Modeling Concurrent Systems

Hao Zheng

Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
Email: haozheng@usf.edu
Phone: (813)974-4757
Fax: (813)974-5456

Overview

1. **Modeling Formalisms**
   - Transition Systems
   - Modeling HW
   - Modeling SW
2. **Parallel Composition**
   - Composing Independent Processes
   - Composing Concurrent Processes: Shared Variables
   - Composing Concurrent Processes: Handshaking
   - Synchronous Composition
3. **Understanding State Space Explosion**

Reading

Principle of Model Checking, Chapter 2

Contents

2.1 Transition Systems

Transition system is a common semantic model to describe computation/communcation in HW/SW systems.

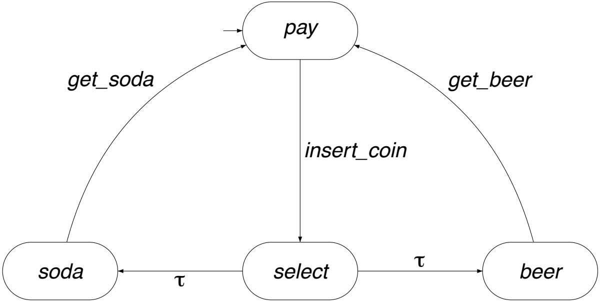
Definition 2.1 Transition Systems

A transition system TS is a tuple $\langle S, Act, \longrightarrow, I, AP, L\rangle$ where:

- S is a set of states.
- *Act* is a set of actions.
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation (denoted $s \xrightarrow{\alpha} s'$).
- $I \subseteq S$ is a set of initial states.
- *AP* is a set of atomic propositions.
- $L : S \to 2^{AP}$ is a labeling function.

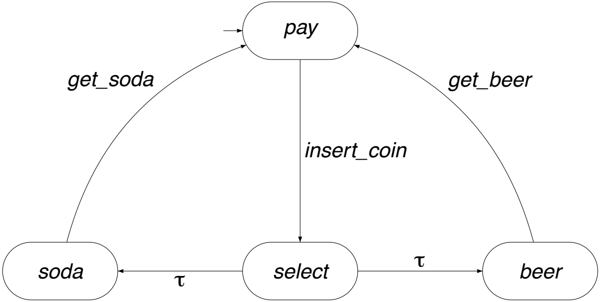
- Note that S and *Act* can be finite or countably infinite.

Example 2.2 Beverage Vending Machine



- $S = \{\text{pay}, \text{select}, \text{soda}, \text{beer}\}$
- $Act = \{\text{insert\_coin}, \text{get\_soda}, \text{get\_beer}, \tau\}$
- $I = \{\text{pay}\}$
- $AP = S$
- $L(s) = \{s\}$

Example 2.2 Beverage Vending Machine



- $S = \{\text{pay}, \text{select}, \text{soda}, \text{beer}\}$
- $Act = \{\text{insert\_coin}, \text{get\_soda}, \text{get\_beer}, \tau\}$
- $I = \{\text{pay}\}$
- $AP = \{\text{paid}, \text{drink}\}$
- $L(\text{pay}) = \emptyset, L(\text{select}) = \{paid\}, L(\text{soda}) = L(\text{beer}) = \{\text{paid, drink}\}$

The Role of Nondeterminism

- Used to model concurrency by interleaving.
  - No assumption about the relative speed of processes.
- Used to model implementation freedom.
  - Only describes what a system should do, not how.
- Used to model under-specified systems, or abstractions of real systems.
  - Use incomplete information.

Definition 2.3 Direct Successors and Predecessors

$$\mathit{Post}(s,\alpha) = \left\{ s' \in S \mid s \xrightarrow{\alpha} s' \right\}, \quad \mathit{Post}(s) = \bigcup_{\alpha \in \mathit{Act}} \mathit{Post}(s,\alpha)$$

$$\mathit{Pre}(s,\alpha) = \left\{ s' \in S \mid s' \xrightarrow{\alpha} s \right\}, \quad \mathit{Pre}(s) = \bigcup_{\alpha \in \mathit{Act}} \mathit{Pre}(s,\alpha).$$

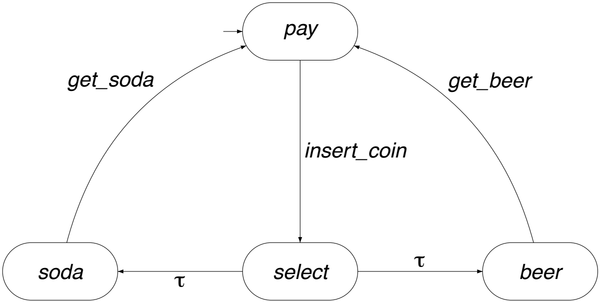
$$\mathit{Post}(C,\alpha) = \bigcup_{s \in C} \mathit{Post}(s,\alpha), \quad \mathit{Post}(C) = \bigcup_{s \in C} \mathit{Post}(s) \text{ for } C \subseteq S.$$

$$\mathit{Pre}(C,\alpha) = \bigcup_{s \in C} \mathit{Pre}(s,\alpha), \quad \mathit{Pre}(C) = \bigcup_{s \in C} \mathit{Pre}(s) \text{ for } C \subseteq S.$$

Definition 2.4 Terminal State

State s is called *terminal* if and only if $\mathit{Post}(s) = \emptyset$.

