Concurrency

Lecture #3 of Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling and Verification

E-mail: `katoen@cs.rwth-aachen.de`

October 28, 2008

Overview Lecture #3

⇒ *Concurrency*

- The interleaving paradigm

- Communication principles
  - Shared variable "communication"
  - Handshaking
  - Synchronous communication

- Channel systems

- The state-space explosion problem

Concurrent systems

- Transition systems
  - suited for modeling sequential data-dependent systems
  - and for modeling sequential hardware circuits
- How about *concurrent* systems?
  - multi-threading
  - distributed algorithms and communication protocols
- Can we model:
  - multi-threading with shared variables?
  - synchronous communication?
  - synchronous composition of hardware?

Interleaving

- Abstract from decomposition of system in components
- Actions of independent components are merged or “interleaved”
  - a single processor is available
  - on which the actions of the processes are interlocked
- No assumptions are made on the order of processes
  - possible orders for non-terminating independent processes P and Q:

$$\begin{array}{cccccccccc} P & Q & P & Q & P & Q & Q & Q & P & \ldots \\ P & P & Q & P & P & Q & P & P & Q & \ldots \\ P & Q & P & P & Q & P & P & P & Q & \ldots \end{array}$$

  - assumption: there is a scheduler with an a priori unknown strategy

Interleaving

- Justification for interleaving:

  the effect of concurrently executed, independent actions α and β equals the effect when α and β are successively executed in arbitrary order

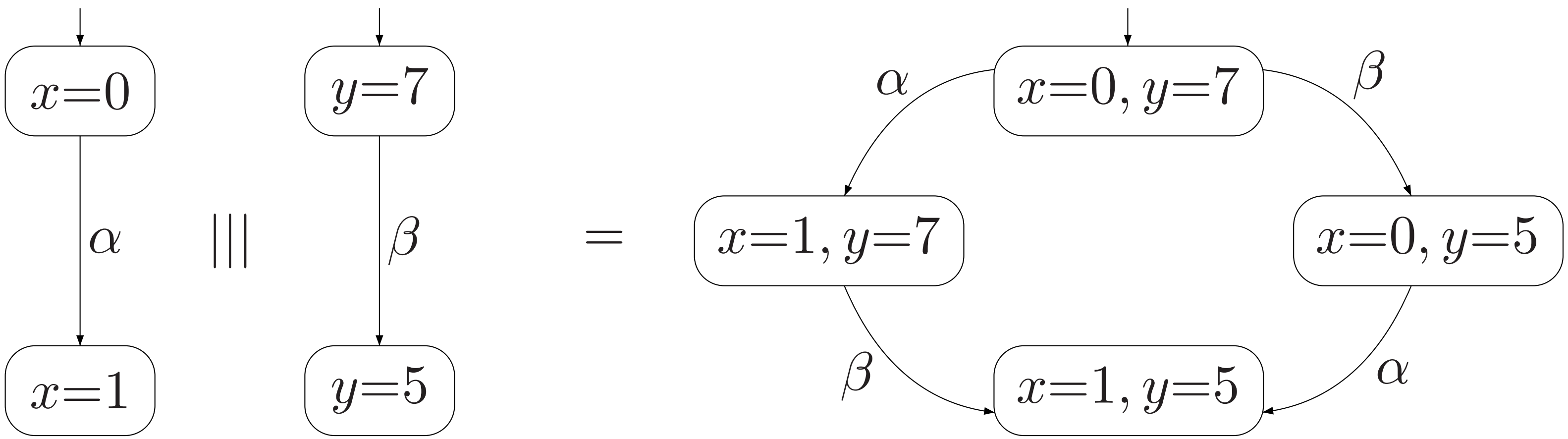
- Symbolically this is stated as:

$$\textit{Effect}(\alpha \,|||\, \beta, \eta) \;=\; \textit{Effect}((\alpha\,;\,\beta) + (\beta\,;\,\alpha), \eta)$$

  - $|||$ stands for the (binary) interleaving operator
  - ";" stands for sequential execution, and "$+$" for non-deterministic choice

