, \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$$

interleaving for all actions $\alpha \in Act_i \setminus Syn$:

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

handshaking (rendezvous) for all $\alpha \in Syn$:

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

Mutual exclusion

by synchronous message passing

Mutual exclusion

by synchronous message passing using an arbiter

Mutual exclusion with an arbiter

protocol for process P_i

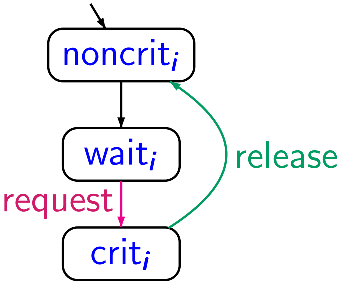
```
LOOP FOREVER DO
      noncritical actions
      request
      critical section
      release
      noncritical actions
OD
```

Mutual exclusion with an arbiter

protocol for process P_i

```
LOOP FOREVER DO
      noncritical actions
      request
      critical section
      release
      noncritical actions
OD
```

transition system $\mathcal{T}_i$

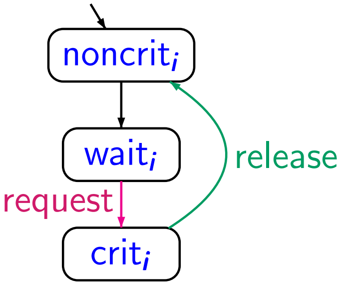


Mutual exclusion with an arbiter

protocol for process P_i

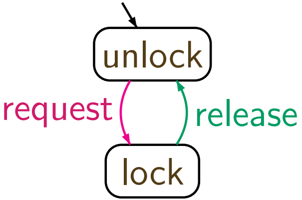
```
LOOP FOREVER DO
      noncritical actions
      request
      critical section
      release
      noncritical actions
OD
```

transition system $\mathcal{T}_i$



Arbiter:

selects nondeterministically a synchronization partner $\mathcal{T}_1$ or $\mathcal{T}_2$



Mutual exclusion with an arbiter

$$(\mathcal{T}_1 \; ||| \; \mathcal{T}_2) \; \|_{\mathit{Syn}} \; \mathit{Arbiter} \text{ where } \mathit{Syn} = \{\mathsf{request}, \mathsf{release}\}$$

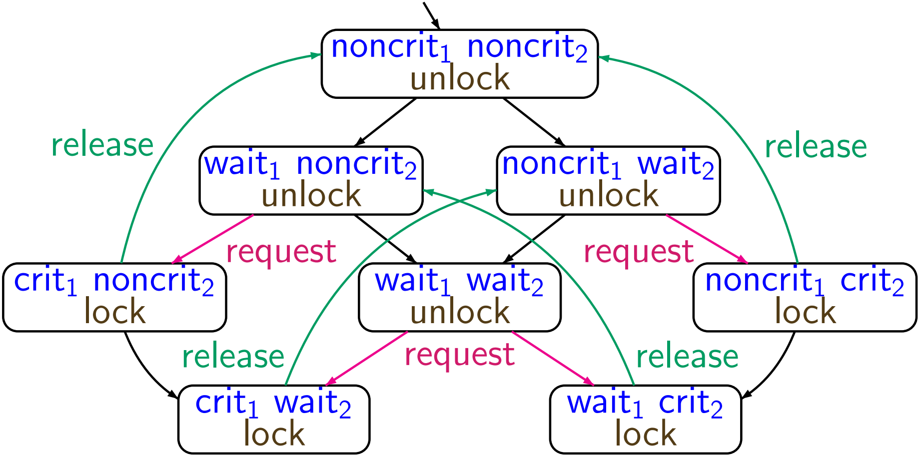
$$(\mathcal{T}_1 \;|||\; \mathcal{T}_2) \;\|_{Syn}\; \textit{Arbiter} \text{ where } Syn = \{\text{request}, \text{release}\}$$

