Verification

Lecture 2

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

REVIEW: Transition systems

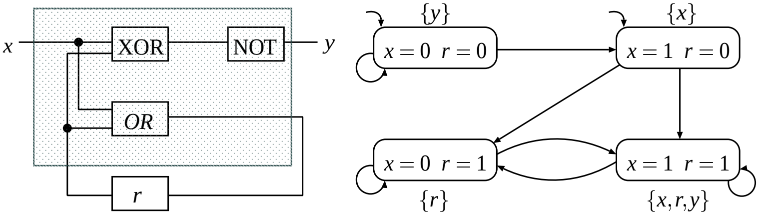
A transition system TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where

- S is a set of states
- Act is a set of actions
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation
- $I \subseteq S$ is a set of initial states
- AP is a set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a labeling function

S and Act are either finite or countably infinite

Notation: $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \longrightarrow$

REVIEW: 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 $x \vee r$

Modeling data-dependent systems

The beverage vending machine revisited:

"Abstract" transitions:

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

$$\mathit{select} \xrightarrow{\mathit{nsprite}>0:\mathit{sget}} \mathit{start} \quad \text{and} \quad \mathit{select} \xrightarrow{\mathit{nbeer}>0:\mathit{bget}} \mathit{start}$$

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

Action	Effect on variables
coin	
ret_coin	
sget	$\mathit{nsprite} := \mathit{nsprite} - 1$
bget	$\mathit{nbeer} := \mathit{nbeer} - 1$
refill	$\mathit{nsprite} := \mathit{max}; \mathit{nbeer} := \mathit{max}$

Some preliminaries

- typed variables with a valuation that assigns values to variables
  - e.g., $\eta(x) = 17$ and $\eta(y) = -2$
- the set of Boolean conditions over *Var*
  - propositional logic formulas whose propositions are of the form "$\overline{x} \in \overline{D}$"
  - $(-3 < x \leq 5) \land (y = green) \land (x \leq 2 \cdot x')$
- effect of the actions is formalized by means of a mapping:

$$Effect : Act \times Eval(Var) \rightarrow Eval(Var)$$

  - e.g., $\alpha \equiv x := y+5$ and evaluation $\eta(x) = 17$ and $\eta(y) = -2$
  - $Effect(\alpha, \eta)(x) = \eta(y) + 5 = 3$, and $Effect(\alpha, \eta)(y) = \eta(y) = -2$

Program graphs

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

$$(Loc, Act, Effect, \longrightarrow, Loc_0, g_0) \quad \text{where}$$

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

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

Beverage vending machine

- $Loc = \{ start, select \}$ with $Loc_0 = \{ start \}$
- $Act = \{ bget, sget, coin, ret\_coin, refill \}$
- $Var = \{ nsprite, \; nbeer \}$ with domain $\{ 0, 1, \ldots, max \}$
- $Effect$:

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

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

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: "ℓ" and "$x \in D$" for $D \subseteq dom(x)$
  - $\langle \ell, \eta \rangle$ is labeled with "ℓ" and all conditions that hold in η
- $\ell \xrightarrow{g:\alpha} \ell'$ and g holds in η then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle$

Structured operational semantics

- The notation $\dfrac{\text{premise}}{\text{conclusion}}$ means:

  If the premise holds, then the conclusion holds

- Such "if . . ., then . . ." propositions are also called <u>inference rules</u>
- If the premise is a tautology, it may be omitted (as well as the "solid line")
- In the latter case, the rule is also called an <u>axiom</u>

Transition systems for program graphs

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

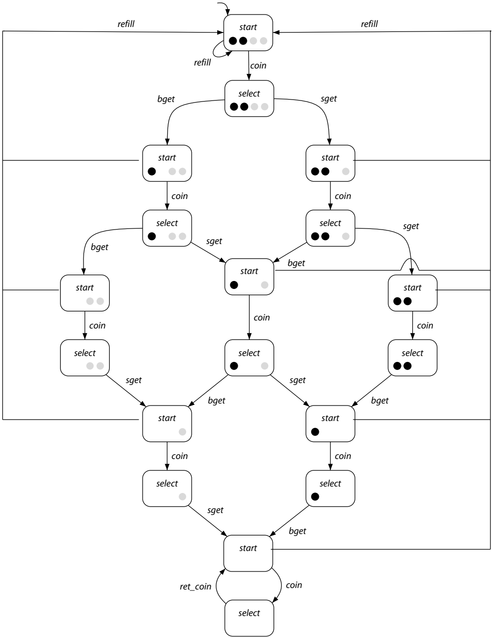
$$PG = (Loc, Act, Effect, \longrightarrow, Loc_0, g_0)$$