Successors and Predecessors: Example



- $Post(pay, insert\_coin) = \{select\}$
- $Pre(pay, get\_soda) = \{soda\}$
- $Pre(pay) = \{soda, beer\}$

Definition 2.5 Deterministic Transition Systems

- Transition system $TS = (S, Act, \to, I, AP, L)$ is *action-deterministic* iff:

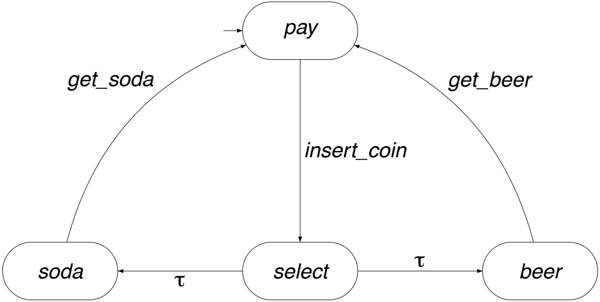
$$|I| \leq 1 \quad \text{and} \quad |Post(s, \alpha)| \leq 1 \quad \text{for all } s, \alpha$$

  - No more than 2 successor states due to the same action
- Transition system $TS = (S, Act, \to, I, AP, L)$ is *AP-deterministic* iff:

$$|I| \leq 1 \ \text{and} \ |\underbrace{Post(s) \cap \{ s' \in S \mid L(s') = A \}}_{\text{equally labeled successors of } s}| \leq 1 \quad \text{for all } s, A \in 2^{AP}$$

  - No more than 2 successor states of same labeling

Deterministic Transition Systems: Example



- Is this TS action-deterministic?

2.1.1 Executions

- An execution (run) is a linear sequence of state transitions.
- Used to describe dynamic behavior of transition systems.

Definition 2.6 Execution Fragments

- A *finite execution fragment* ρ of *TS* is an alternating sequence of states and actions ending with a state:

$$\rho = s_0 \alpha_1 s_1 \alpha_2 \ldots \alpha_n s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i < n.$$

- An *infinite execution fragment* ρ of *TS* is an infinite, alternating sequence of states and actions:

$$\rho = s_0 \alpha_1 s_1 \alpha_2 s_2 \alpha_3 \ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i.$$

2.1.1 Executions

Definition 2.7 Maximal and Initial Execution

An *execution* of *TS* is an *initial*, *maximal* execution fragment

- An execution fragment is *initial* if $s_0 \in I$.
- A maximal execution fragment can be finite, ending in a terminal state, or infinite.

Definition 2.9 Executions

An *execution* of transition system TS is an initial, maximal execution fragment.

Example 2.8 Executions of the Vending Machine

$$\rho_1 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} \ldots$$

$$\rho_2 = select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots$$

$$\rho_3 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda$$

- Which execution fragments are initial?

Example 2.8 Executions of the Vending Machine

$$\rho_1 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} \ldots$$

$$\rho_2 = select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots$$

$$\rho_3 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda$$

- Which execution fragments are initial? ρ_1 and ρ_3

Example 2.8 Executions of the Vending Machine

$$\rho_1 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} \ldots$$

$$\rho_2 = select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots$$

$$\rho_3 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda$$

- Which execution fragments are initial? ρ_1 and ρ_3
- Which execution fragments are maximal?

Example 2.8 Executions of the Vending Machine

$$\rho_1 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} \ldots$$

$$\rho_2 = select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots$$

$$\rho_3 = pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda$$

- Which execution fragments are initial? ρ_1 and ρ_3
- Which execution fragments are maximal? ρ_1 and ρ_2

Example 2.8 Executions of the Vending Machine

$$\begin{aligned}
\rho_1 &= pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} \ldots \\
\rho_2 &= select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots \\
\rho_3 &= pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda
\end{aligned}$$

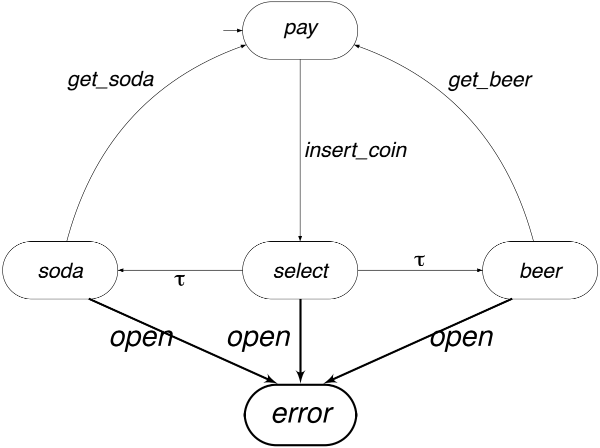
- Which execution fragments are initial? ρ_1 and ρ_3
- Which execution fragments are maximal? ρ_1 and ρ_2
- Which execution fragments are “executions”?