"pure" interleaving for TS

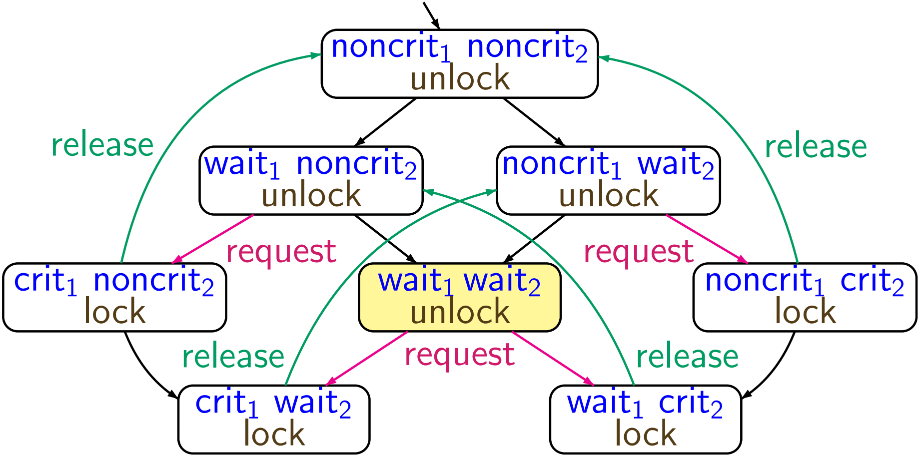
handshaking for actions request and release

Mutual exclusion with an arbiter

$(\mathcal{T}_1 \,|||\, \mathcal{T}_2) \parallel_{Syn} \textit{Arbiter}$ where $Syn = \{\text{request}, \text{release}\}$



$(\mathcal{T}_1 \,|||\, \mathcal{T}_2) \parallel_{Syn} \textit{Arbiter}$ where $Syn = \{\text{request}, \text{release}\}$



nondeterministic choice: who enters the critical section**?**

Synchronous message passing

synchronization operator $\|_{Syn}$ for
three or more processes

Synchronous message passing

$$\left.\begin{array}{rcl} \mathcal{T}_1 & = & (S_1, Act_1, \rightarrow_1, \ldots) \\ \mathcal{T}_2 & = & (S_2, Act_2, \rightarrow_2, \ldots) \\ \mathcal{T}_3 & = & (S_3, Act_3, \rightarrow_3, \ldots) \\ \mathcal{T}_4 & = & (S_4, Act_4, \rightarrow_4, \ldots) \end{array}\right\} \text{ transition systems}$$

$$\begin{array}{rcl} \vdots & & \vdots \end{array}$$

Synchronous message passing

$$\left.\begin{array}{lcl} \mathcal{T}_1 & = & (S_1, Act_1, \rightarrow_1, \ldots) \\ \mathcal{T}_2 & = & (S_2, Act_2, \rightarrow_2, \ldots) \\ \mathcal{T}_3 & = & (S_3, Act_3, \rightarrow_3, \ldots) \\ \mathcal{T}_4 & = & (S_4, Act_4, \rightarrow_4, \ldots) \end{array}\right\} \text{transition systems}$$

$$\vdots \qquad\qquad \vdots$$

for $Syn \subseteq Act_1 \cup Act_2 \cup Act_3 \cup Act_4 \cup \ldots$

$$\boxed{\begin{array}{l} \mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 \parallel_{Syn} \mathcal{T}_3 \parallel_{Syn} \mathcal{T}_4 \parallel_{Syn} \ldots \overset{\text{def}}{=} \\ \qquad \Big(((\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2) \parallel_{Syn} \mathcal{T}_3) \parallel_{Syn} \mathcal{T}_4\Big) \parallel_{Syn} \ldots \end{array}}$$