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

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

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

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


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

Transition systems ≠ finite automata

As opposed to finite automata, in a transition system:

- there are no accept states
- set of states and actions may be countably infinite
- may have infinite branching
- actions may be subject to synchronization
- nondeterminism has a different role

Transition systems are appropriate for reactive system behaviour

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 interleaved
- 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

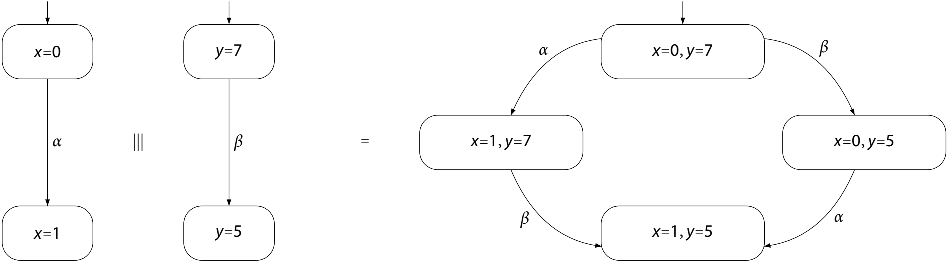
- Symbolically this is stated as:

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

  - $|||$ 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 $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$ $i{=}1, 2$, be two transition systems.

Transition system

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

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

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

Interleaving of program graphs

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

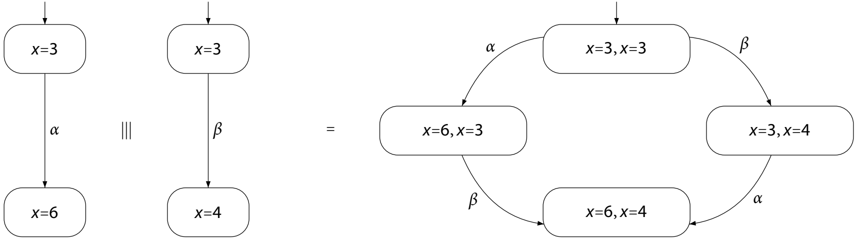
$$TS(PG_1) \mid\mid\mid TS(PG_2)$$

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

Idea: first interleave, then unfold

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 $PG_1 \mid\mid\mid PG_2$ over $Var_1 \cup Var_2$ is defined by:

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

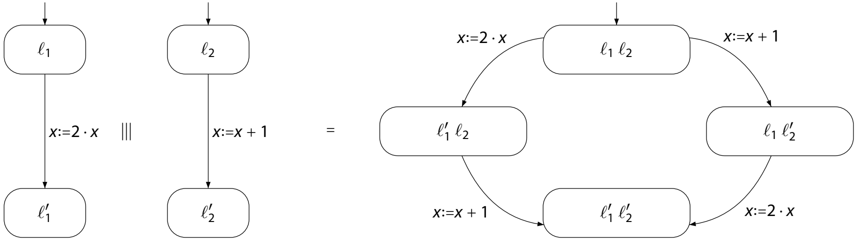
where $\longrightarrow$ is defined by the inference rules:

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

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

Example

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



note that $TS(PG_1) \,|||\, TS(PG_2) \neq TS(PG_1 \,|||\, PG_2)$

On atomicity

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

Possible execution fragment:

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

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

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

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

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

Banking system

Person Left behaves as follows:

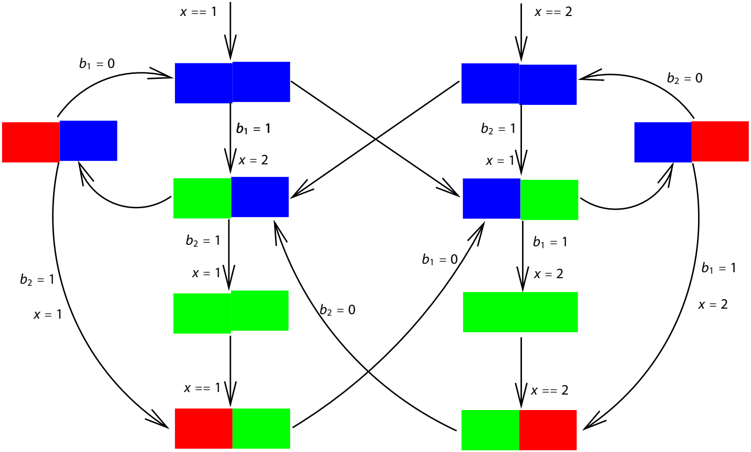
```
        while true {
            ......
nc:         $\langle b_1, x = \text{true}, 2; \rangle$
wt:         wait until $(x == 1 \,||\, \neg b_2)$ {
cs:             ...@account...}
            $b_1$ = false;
            ......
        }
```