Interleaving

$$\underbrace{x := x + 1}_{=\alpha} \;|||\; \underbrace{y := y - 2}_{=\beta}$$



Interleaving of transition systems

Let $\mathit{TS}_i = (S_i, \mathit{Act}_i, \rightarrow_i, I_i, \mathit{AP}_i, L_i)$ $i{=}1, 2$, be two transition systems

Transition system

$$\mathit{TS}_1 \,|||\, \mathit{TS}_2 = (S_1 \times S_2, \mathit{Act}_1 \uplus \mathit{Act}_2, \rightarrow, I_1 \times I_2, \mathit{AP}_1 \uplus \mathit{AP}_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\rightarrow$ is defined by the rules:

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

What are program graphs?

A *program graph* *PG* over a set *Var* of typed variables is a tuple

$$(\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \longrightarrow, \mathit{Loc}_0, g_0) \quad \text{where}$$

- *Loc* is a set of *locations* with initial locations $\mathit{Loc}_0 \subseteq \mathit{Loc}$
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \rightarrow \mathit{Eval}(\mathit{Var})$ is the *effect* function
- $\longrightarrow \subseteq \mathit{Loc} \times \underbrace{\mathit{Cond}(\mathit{Var})}_{\text{Boolean conditions over} \mathit{Var}} \times \mathit{Act} \times \mathit{Loc}$, transition relation
- $g_0 \in \mathit{Cond}(\mathit{Var})$ is the initial *condition*.

Beverage vending machine

- *Loc* $= \{ \, start, select \, \}$ with *Loc*$_0 = \{ \, start \, \}$

- *Act* $= \{ \, bget, sget, coin, ret\_coin, refill \, \}$

- *Var* $= \{ \, nsprite, \;\; nbeer \, \}$ with domain $\{ \, 0, 1, \ldots, max \, \}$

- $$\begin{array}{lcl} \textit{Effect}(coin, \eta) & = & \eta \\ \textit{Effect}(ret\_coin, \eta) & = & \eta \\ \textit{Effect}(sget, \eta) & = & \eta[nsprite := nsprite{-}1] \\ \textit{Effect}(bget, \eta) & = & \eta[nbeer := nbeer{-}1] \\ \textit{Effect}(refill, \eta) & = & [nsprite := max, nbeer := max] \end{array}$$

- $g_0 = (nsprite = max \;\; \wedge \;\; nbeer = max)$

From program graphs to transition systems

- Basic strategy: *unfolding*
  - state = location (current control) ℓ + data valuation η
  - initial state = initial location satisfying the initial condition g_0
- Propositions and labeling
  - propositions: "at ℓ" and "$x \in D$" for $D \subseteq \mathit{dom}(x)$
  - $\langle \ell, \eta \rangle$ is labeled with "at ℓ" and all conditions that hold in η
- if $\ell \xrightarrow{g:\alpha} \ell'$ and g holds in η then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle$

Transition systems for program graphs

The transition system *TS*(PG) of the program graph

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

over a 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 rule: $\dfrac{\ell \xrightarrow{g:\alpha} \ell' \quad \wedge \quad \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$
- $I = \{ \langle \ell, \eta \rangle \mid \ell \in Loc_0, \eta \models g_0 \}$
- $AP = Loc \cup Cond(Var)$ and $L(\langle \ell, \eta \rangle) = \{ \ell \} \cup \{ g \in Cond(Var) \mid \eta \models g \}$.

Interleaving of program graphs

For program graphs PG_1 (on Var_1) and PG_2 (on Var_2) *without* shared variables, i.e., $Var_1 \cap Var_2 = \varnothing$,