Synchronous message passing

$$\left.\begin{array}{lcl}\mathcal{T}_1 & = & (S_1, Act_1, \rightarrow_1, \ldots) \\ \mathcal{T}_2 & = & (S_2, Act_2, \rightarrow_2, \ldots) \\ \mathcal{T}_3 & = & (S_3, Act_3, \rightarrow_3, \ldots) \\ \mathcal{T}_4 & = & (S_4, Act_4, \rightarrow_4, \ldots) \\ \vdots & & \qquad\quad \vdots \end{array}\right\} \text{transition systems}$$

for $Syn \subseteq Act_1 \cup Act_2 \cup Act_3 \cup Act_4 \cup \ldots$

$$\boxed{\begin{array}{l}\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 \parallel_{Syn} \mathcal{T}_3 \parallel_{Syn} \mathcal{T}_4 \parallel_{Syn} \ldots \stackrel{\text{def}}{=} \\ \qquad \Big(((\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2) \parallel_{Syn} \mathcal{T}_3) \parallel_{Syn} \mathcal{T}_4\Big) \parallel_{Syn} \ldots \end{array}}$$

or any other order of paranthesis

Synchronous message passing

$$\left.\begin{array}{lcl} \mathcal{T}_1 & = & (S_1, Act_1, \rightarrow_1, \ldots) \\ \mathcal{T}_2 & = & (S_2, Act_2, \rightarrow_2, \ldots) \\ \mathcal{T}_3 & = & (S_3, Act_3, \rightarrow_3, \ldots) \\ \mathcal{T}_4 & = & (S_4, Act_4, \rightarrow_4, \ldots) \\ \vdots & & \quad\vdots \end{array}\right\} \text{ transition systems}$$

for $Syn \subseteq Act_1 \cup Act_2 \cup Act_3 \cup Act_4 \cup \ldots$

$$\boxed{\begin{array}{l} \mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 \parallel_{Syn} \mathcal{T}_3 \parallel_{Syn} \mathcal{T}_4 \parallel_{Syn} \ldots \overset{\text{def}}{=} \\ \qquad \Big(((\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2) \parallel_{Syn} \mathcal{T}_3) \parallel_{Syn} \mathcal{T}_4\Big) \parallel_{Syn} \ldots \end{array}}$$

where, e.g., $\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 \overset{\text{def}}{=} \mathcal{T}_1 \parallel_H \mathcal{T}_2$

with $H = Syn \cap Act_1 \cap Act_2$

Parallel operator $\|$

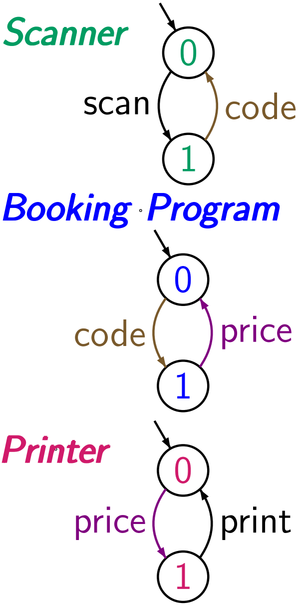
$$\begin{array}{lcl} \mathcal{T}_1 & = & (S_1, Act_1, \rightarrow_1, \ldots) \\ \mathcal{T}_2 & = & (S_2, Act_2, \rightarrow_2, \ldots) \\ \mathcal{T}_3 & = & (S_3, Act_3, \rightarrow_3, \ldots) \\ \mathcal{T}_4 & = & (S_4, Act_4, \rightarrow_4, \ldots) \\ \vdots & & \vdots \end{array}$$

transition systems s.t. $Act_i \cap Act_j \cap Act_k = \emptyset$ if i, j, k are pairwise distinct

$$\boxed{\begin{array}{l} \mathcal{T}_1 \,\|\, \mathcal{T}_2 \,\|\, \mathcal{T}_3 \,\|\, \mathcal{T}_4 \,\|\, \ldots \stackrel{\text{def}}{=} \\ \quad (((\mathcal{T}_1 \,\|_{Syn_{1,2}}\, \mathcal{T}_2) \,\|_{Syn_{1,2,3}}\, \mathcal{T}_3) \,\|_{Syn_{1,2,3,4}}\, \mathcal{T}_4) \ldots \end{array}}$$