Person Right behaves as follows:

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

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

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 {
          ......
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;
          ......
      }
```

On atomicity again

Possible state sequence:

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

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

⇒ 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:

- interleaving for $\alpha \notin H$:

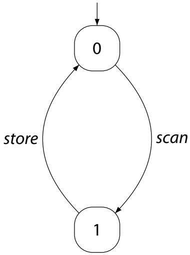
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

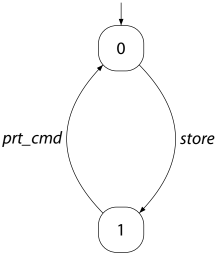
- handshaking for $\alpha \in H$:

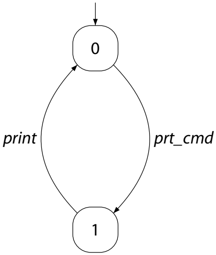
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle}$$

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



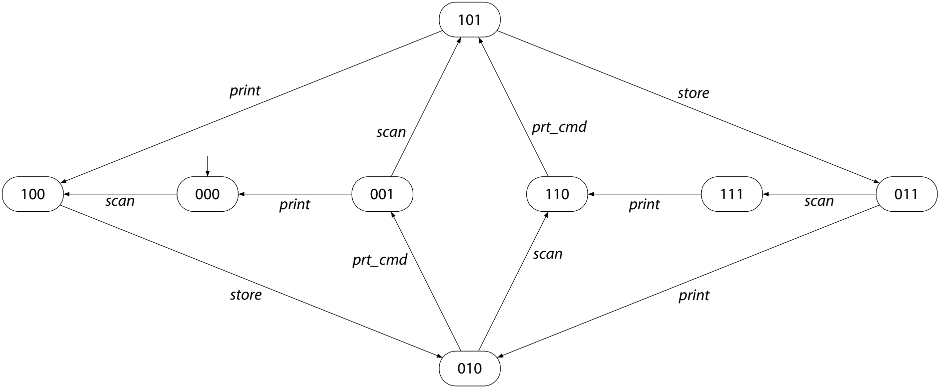




$$BCR \parallel BP \parallel Printer$$

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

The parallel composition



Pairwise handshaking

$TS_1 \parallel \ldots \parallel TS_n$ 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 $TS_1 \parallel \ldots \parallel TS_n$ is the Cartesian product of those of TS_i

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

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

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

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

Synchronous parallelism

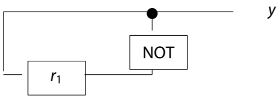
Let $TS_i = (S_i, Act, \rightarrow_i, I_i, AP_i, L_i)$ and $Act \times Act \rightarrow Act, \ (\alpha, \beta) \rightarrow \alpha * \beta$

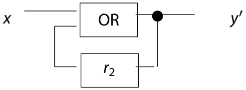
$$TS_1 \otimes TS_2 = (S_1 \times S_2, Act, \rightarrow, I_1 \times I_2, AP_1 \uplus AP_2, L)$$

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

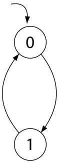
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2' \rangle}$$

typically used for synchronous hardware circuits, cf. next example

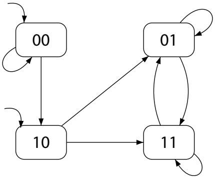




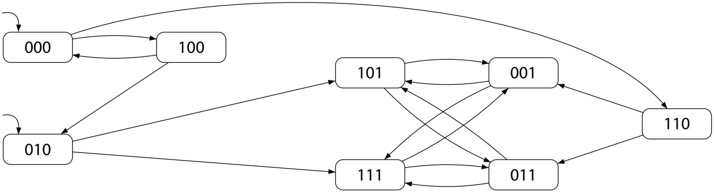
TS_1 :



TS_2 :



$TS_1 \otimes TS_2$:



Channels

- Processes communicate via channels ($c \in Chan$)
- Channels are first-in, first-out buffers
- Channels are types (wrt. their content --- $dom(c)$)
- Channels buffer messages (of appropriate type)
- Channel capacity = maximum # messages that can be stored
  - if $cap(c) \in \mathbb{N}$ then c is a channel with finite capacity
  - if $cap(c) = \infty$ then c has an infinite capacity
  - if $cap(c) > 0$, there is some "delay" between sending and receipt
  - if $cap(c) = 0$, then communication via c amounts to handshaking

Channels

- Process P_i = program graph PG_i + communication actions

  $c!v$ transmit the value v along channel c

  $c?x$ receive a message via channel c and assign it to variable x

- $Comm$ =
  $\{ c!v,\ c?x \mid c \in Chan,\ v \in dom(c),\ x \in Var.\ dom(x) \supseteq dom(c) \}$