Example 2.8 Executions of the Vending Machine

$$\begin{aligned}
\rho_1 &= pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} \ldots \\
\rho_2 &= select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots \\
\rho_3 &= pay \xrightarrow{coin} select \xrightarrow{\tau} soda \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} soda
\end{aligned}$$

- Which execution fragments are initial? ρ_1 and ρ_3
- Which execution fragments are maximal? ρ_1 and ρ_2
- Which execution fragments are “executions”? ρ_1

Executions: Another Example



- An execution

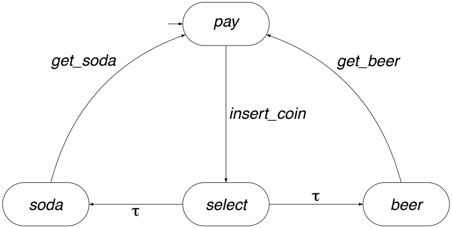
$$\rho_4 = pay \xrightarrow{insert\_coin} select \xrightarrow{open} error$$

Definition 2.10 Reachable States

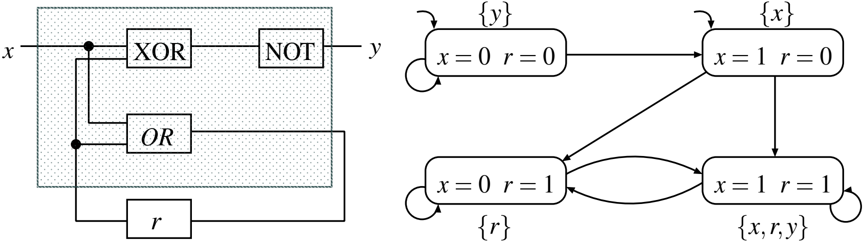
- State $s \in S$ is called *reachable* in *TS* if there exists an initial, finite execution fragment (execution)

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

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



2.1.2 Modeling Sequential Circuits



- Transition system representation of a simple hardware circuit.
- Input variable x, output variable y, and register r.
- Output function $\neg(x \oplus r)$ and register evaluation function $x \vee r$.
- Actions in Act are irrelevant here.

Atomic Propositions

Consider three possible state-labelings:

- Let $AP = \{x, y, r\}$
  - $L(\langle x=0, r=1\rangle) = \{r\}$ and $L(\langle x=1, r=1\rangle) = \{x, r, y\}$
  - $L(\langle x=0, r=0\rangle) = \{y\}$ and $L(\langle x=1, r=0\rangle) = \{x\}$
  - Property e.g., “once the register is one, it remains one”
- Let $AP' = \{x, y\}$ – the register evaluations are now “invisible”
  - $L(\langle x=0, r=1\rangle) = \emptyset$ and $L(\langle x=1, r=1\rangle) = \{x, y\}$
  - $L(\langle x=0, r=0\rangle) = \{y\}$ and $L(\langle x=1, r=0\rangle) = \{x\}$
  - Property e.g., “the output bit y is set infinitely often”
- Let $AP' = \{x, r\}$ – output y can be derived from x and r.
  - $L(\langle x=0, r=1\rangle) = \{\qquad\}$ and $L(\langle x=1, r=1\rangle) = \{\qquad\}$
  - $L(\langle x=0, r=0\rangle) = \{\qquad\}$ and $L(\langle x=1, r=0\rangle) = \{\qquad\}$
  - How to check “the output bit y is set infinitely often”?

Atomic Propositions

Consider three possible state-labelings:

- Let $AP = \{x, y, r\}$
  - $L(\langle x = 0, r = 1\rangle) = \{r\}$ and $L(\langle x = 1, r = 1\rangle) = \{x, r, y\}$
  - $L(\langle x = 0, r = 0\rangle) = \{y\}$ and $L(\langle x = 1, r = 0\rangle) = \{x\}$
  - Property e.g., “once the register is one, it remains one”
- Let $AP' = \{x, y\}$ – the register evaluations are now “invisible”
  - $L(\langle x = 0, r = 1\rangle) = \emptyset$ and $L(\langle x = 1, r = 1\rangle) = \{x, y\}$
  - $L(\langle x = 0, r = 0\rangle) = \{y\}$ and $L(\langle x = 1, r = 0\rangle) = \{x\}$
  - Property e.g., “the output bit y is set infinitely often”
- Let $AP' = \{x, r\}$ – output y can be derived from x and r.
  - $L(\langle x = 0, r = 1\rangle) = \{\qquad\}$ and $L(\langle x = 1, r = 1\rangle) = \{\qquad\}$
  - $L(\langle x = 0, r = 0\rangle) = \{\qquad\}$ and $L(\langle x = 1, r = 0\rangle) = \{\qquad\}$
  - How to check “the output bit y is set infinitely often”?
    - Convert to check “$\neg(x \oplus r)$ holds infinitely often"

Sequential Circuit Representation

A sequential circuit is typically represented in an intermediate format below before its TS is derived.

$$Cir = (X, Reg, I, R, AP, L)$$

where

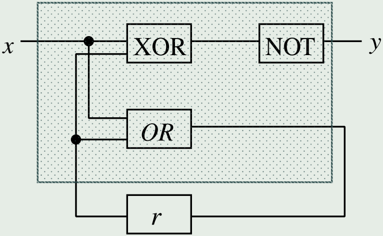
- X is a set of input variables.
- Reg is a set of registers.
- $I = \{c_{0,1}, \ldots, c_{0,k}\}$: a set of initial states. $-$ values assigned to Reg
- R is the transition relation of the following form