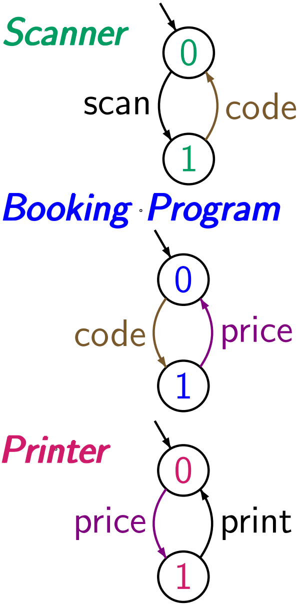
where

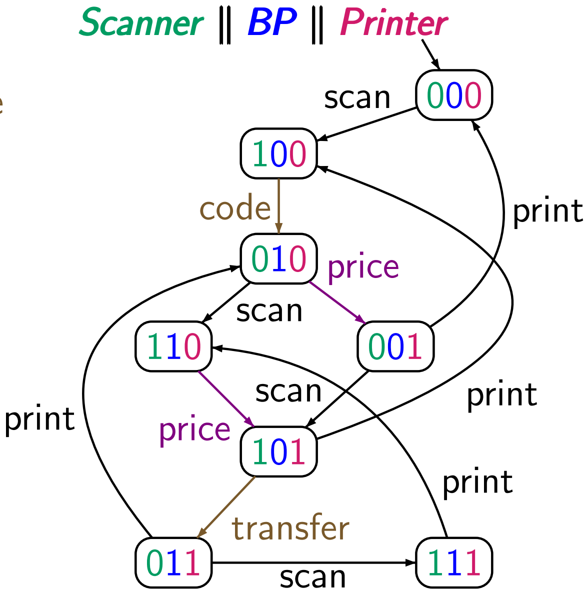
$$\begin{array}{lcl} Syn_{1,2} & = & Act_1 \cap Act_2 \\ Syn_{1,2,3} & = & (Act_1 \cup Act_2) \cap Act_3 \\ Syn_{1,2,3,4} & = & (Act_1 \cup Act_2 \cup Act_3) \cap Act_4 \\ \vdots & & \vdots \end{array}$$