- Sending and receiving a message
  - $c!v$ puts the value v at the rear of the buffer c (if c is not full)
  - $c?x$ retrieves the front element of the buffer and assigns it to x (if c is not empty)
  - if $cap(c) = 0$, channel c has no buffer
  - if $cap(c) = 0$, sending and receiving takes place simultaneously this is called synchronous message passing or handshaking
  - if $cap(c) > 0$, sending and receiving can never take place simultaneously this is called asynchronous message passing

Channel systems

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

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

where

$$\rightarrow \ \subseteq \ Loc \times (Cond(Var) \times Act) \times Loc \ \cup \ \underbrace{Loc \times Comm \times Loc}_{\text{communication actions}}$$

A channel system CS over $(\bigcup_{0<i\leq n} Var_i, Chan)$:

$$CS = [PG_1 \mid \ldots \mid PG_n]$$

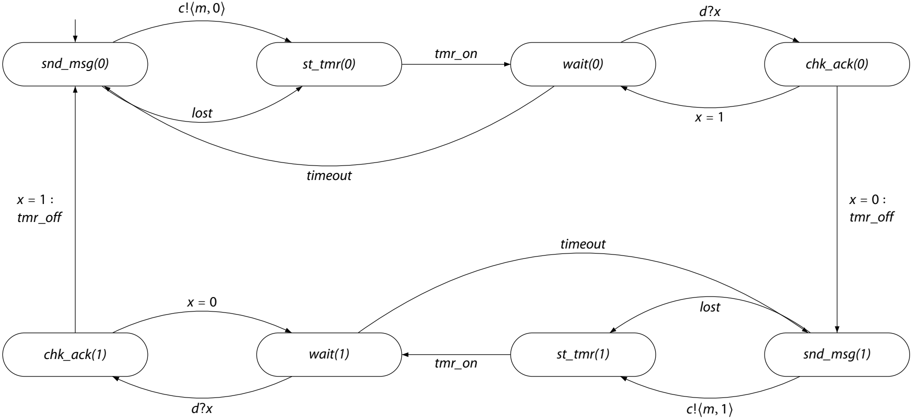
with program graphs PG_i over $(Var_i, Chan)$

Communication actions

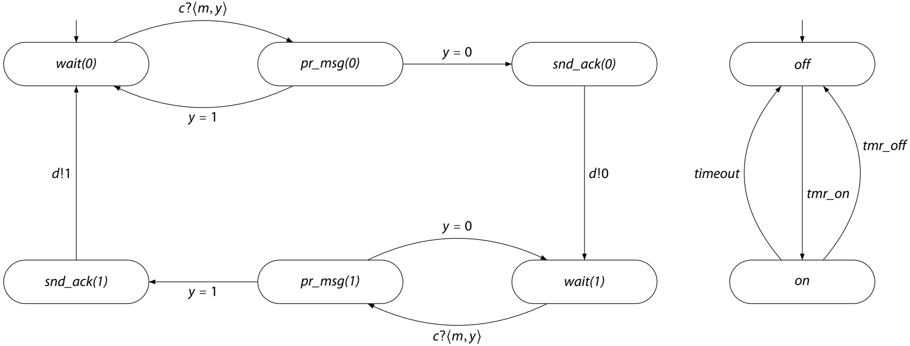
- Handshaking
  - if $cap(c) = 0$, then process P_i can perform $\ell_i \xrightarrow{c!v} \ell_i'$ only
  - . . . if P_j, say, can perform $\ell_j \xrightarrow{c?x} \ell_j'$
  - the effect corresponds to the (atomic) distributed assignment $x := v$.
- Asynchronous message passing
  - if $cap(c) > 0$, then process P_i can perform $\ell_i \xrightarrow{c!v} \ell_i'$
  - . . . if and only if less than $cap(c)$ messages are stored in c
  - P_j may perform $\ell_j \xrightarrow{c?v} \ell_j'$ if and only if the buffer of c is not empty
  - then the first element v of the buffer is extracted and assigned to x (atomically)

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

The alternating bit protocol: sender



The alternating bit protocol: receiver



Channel evaluations

- A channel evaluation ξ is
  - a mapping from channel $c \in \mathit{Chan}$ onto a sequence $\xi(c) \in \mathit{dom}(c)^*$ such that
  - current length cannot exceed the capacity of c: $\mathit{len}(\xi(c)) \leq \mathit{cap}(c)$
  - $\xi(c) = v_1\, v_2\, \ldots v_k$ ($\mathit{cap}(c) \geq k$) denotes v_1 is at front of buffer etc.
- $\xi[c := v_1 \ldots v_k]$ denotes the channel evaluation

$$\xi[c := v_1 \ldots v_k](c') = \begin{cases} \xi(c') & \text{if } c \neq c' \\ v_1 \ldots v_k & \text{if } c = c'. \end{cases}$$

- Initial channel evaluation ξ_0 equals $\xi_0(c) = \varepsilon$ for any c

Transition system semantics of a channel system

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

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

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

- $S = (Loc_1 \times \cdots \times Loc_n) \times Eval(Var) \times Eval(Chan)$
- $Act = (\biguplus_{0<i\leq n} Act_i) \uplus \{ \tau \}$
- $\rightarrow$ is defined by the inference rules on the next slides
- $I = \left\{ \langle \ell_1, \ldots, \ell_n, \eta, \xi_0 \rangle \mid \forall i.\ (\ell_i \in Loc_{0,i} \ \&\ \eta \models g_{0,i}) \ \&\ \forall c.\ \xi_0(c) = \varepsilon \right\}$