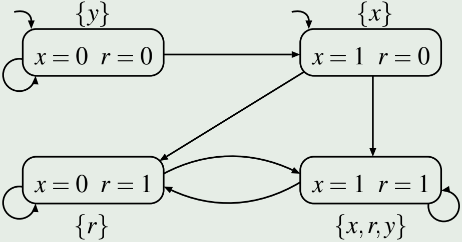
$$\bigwedge_{r_i \in Reg} r_i' = f(x_1, \ldots, x_n, r_1, \ldots, r_k)$$

where r_i' represents the value of r_i in the next state.

Sequential Circuit Representation – Example

How to represent the previous circuit example and find it TS?





2.1.2 Modeling SW: Program Graphs

- How to model the following construct?

```
if x\%2 = 1 then
   x := x+1;
else
   x := 2 * x
```

- Two modeling issues:
  - Data variables
  - Data-dependent control

2.1.2 Modeling SW: Program Graphs

Definition 2.13 Program Graphs

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

$$\langle Loc, Act, Effect, \hookrightarrow, Loc_0, g_0 \rangle \quad \text{where}$$

- *Loc* is a set of *locations* with initial locations $Loc_0 \subseteq Loc$
- *Act* is a set of actions
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$ is the *effect* function
- $\hookrightarrow \subseteq Loc \times \underbrace{Cond(Var)}_{\text{Boolean conditions over } Var} \times Act \times Loc$, is the transition relation
- $g_0 \in Cond(Var)$ is the initial *condition*.

Notation: $\ell \xrightarrow{g:\alpha} \ell'$ denotes $(\ell, g, \alpha, \ell') \in \hookrightarrow$

Example 2.12 − Beverage VM Revisited

Suppose the VM keeps track of number of beer or soda bottles sold.

- $\mathit{Loc} = \{\, start, select \,\}$ with $\mathit{Loc}_0 = \{\, start \,\}$
- $\mathit{Act} = \{\, bget, sget, coin, ret\_coin, refill \,\}$
- $\mathit{Var} = \{\, nsoda, \;\; nbeer \,\}$ with domain $\{\, 0, 1, \ldots, max \,\}$
- $g_0 = (nsoda = max \;\wedge\; nbeer = max)$

Example 2.12 – Beverage VM Revisited

- Transition relation $\hookrightarrow$ is

$$start \xhookrightarrow{true:coin} select \quad \text{and} \quad start \xhookrightarrow{true:refill} start$$

$$select \xhookrightarrow{nsoda>0:sget} start \quad \text{and} \quad select \xhookrightarrow{nbeer>0:bget} start$$

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

- Effects of actions

Action	Effect on variables
$coin$	
ret_coin	
$sget$	$nsoda := nsoda - 1$
$bget$	$nbeer := nbeer - 1$
$refill$	$nsoda := max; nbeer := max$

Definition 2.15 Transition Systems for Program Graphs

The transition system $\mathit{TS}(\mathit{PG})$ of program graph

$$\mathit{PG} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0)$$

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

- $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$
- $\longrightarrow \subseteq S \times Act \times S$ is defined by the rule:

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

- $I = \{\langle \ell, \eta \rangle \mid \ell \in \mathit{Loc}_0, \eta \models g_0\}$
- $\mathit{AP} = \{/*$ property dependent $*/\}$
- $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{g \in \mathit{Cond}(\mathit{Var}) \mid \eta \models g\}$.

Transition System for Beverage Machine

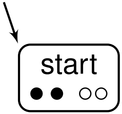
$$start \xhookrightarrow[coin]{true} select$$

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

$$select \xhookrightarrow[sget]{nsoda>0:} start$$

$$select \xhookrightarrow[bget]{nbeer>0:} start$$

$$select \xhookrightarrow[ret\_coin]{nsoda=0 \land nbeer=0:} start$$



Transition System for Beverage Machine

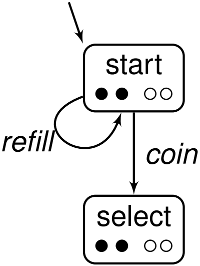
$$start \xhookrightarrow[coin]{true} select$$

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

$$select \xhookrightarrow[sget]{nsoda>0:} start$$

$$select \xhookrightarrow[bget]{nbeer>0:} start$$

$$select \xhookrightarrow[ret\_coin]{nsoda=0 \land nbeer=0:} start$$



Transition System for Beverage Machine

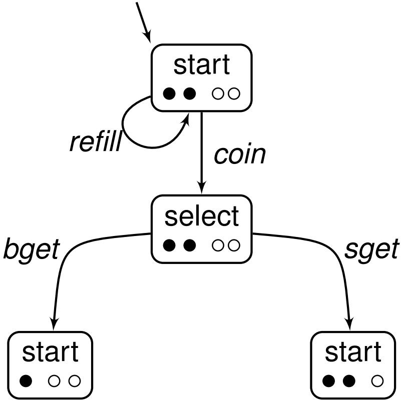
$$start \xhookrightarrow[coin]{true} select$$