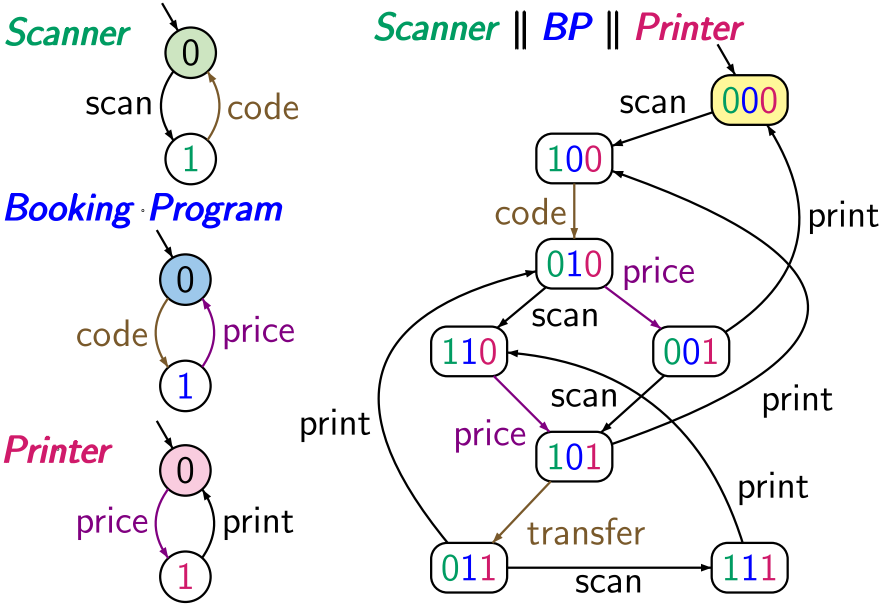
Scanner $\|$ *BP* $\|$ *Printer*

Booking system in supermarket

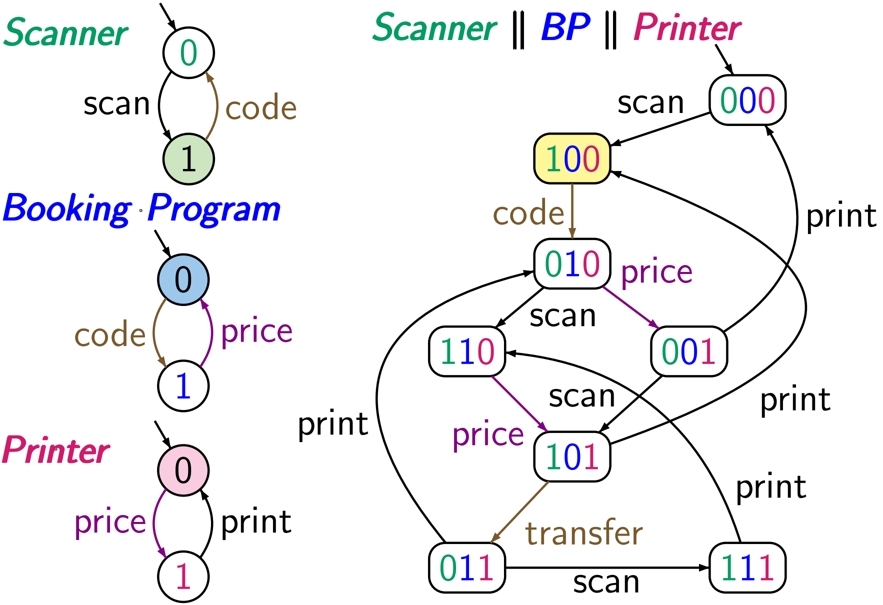




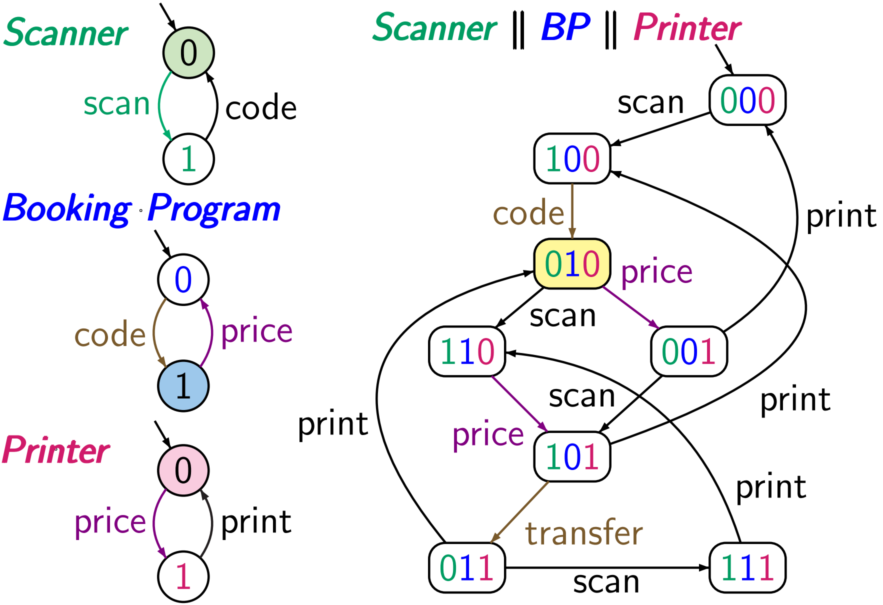
Booking system in supermarket