- $AP = \biguplus_{0<i\leq n} Loc_i \uplus Cond(Var)$
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle) = \{ \ell_1, \ldots, \ell_n \} \cup \{ g \in Cond(Var) \mid \eta \models g \}$

Inference rules (I)

- Interleaving for $\alpha \in Act_i$:

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

  where $\eta' = \mathit{Effect}(\alpha, \eta)$
- Synchronous message passing over $c \in \mathit{Chan}$, $\mathit{cap}(c) = 0$:

$$\frac{\ell_i \xrightarrow{c?x} \ell_i' \quad \wedge \quad \ell_j \xrightarrow{c!v} \ell_j' \quad \wedge \quad i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle}$$

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

Inference rules (II)

- Asynchronous message passing for $c \in Chan$, $cap(c) > 0$:
  - receive a value along channel c and assign it to variable x:

$$\frac{\ell_i \xrightarrow{c?x} \ell_i' \quad \wedge \quad len(\xi(c)) = k > 0 \quad \wedge \quad \xi(c) \;=\; v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle}$$

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

  - transmit value $v \in dom(c)$ over channel c:

$$\frac{\ell_i \xrightarrow{c!v} \ell_i' \quad \wedge \quad len(\xi(c)) = k < cap(c) \quad \wedge \quad \xi(c) \;=\; v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle}$$

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

Handling unexpected messages

sender S	timer	receiver R	channel c	channel d	event
$snd\_msg(0)$	*off*	$wait(0)$	$\varnothing$	$\varnothing$	
$st\_tmr(0)$	*off*	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	message with bit 0 transmitted
$wait(0)$	*on*	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	
$snd\_msg(0)$	*off*	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	timeout
$st\_tmr(0)$	*off*	$wait(0)$	$\langle m, 0\rangle \langle m, 0\rangle$	$\varnothing$	retransmission
$st\_tmr(0)$	*off*	$pr\_msg(0)$	$\langle m, 0\rangle$	$\varnothing$	receiver reads first message
$st\_tmr(0)$	*off*	$snd\_ack(0)$	$\langle m, 0\rangle$	$\varnothing$	
$st\_tmr(0)$	*off*	$wait(1)$	$\langle m, 0\rangle$	0	receiver changes into mode-1
$st\_tmr(0)$	*off*	$pr\_msg(1)$	$\varnothing$	0	receiver reads retransmission
$st\_tmr(0)$	*off*	$wait(1)$	$\varnothing$	0	and ignores it
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

nanoPromela

- Promela (Process Meta Language):
  modeling language for SPIN
    - most widely used model checker
    - developed by Gerard Holzmann (Bell Labs, NASA JPL)
    - ACM Software Award 2002
- nanoPromela is the core of Promela
    - shared variables and channel-based communication
    - formal semantics of a Promela model is a channel system
    - processes are defined by means of a guarded command language
- No actions, statements describe effect of actions

nanoPromela

nanoPromela-program $\overline{\mathcal{P}} = [\mathcal{P}_1 | \ldots | \mathcal{P}_n]$ with $\mathcal{P}_i$ processes