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

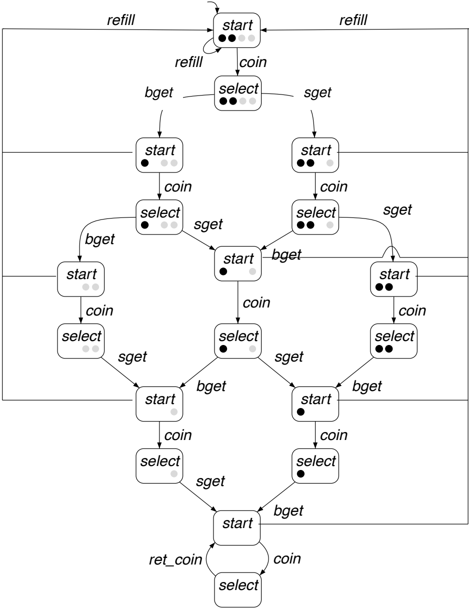
$$select \xhookrightarrow[sget]{nsoda>0:} start$$

$$select \xhookrightarrow[bget]{nbeer>0:} start$$

$$select \xhookrightarrow[ret\_coin]{nsoda=0 \wedge nbeer=0:} start$$



Transition System for Beverage Machine



From Promela to Program Graphs

```
bool turn, flag[2];
byte ncrit;

active [2] proctype user()
{
        assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);

        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;

        flag[_pid] = 0;
        goto again
}
```

From Promela to Program Graphs

```
bool turn, flag[2];
byte ncrit;

active [2] proctype user()
{
l1:     assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
l2:     turn = _pid;
l3:     (flag[1 - _pid] == 0 || turn == 1 - _pid);

l4:     ncrit++;
l5:     assert(ncrit == 1); /* critical section */
l6:     ncrit--;

l7:     flag[_pid] = 0;
l8:     goto again
}
```

Contents

2.2 Parallelism and Communications

- Transition systems can model:
  - Sequential data-dependent systems.
  - Sequential hardware circuits.
- How about *concurrent* systems?
  - Multi-threading with shared variables.
  - Parallel distributed algorithms.
  - Synchronous/asynchronous communication protocols.
  - Synchronous/asynchronous composition of hardware.
- Parallel composition $\|$

$$TS = TS_1 \| TS_2 \| \ldots \| TS_n$$

2.2.1 Concurrency and Interleaving

- *Interleaving* is a widely accepted paradigm for parallel systems.
- Actions of independent components are merged or “interleaved”.
- No assumptions are made on the order of process executions.
- 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.
  - Scheduling needs to fair.

Definition 2.18 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

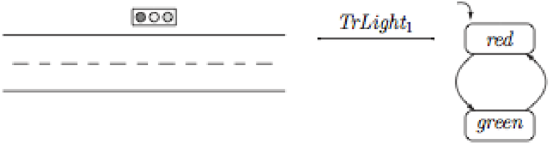
$$\mathit{TS}_1 \,|||\, \mathit{TS}_2 \;=\; (S_1 \times S_2, \mathit{Act}_1 \cup \mathit{Act}_2, \longrightarrow, I_1 \times I_2, \mathit{AP}_1 \cup \mathit{AP}_2, L)$$

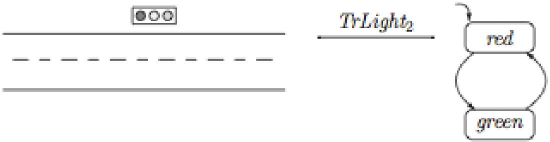
where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\longrightarrow$ is defined by the rules:

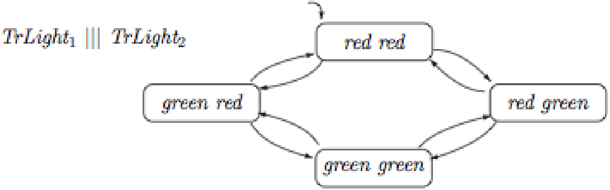
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \quad \text{and} \quad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

TS_1 and TS_2 are assumed *independent*, ie, no shared actions or variables.

Two Independent Traffic Lights







Justification for Interleaving

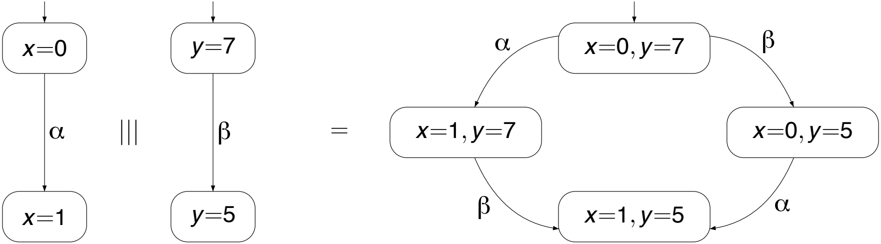
- The effect of concurrently executed, independent actions α and β equals the effect when α and β are successively executed in arbitrary order
- Symbolically this is stated as:

$$\begin{aligned} \mathit{Effect}(\alpha \,|||\, \beta, \eta) &= \mathit{Effect}((\alpha\,;\beta) + (\beta\,;\alpha), \eta) \\ &= \mathit{Effect}((\alpha\,;\beta), \eta) \\ &= \mathit{Effect}((\beta\,;\alpha), \eta) \end{aligned}$$