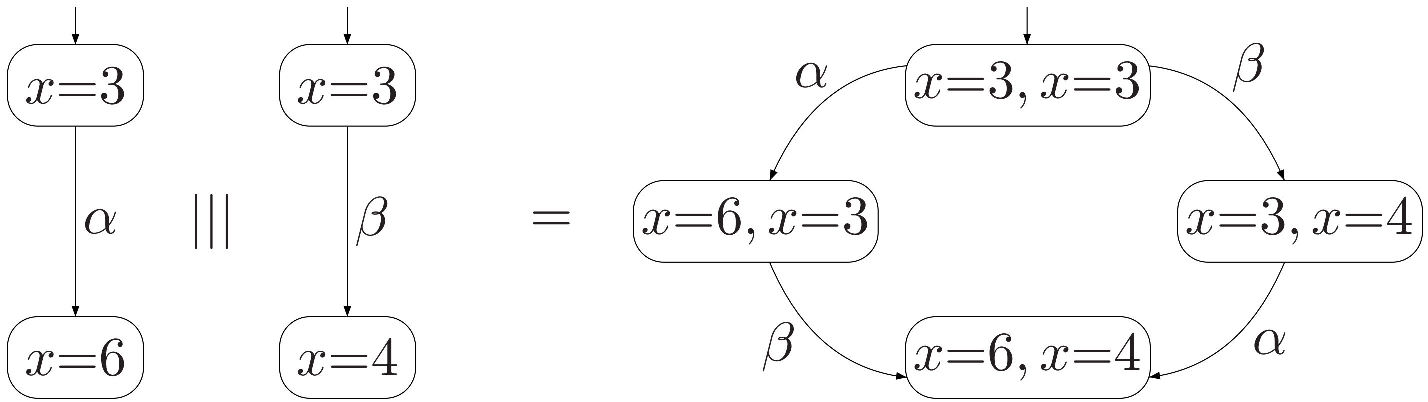
$$TS(PG_1) \; ||| \; TS(PG_2)$$

faithfully describes the concurrent behavior of PG_1 and PG_2

what if they have variables in common?

Shared variable communication

$$\underbrace{x := 2 \cdot x}_{\text{action } \alpha} \;|||\; \underbrace{x := x + 1}_{\text{action } \beta} \qquad \text{with initially } x = 3$$



$\langle x{=}6, x{=}4 \rangle$ is an *inconsistent* state!

$\Rightarrow$ no faithful model of the concurrent execution of α and β

Modeling concurrent program graphs

- If PG_1 and PG_2 share no variables:

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

  - interleaving of transition systems

- If PG_1 and PG_2 share some variables:

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

  - interleaving of program graphs

- In general: $TS(PG_1) \mid\mid\mid TS(PG_2) \neq TS(PG_1 \mid\mid\mid PG_2)$

Interleaving of program graphs

Let $PG_i = (Loc_i, Act_i, Effect_i, \longrightarrow_i, Loc_{0,i}, g_{0,i})$ over variables Var_i.

Program graph $PG_1 \,|||\, PG_2$ over $Var_1 \cup Var_2$ is defined by:

$$(Loc_1 \times Loc_2, Act_1 \uplus Act_2, Effect, \longrightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \wedge g_{0,2})$$

where $\longrightarrow$ is defined by the inference rules:

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

and $Effect(\alpha, \eta) \;=\; Effect_i(\alpha, \eta)$ if $\alpha \in Act_i$.

Example

$$\underbrace{x := 2{\cdot}x}_{\text{action } \alpha} \;|||\; \underbrace{x := x+1}_{\text{action } \beta} \qquad \text{with initially } x = 3$$

note that $\mathit{TS}(\mathit{PG}_1) \,|||\, \mathit{TS}(\mathit{PG}_2) \neq \mathit{TS}(\mathit{PG}_1 \,|||\, \mathit{PG}_2)$

On atomicity

$$\underbrace{x := x + 1; y := 2x + 1; z := y \textbf{ div } x}_{\text{non-atomic}} \;|||\; x := 0$$

Possible execution fragment:

$$\langle x = 11\rangle \xrightarrow{x:=x+1} \langle x = 12\rangle \xrightarrow{y:=2x+1} \langle x = 12\rangle \xrightarrow{x:=0} \langle x = 0\rangle \xrightarrow{z:=y/x} \dagger \ldots$$

$$\underbrace{\langle x := x + 1; y := 2x + 1; z := y \textbf{ div } x\rangle}_{\text{atomic}} \;|||\; x := 0$$