A process is specified by a statement:

$$\begin{array}{lcl}
\text{stmt} & ::= & \texttt{skip} \;\big|\; x := \text{expr} \;\big|\; c?x \;\big|\; c!\text{expr} \;\big| \\
 & & \text{stmt}_1 \,;\, \text{stmt}_2 \;\big|\; \texttt{atomic}\{\text{assignments}\} \;\big| \\
 & & \textbf{if} \quad :: g_1 \Rightarrow \text{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \text{stmt}_n \quad \textbf{fi} \quad \big| \\
 & & \textbf{do} \quad :: g_1 \Rightarrow \text{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \text{stmt}_n \quad \textbf{od} \\
\text{assignments} & ::= & x_1 := \text{expr}_1 \,;\, x_2 := \text{expr}_2 \,;\, \ldots \,;\, x_m := \text{expr}_m
\end{array}$$

x is a variable in *Var*, expr an expression and *c* a channel, g_i a guard

assume the Promela specification is type-consistent

Conditional statements

$$\textbf{if} \;\; :: g_1 \Rightarrow \text{stmt}_1 \; \ldots \; :: g_n \Rightarrow \text{stmt}_n \; \textbf{fi}$$

- Nondeterministic choice between statements stmt_i for which g_i holds
- Test-and-set semantics: (deviation from Promela)
  - guard evaluation + selection of enabled command + execution first atomic step
    of selected statement is all performed atomically
- The **if**--**fi**--command blocks if no guard holds
  - parallel processes may unblock a process by changing shared variables
  - e.g., when y=0, **if** $:: y > 0 \Rightarrow x := 42$ **fi** waits until y exceeds 0
- Standard abbreviations:
  - **if** g **then** stmt_1 **else** stmt_2 **fi** $\equiv$ **if** $:: g \Rightarrow \text{stmt}_1 \;\; :: \neg g \Rightarrow \text{stmt}_2$ **fi**
  - **if** g **then** stmt_1 **fi** $\equiv$ **if** $:: g \Rightarrow \text{stmt}_1 \;\; :: \neg g \Rightarrow$ `skip` **fi**

Iteration statements

$$\textbf{do} \;\; :: g_1 \Rightarrow \mathrm{stmt}_1 \; \ldots \; :: g_n \Rightarrow \mathrm{stmt}_n \; \textbf{od}$$

- Iterative execution of nondeterministic choice among $g_i \Rightarrow \mathrm{stmt}_i$
  - where guard g_i holds in the current state
- No blocking if all guards are violated; instead, loop is aborted
- $\textbf{do} \;\; :: g \Rightarrow \mathrm{stmt} \; \textbf{od} \equiv \textbf{while} \; g \; \textbf{do} \; \mathrm{stmt} \; \textbf{od}$
- No break-statements to abort a loop (deviation from Promela)

Peterson's algorithm

The nanoPromela-code of process $\mathcal{P}_1$ is given by the statement:

```
do  ::  true ⇒  skip;
                atomic{$b_1$ := true; $x$ := 2};
                if  ::  $(x = 1) \lor \neg b_2$  ⇒  $crit_1$ := true  fi
                atomic{$crit_1$ := false; $b_1$ := false}
od
```

Beverage vending machine

The following nanoPromela program describes its behaviour:

```
do  :: true ⇒
         skip;
         if    :: $nsprite > 0 \Rightarrow nsprite := nsprite - 1$
               :: $nbeer > 0 \Rightarrow nbeer := nbeer - 1$
               :: $nsprite = nbeer = 0 \Rightarrow$ skip
         fi
    :: true ⇒  atomic{$nbeer := max; nsprite := max$}
od
```