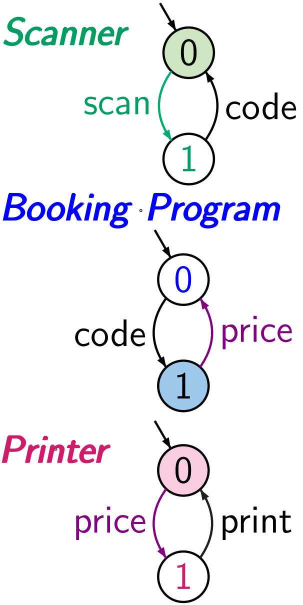


Booking system in supermarket

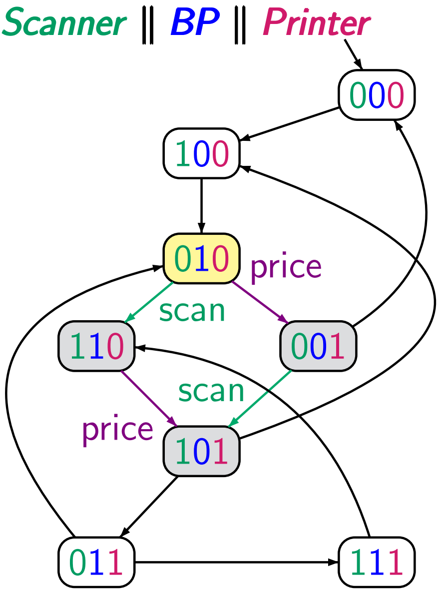


Booking system in supermarket



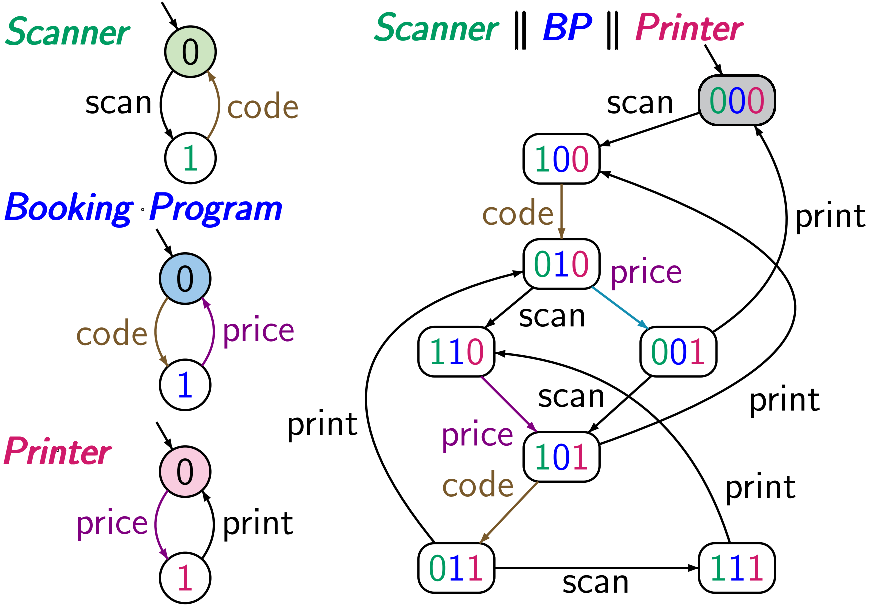


Scanner
0
scan
code
1