Either the left process or the right process is completed first:

$$\langle x = 11\rangle \xrightarrow{x:=x+1} \langle x = 12\rangle \xrightarrow{y:=2x+1} \langle x = 12\rangle \xrightarrow{z:=y/x} \langle x = 12\rangle \xrightarrow{x:=0} \langle x = 0\rangle$$

Peterson's mutual exclusion algorithm

P_1 **loop forever**

$\vdots$ (* non-critical actions *)

$\langle b_1$:= true; $x := 2\rangle$; (* request *)

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

do critical section **od**

b_1 := false (* release *)

$\vdots$ (* non-critical actions *)

end loop

b_i is true if and only if process P_i is waiting or in critical section

if both processes want to enter their critical section, x decides who gets access

Banking system

Person Left behaves as follows:

$$
\begin{array}{ll}
 & \textbf{while } \text{true} \ \{ \\
 & \quad \ldots\ldots \\
nc: & \quad \langle b_1, x = \text{true}, 2; \rangle \\
wt: & \quad \textbf{wait until}(x == 1 \,||\, \neg b_2) \ \{ \\
cs: & \quad\quad \ldots @\text{account} \ldots \} \\
 & \quad b_1 = \text{false}; \\
 & \quad \ldots\ldots \\
 & \}
\end{array}
$$

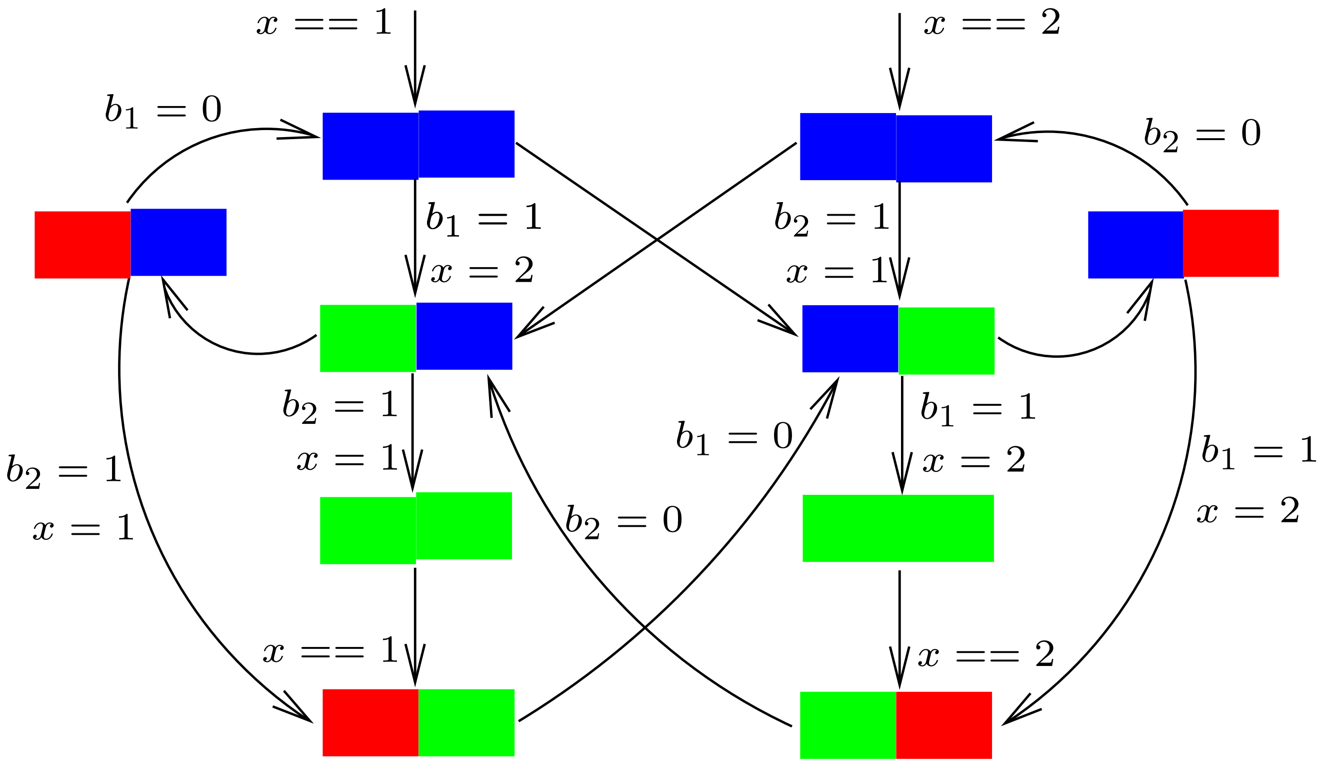
Person Right behaves as follows:

$$
\begin{array}{ll}
 & \textbf{while } \text{true} \ \{ \\
 & \quad \ldots\ldots \\
nc: & \quad \langle b_2, x = \text{true}, 1; \rangle \\
wt: & \quad \textbf{wait until}(x == 2 \,||\, \neg b_1) \ \{ \\
cs: & \quad\quad \ldots @\text{account} \ldots \} \\
 & \quad b_2 = \text{false}; \\
 & \quad \ldots\ldots \\
 & \}
\end{array}
$$

Can we guarantee that only one person at a time has access to the bank account?

Program graph representation

Is the banking system safe?



Manually inspect whether two may have access to the account simultaneously:No

Banking system with non-atomic assignment

Person Left behaves as follows:

	while true $\{$
	$\ldots\ldots$
nc :	$x = 2;$
rq :	$b_1 =$ true;
wt :	$\textbf{wait until}(x == 1 \,\|\|\, \neg b_2)\ \{$
cs :	$\ldots$ @account $\ldots\}$
	$b_1 =$ false;
	$\ldots\ldots$
	$\}$

Person Right behaves as follows:

	while true $\{$
	$\ldots\ldots$
nc :	$x = 1;$
rq :	$b_2 =$ true;
wt :	$\textbf{wait until}(x == 2 \,\|\|\, \neg b_1)\ \{$
cs :	$\ldots$ @account $\ldots\}$
	$b_2 =$ false;
	$\ldots\ldots$
	$\}$

On atomicity again

Assume that the location inbetween the assignments $x := \ldots$ and $b_i :=$ true in program graph *PG*$_i$ is called rq_i. Possible state sequence:

$$
\begin{array}{lllll}
\langle nc_1, & nc_2, & x = 1, & b_1 = \text{false}, & b_2 = \text{false}\rangle \\
\langle nc_1, & rq_2, & \underline{x = 1}, & b_1 = \text{false}, & b_2 = \text{false}\rangle \\
\langle rq_1, & rq_2, & \underline{x = 2}, & b_1 = \text{false}, & b_2 = \text{false}\rangle \\
\langle wt_1, & rq_2, & x = 2, & \underline{b_1 = \text{true}}, & b_2 = \text{false}\rangle \\
\langle cs_1, & rq_2, & x = 2, & b_1 = \text{true}, & b_2 = \text{false}\rangle \\
\langle cs_1, & wt_2, & x = 2, & b_1 = \text{true}, & \underline{b_2 = \text{true}}\rangle \\
\langle cs_1, & cs_2, & x = 2, & b_1 = \text{true}, & b_2 = \text{true}\rangle !
\end{array}
$$

violation of the mutual exclusion property

Parallelism and handshaking

- Concurrent processes run truly in parallel
- To obtain cooperation, some interaction mechanism is needed
- If processes are distributed there is no shared memory

$\Rightarrow$ Message passing

- synchronous message passing (= handshaking)
- asynchronous message passing (= channel communication)

Handshaking

- Concurrent processes interact by *synchronous message passing*
  - processes execute synchronized actions together
  - that is, in interaction both processes need to participate at the same time
  - the interacting processes “shake hands”

- Abstract from information that is exchanged

- H is a set of *handshake actions*
  - actions outside H are independent and are interleaved
  - actions in H need to be synchronized

Handshaking

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, $i{=}1,2$ and $H \subseteq Act_1 \cap Act_2$