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

Another Interleaving Example

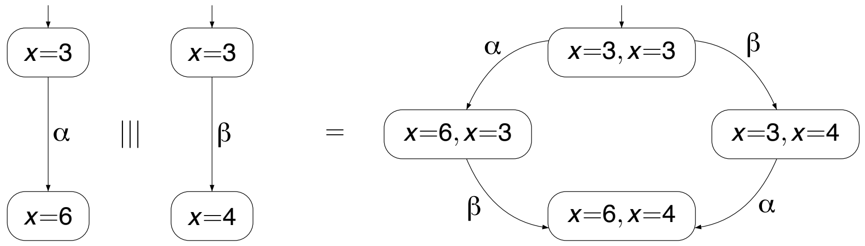
$\underbrace{x := x+1}_{=\alpha} \,|||\, \underbrace{y := y-2}_{=\beta}$ with initially $x = 0$ and $y = 7$



2.2.2 Communication via Shared Variables

Example 2.20

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



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

$\Rightarrow$ Not a faithful model of the concurrent execution of α and β

Interleaving Program Graphs

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

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

Interleaving of transition systems

- If PG_1 and PG_2 share some variables (i.e., $Var_1 \cap Var_2 \neq \emptyset$):

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

Interleaving of program graphs

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

Definition 2.21 Interleaving of Program Graphs

- Let $PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$ over variables Var_i.
- Program graph $PG_1 \,|||\, PG_2$ over $Var_1 \cup Var_2$ is defined by:

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

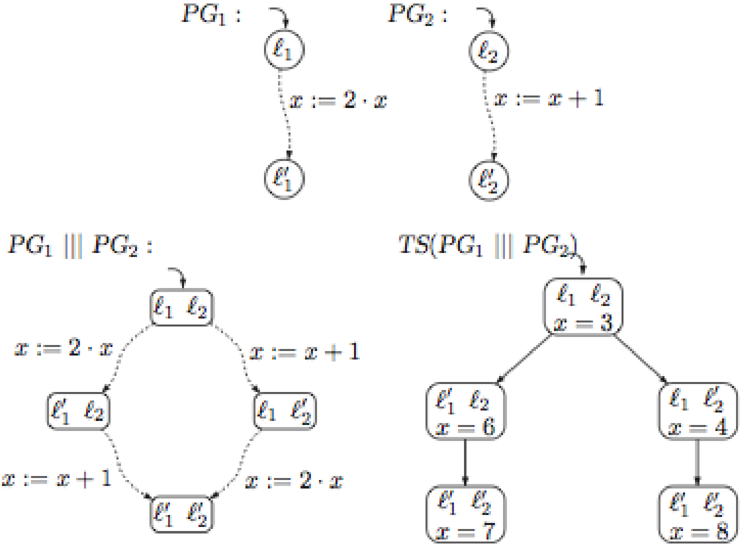
where $\hookrightarrow$ is defined by the inference rules:

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

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

For PG_1 and PG_2, $Loc_1 \cap Loc_2 = \emptyset$ and $Act_1 \cap Act_2 = \emptyset$.

Example 2.22 Interleaving of Program Graphs



Critical and Noncritical Actions

- Actions that access shared variables are *critical*, otherwise they are *noncritical*.
- Nondeterminism in a state may be due to:
  - An internal nondeterministic choice within program graph PG_1 or PG_2.
  - The interleaving of noncritical actions of PG_1 and PG_2.
  - The resolution of a contention between critical actions of PG_1 and PG_2 (concurrency).
- A noncritical action can be executed in parallel with any other action.
- The schedule of concurrent critical actions affects the global state.
  - Different order of executions of critical actions may lead to different states.

On Atomicity

- Atomicity is used to capture granularity of concurrency.
- Actions $\alpha \in Act$ are consider indivisible.

$$\langle x := x + 1;\ y := 2x + 1;\ \textbf{if}\ x < 12\ \textbf{then}\ z := (x - z)^2 * y\ \textbf{fi} \rangle$$

Banking System

Person Left behaves as follows:

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

Person Right behaves as follows:

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

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

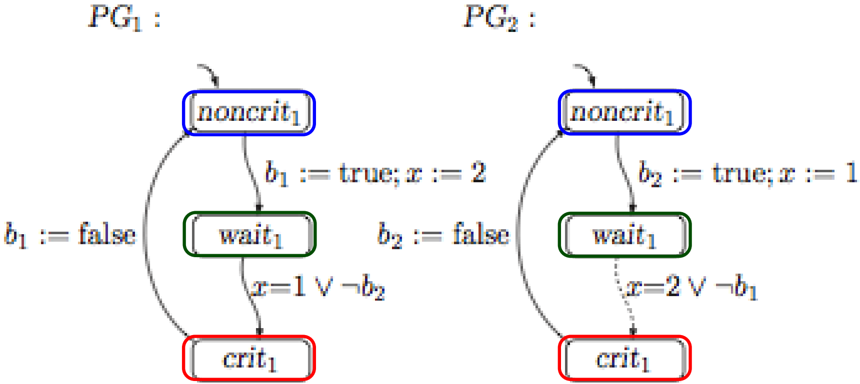
Peterson's Mutual Exclusion Algorithm

```
$P_1$   loop forever
            ⋮                                   (* non-critical actions *)
            $\langle b_1 := \text{true};\ x := 2 \rangle$;                   (* request *)
            wait until $(x = 1 \lor \neg b_2)$
            do critical section od
            $b_1 := \text{false}$                          (* release *)
            ⋮                                   (* non-critical actions *)
        end loop
```

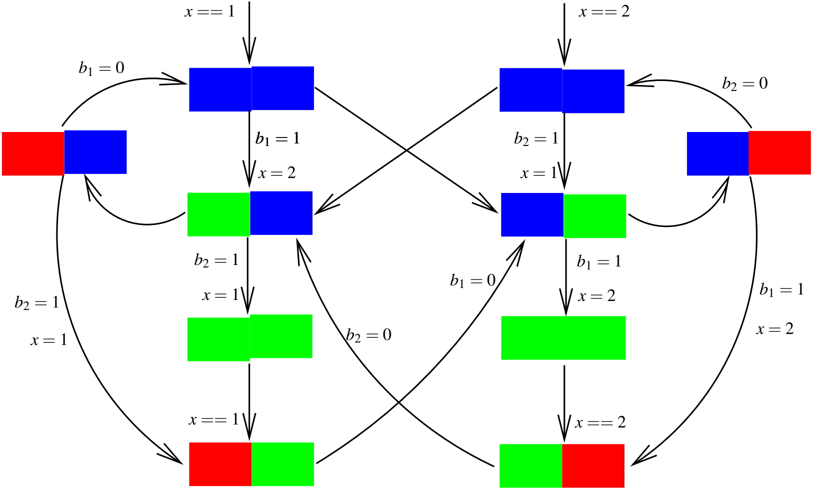
b_i is true if and only if process P_i is waiting or in critical section

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

Program Graph Representation



Transition System



Is mutual exclusion guaranteed?

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$1 : \quad \langle nc_1, \quad nc_2, \quad x = 1, \quad b_1 = \text{false}, \quad b_2 = \text{false} \rangle$$

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$2:\quad \langle nc_1,\quad rq_2,\quad x = 1,\quad b_1 = \text{false},\quad b_2 = \text{false}\rangle$$

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$3 : \quad \langle rq_1, \quad rq_2, \quad x = 2, \quad b_1 = \text{false}, \quad b_2 = \text{false} \rangle$$

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$4: \quad \langle wt_1, \quad rq_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{false} \rangle$$

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$5 : \quad \langle cs_1, \quad rq_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{false} \rangle$$

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$6 : \quad \langle cs_1, \quad wt_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{true} \rangle$$

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$7: \quad \langle cs_1, \quad cs_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{true} \rangle$$

Violation of the mutual exclusion property!

Banking System with Non-atomic Assignment

Person Left behaves as follows:

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

Person Right behaves as follows:

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

$$7: \quad \langle cs_1, \quad cs_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{true} \rangle$$

Violation of the mutual exclusion property!

Note that protocol is okay if b_i is assigned before x.

2.2.3 Handshaking

- If processes are distributed there is no shared memory.
- Communications for distributed systems:
  - Synchronous message passing (= handshaking)
  - Asynchronous message passing (= channel communication)
- Concurrent processes interact by *synchronous message passing*.
  - Processes execute synchronized actions together at the same time.
  - The interacting processes "shake hands".
- This does NOT mean it is implemented with synchronous hardware.
- Introduce set H, the *handshake actions*.
  - Actions outside H are independent and are interleaved.
  - Actions in H need to be synchronized.
  - Abstracts away the information that is exchanged.

Handshaking: Formal Definition

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

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

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

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

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

TS_1 and TS_2 do NOT share variables.

Handshaking Properties

- For an empty set of handshake actions:

$$TS_1 \parallel_{\emptyset} TS_2 = TS_1 \mid\mid\mid TS_2$$

.

- Note that it is commutative (i.e., $TS_1 \parallel_H TS_2 = TS_2 \parallel_H TS_1$), but
- Not always associative, i.e.,

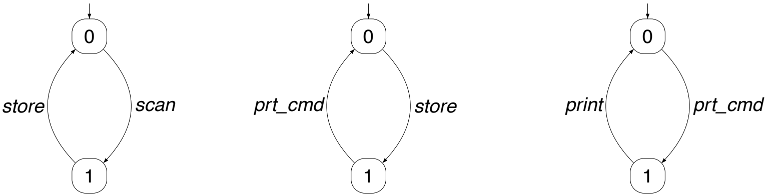
$$(TS_1 \parallel_{H_1} TS_2) \parallel_{H_2} TS_3 \neq TS_1 \parallel_{H_1} (TS_2 \parallel_{H_2} TS_3).$$