Booking Program
0
code
price
1
Printer
0
price
print
1

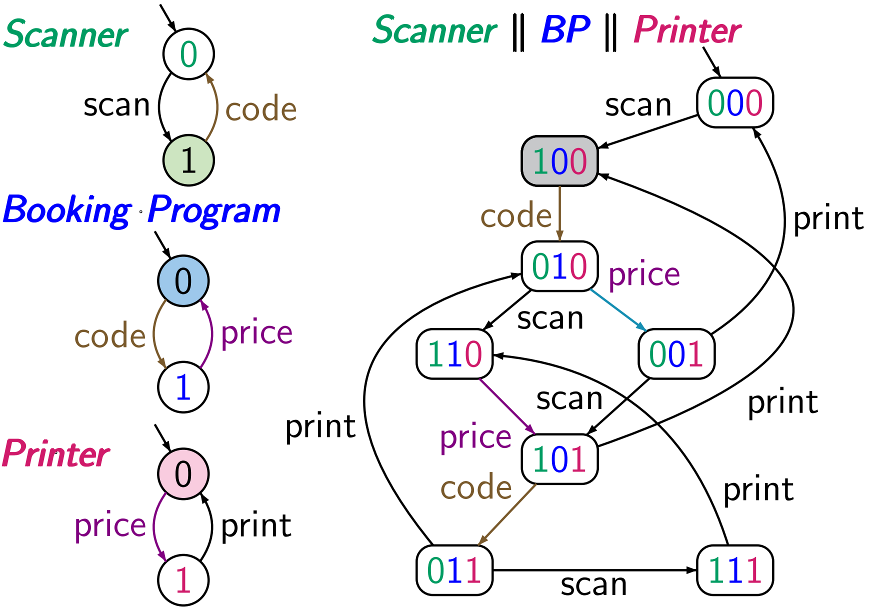


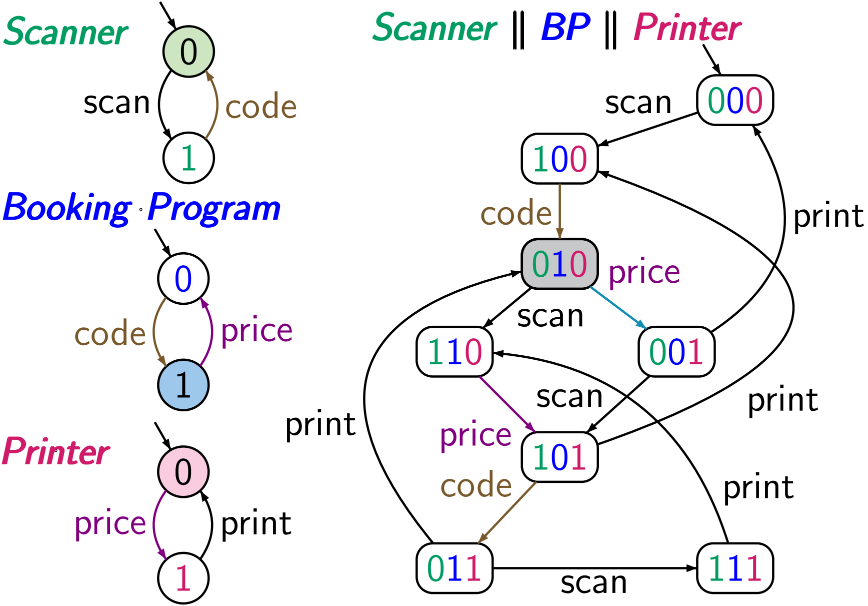
Scanner ‖ BP ‖ Printer
000
100
010
price
scan
110
001
scan
price
101
011
111