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

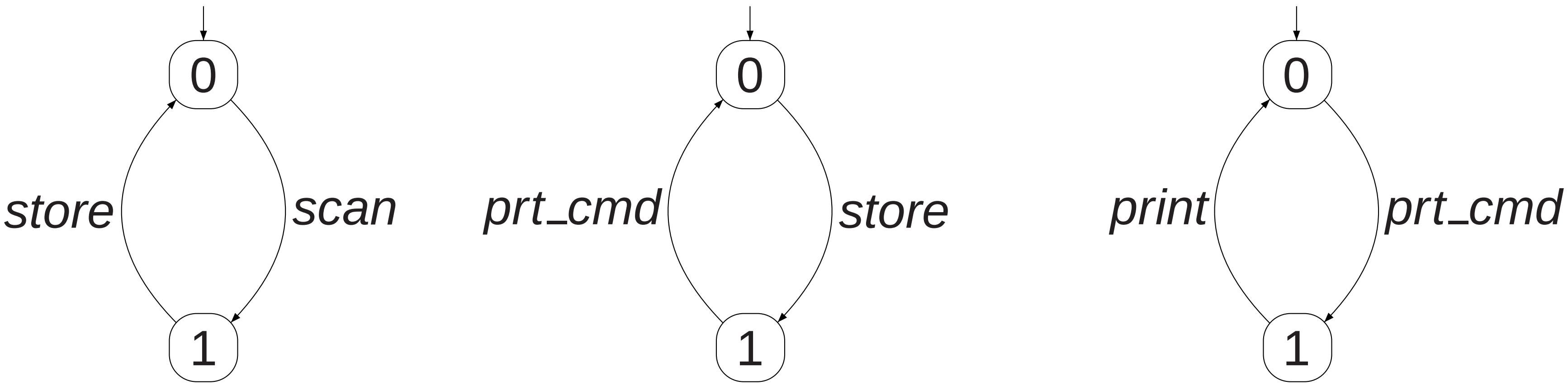
where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and with $\rightarrow$ defined by:

- $\dfrac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \dfrac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$ interleaving for $\alpha \notin H$

- $\dfrac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle}$ handshaking for $\alpha \in H$

note that $TS_1 \parallel_H TS_2 = TS_2 \parallel_H TS_1$ but $(TS_1 \parallel_{H_1} TS_2) \parallel_{H_2} TS_3 \neq TS_1 \parallel_{H_1} (TS_2 \parallel_{H_2} TS_3)$