- It is, however, associative for a fixed set H:

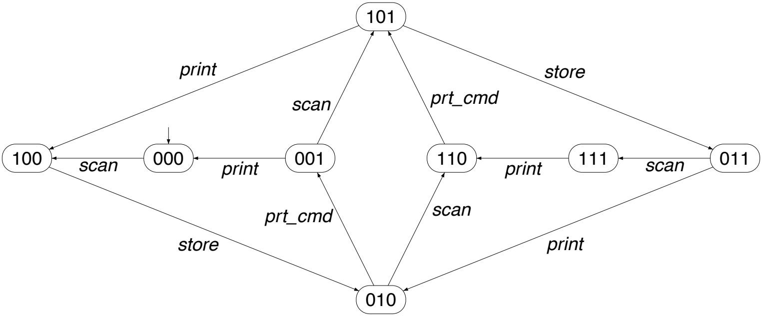
$$TS = TS_1 \parallel_H TS_2 \parallel_H \ldots \parallel_H TS_n.$$

  - Useful to model broadcast communications.

Example 2.28 A Booking System



BCR ∥ *BP* ∥ *Printer* (∥ is a shorthand for $\|_H$ with $H = \mathit{Act}_1 \cap \mathit{Act}_2$)



2.2.6 Synchronous Parallelism

Definition 2.41 Synchronous Product

- Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, $i{=}1,2$, the *synchronous product* of TS_1 and TS_2, $TS_1 \otimes TS_2$, is given by

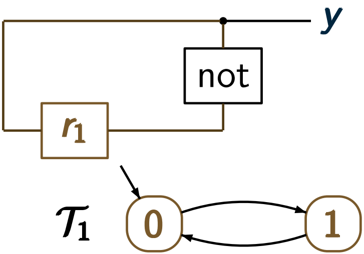
$$TS_1 \otimes TS_2 \;=\; (S_1 \times S_2, Act_1 \times Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

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

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

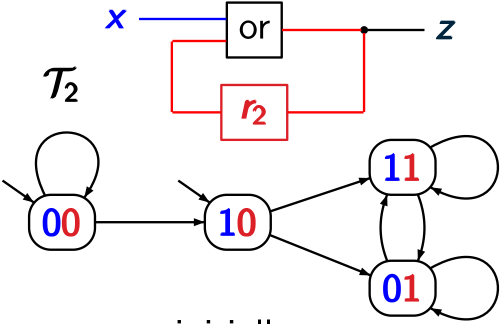
- Often used for composing synchronous digital circuits.

Synchronous Product: Example



initially:
$r_1 = 0$

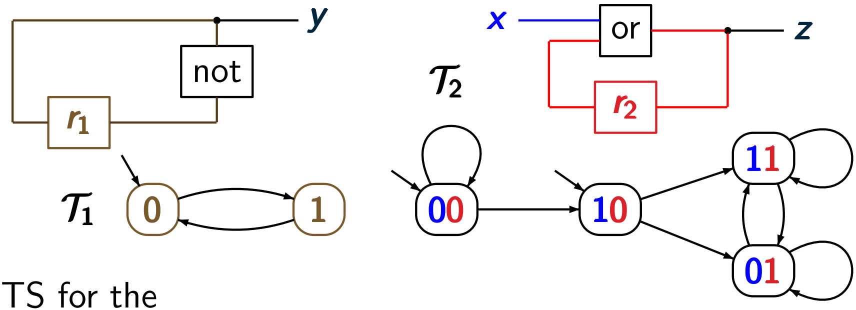
transition function:
$\delta_{r_1} = \neg r_1$



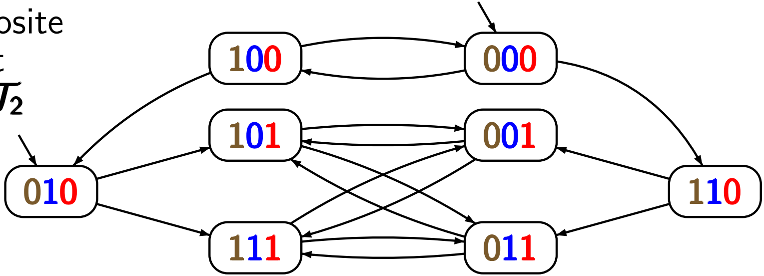
initially:
$r_2 = 0$

transition function:
$\delta_{r_2} = r_2 \vee x$

Synchronous Product: Example



TS for the composite circuit $\mathcal{T}_1 \otimes \mathcal{T}_2$



Contents

2.3 State Explosion

- Given a program graph, the number of states is

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

- Consider $TS = TS_1 \| \ldots \| TS_n$, the number of states is

$$|S_1| \cdot \ldots \cdot |S_n|$$

Summary

- Transition systems
  - A fundamental model for modeling software and hardware systems.
- Executions
  - Alternating sequences of states and actions that cannot be prolonged.
- Interleaving
  - Execution of independent concurrent processes by nondeterminism.
- Shared variables
  - Parallel composition on transition systems is not adequate.
  - Instead, parallel composition of program graphs is used.
- Handshaking on a set H of actions
  - Execute actions in H simultaneously and those not in H autonomously.