Booking system in supermarket

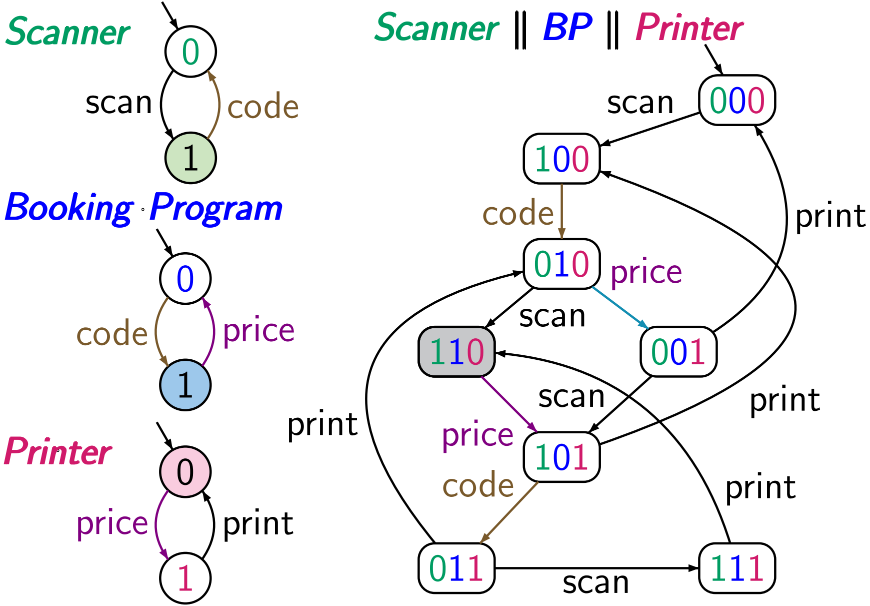


Booking system in supermarket



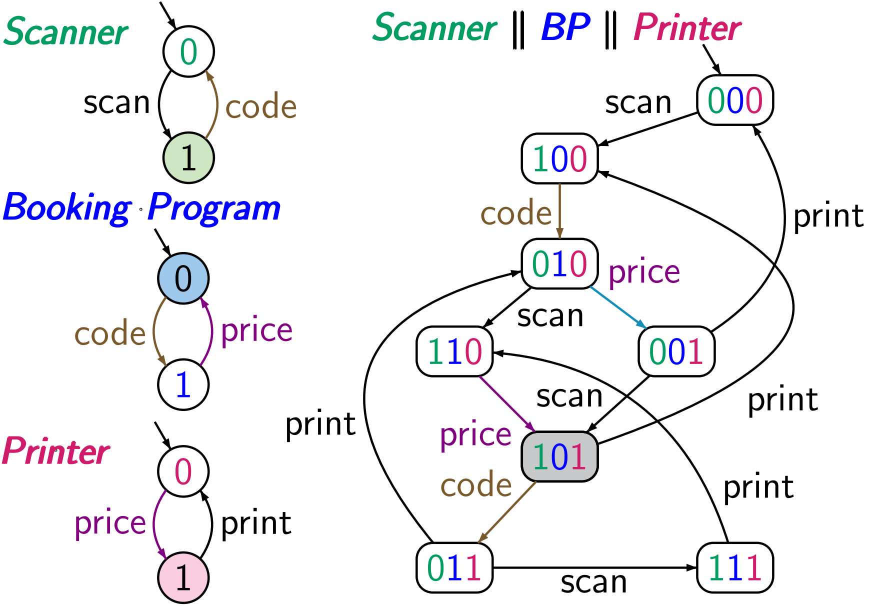


Scanner
0
scan
code
1
Booking Program
0
code
price
1
Printer
0
price
print
1
Scanner ∥ BP ∥ Printer
000
scan
100
code
print
010
price
scan
110
001
scan
print
print
price
101
code
print
011
scan
111

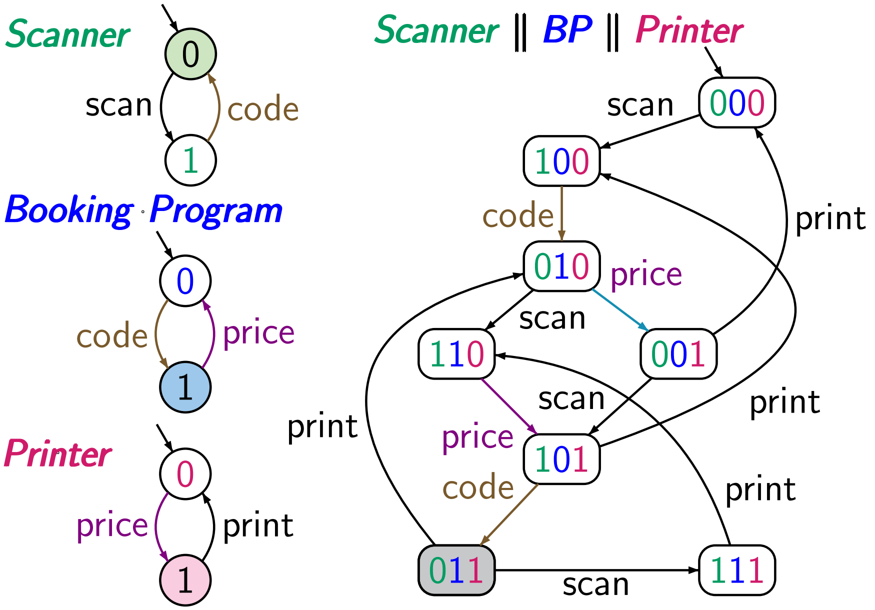


Scanner
0
scan
code
1
Booking Program
0
code
price
1
Printer
0
price
print
1
Scanner || BP || Printer
000
scan
100
code
print
010
price
scan
110
001
scan
print
print
price
101
code
print
011
scan
111

Booking system in supermarket



Booking system in supermarket



Interleaving