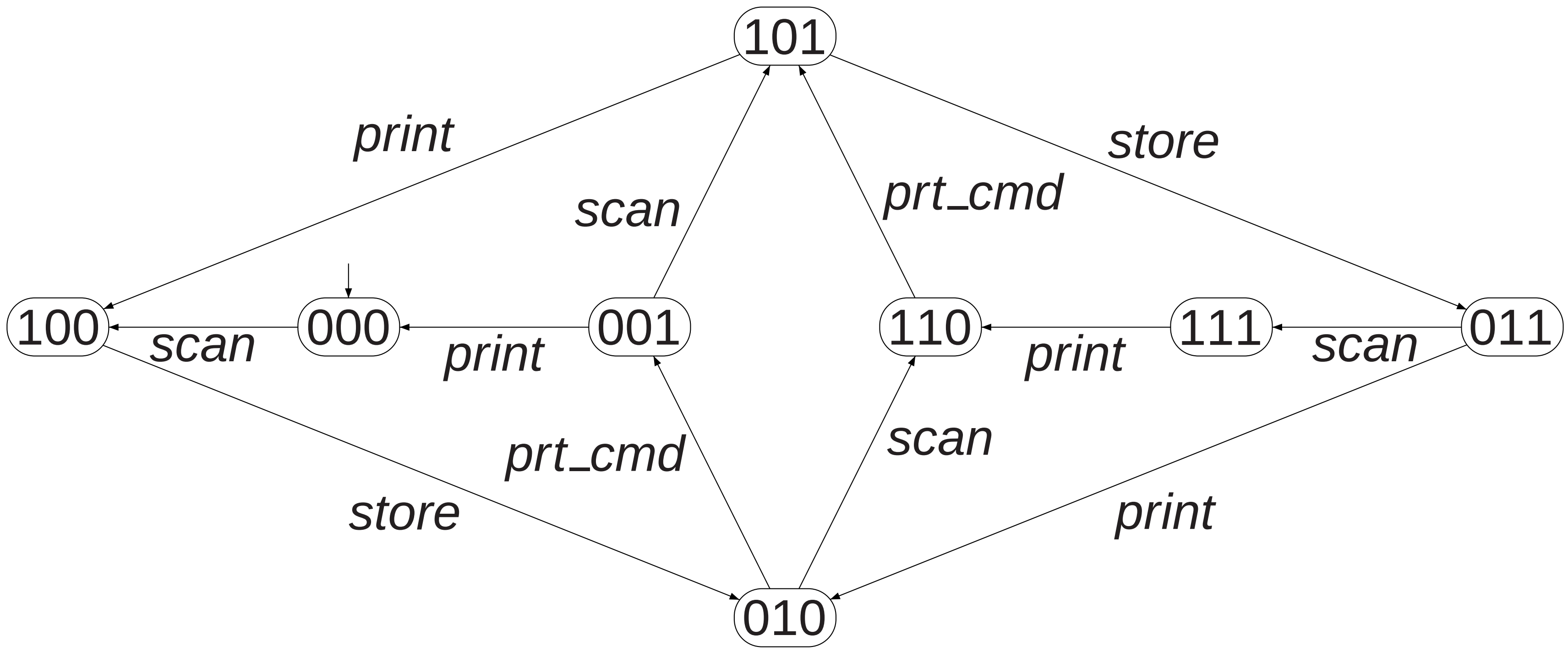
A booking system



$BCR \parallel BP \parallel Printer$

$\parallel$ is a shorthand for $\parallel_H$ with $H = Act_1 \cap Act_2$

The parallel composition



Pairwise handshaking

$TS_1 \| \ldots \| TS_n$ for $H_{i,j} = Act_i \cap Act_j$ with $H_{i,j} \cap Act_k = \varnothing$ for $k \notin \{ i, j \}$

State space of $TS_1 \| \ldots \| TS_n$ is the Cartesian product of those of TS_i

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

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

Synchronous parallelism

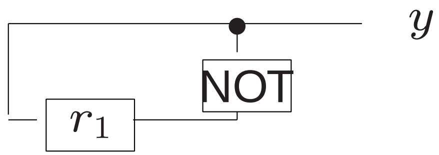
Let $\mathit{TS}_i = (S_i, \mathit{Act}, \rightarrow_i, I_i, \mathit{AP}_i, L_i)$ and $\mathit{Act} \times \mathit{Act} \rightarrow \mathit{Act}, \quad (\alpha, \beta) \rightarrow \alpha * \beta$

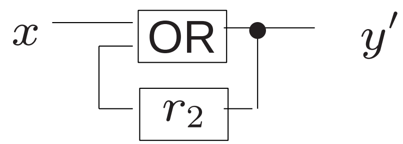
$$\mathit{TS}_1 \otimes \mathit{TS}_2 = (S_1 \times S_2, \mathit{Act}, \rightarrow, I_1 \times I_2, \mathit{AP}_1 \uplus \mathit{AP}_2, L)$$

with L as defined before and $\rightarrow$ is defined by the following rule:

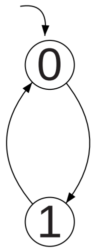
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2' \rangle}$$

typically used for synchronous hardware circuits, cf. next example

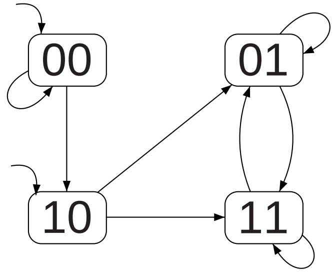




TS_1 :



TS_2 :



$TS_1 \otimes TS_2$: