

System Modeling

- Introduction
- Rugby Meta-Model
- Finite State Machines
- Petri Nets
- Untimed Model of Computation

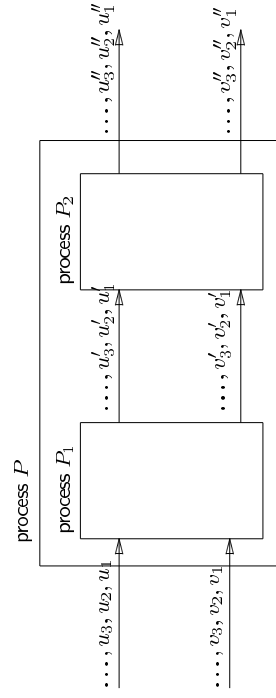
Synchronous Model of Computation

- Timed Model of Computation
- Integration of Computational Models
- Tightly Coupled Process Networks



Perfect Synchrony Hypothesis

Neither computation nor communication takes time.

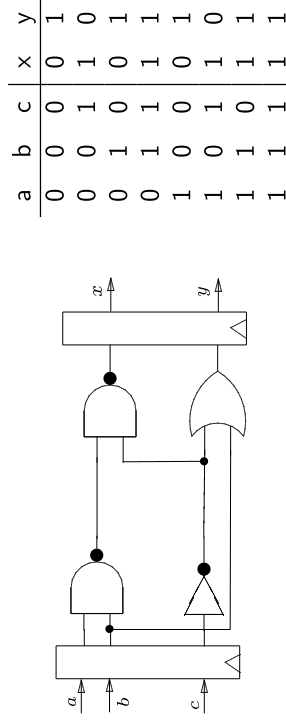


Suitable Applications for the Synchronous Model

- **Reactive systems** receive inputs, react to them by computing outputs and wait for the next inputs to arrive.
- **Embedded control systems** connected to sensors and actuators
- **Wireless communication devices** receiving samples with a fixed, predefined frequency
- **Telecom switches** reading all input data packets before re-emitting al packets
- **Synchronous hardware**



Seperation of Function and Time



Gate	Delay
Inverter	1.5 ns
NAND gate	1.8 ns
OR gate	2.1 ns

a	b	c	x	y
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1



Synchronous Processes

Synchronous processes are restricted untimed processes:

1. All process signatures are constant 1 vectors:
 $\langle \{1, \dots\}, \{1, \dots\} \rangle$
2. The value set is $V \cup \perp$ indicating the **absence of an event**.



Scan Processes

$$\begin{aligned} \text{scanS}(g, w_0) &= \text{scanU}(1, g, w_0) \\ \text{with } \forall w \in V, \exists w' \in V : g(w, \perp) &= w' \\ \text{scanS}(g, w_0) &= \text{scandU}(1, g, w_0) \\ \text{with } \forall w \in V, \exists w' \in V : g(w, \perp) &= w' \end{aligned}$$



Synchronous Processes

Synchronous processes are restricted untimed processes:

1. All process signatures are constant 1 vectors:
 $\langle \{1, \dots\}, \{1, \dots\} \rangle$
2. The value set is $V \cup \perp$ indicating the **absence of an event**.



Map Processes

$$\begin{aligned} \text{mapS}(f) &= \text{mapU}(1, f) \\ \text{with } \exists \bar{e}' \in \bar{E} : f(\perp) &= \bar{e}' \\ \forall \bar{e} \in \bar{E} : \#f(\bar{e}) &= 1 \\ \text{mapSstrict}(f) &= \text{mapU}(1, f') \\ \text{with } \forall \hat{e} \in \hat{E} : \#f(\hat{e}) &= 1 \\ f'(\hat{e}) &= \begin{cases} \perp & \text{if } \hat{e} = \perp \\ f(\hat{e}) & \text{otherwise} \end{cases} \end{aligned}$$



Moore and Mealy Processes

$$\begin{aligned} \text{mooreS}(g, f, w_0) &= \text{mooreU}(1, g, f, w_0) \\ \text{with } \forall w \in V, \exists w' \in V : g(w, \perp) &= w' \\ \forall w \in V, \exists \bar{e}' \in \bar{E} : f(w, \perp) &= \bar{e}' \\ \forall w \in V, \bar{e} \in \bar{E} : \#f(w, \bar{e}) &= 1 \\ \text{mealyS}(g, f, w_0) &= \text{mealyU}(1, g, f, w_0) \\ \text{with } \forall w \in V, \exists w' \in V : g(w, \perp) &= w' \\ \forall w \in V, \exists \bar{e}' \in \bar{E} : f(w, \perp) &= \bar{e}' \\ \forall w \in V, \bar{e} \in \bar{E} : \#f(w, \bar{e}) &= 1 \end{aligned}$$



Zip Processes

$$\begin{aligned} \text{zipS}() &= p \\ \text{with } p(\bar{s}_a, \bar{s}_b) &= \bar{s}_c \\ \langle \bar{c}_i \rangle &= \begin{cases} \perp & \text{if } \bar{a}_i = \perp \text{ and } \bar{b}_i = \perp \\ (\bar{a}_i, \bar{b}_i) & \text{otherwise} \end{cases} \\ \pi(\nu_a, \bar{s}_a) &= \langle \bar{a}_i \rangle, \nu_a(i) = 1 \\ \pi(\nu_b, \bar{s}_b) &= \langle \bar{b}_i \rangle, \nu_b(i) = 1 \\ \pi(\nu_c, \bar{s}_c) &= \langle \bar{c}_i \rangle, \nu'(i) = 1 \end{aligned}$$

$$\text{zipWithS}(f) = \text{zipWithU}(1, 1, f)$$



Sources and Sinks

$$\begin{aligned} \text{sources}(g, w_0) &= p \\ \text{where } p() &= \bar{s}' \\ w_i &= \bar{e}'_i \\ g(w_i) &= w_{i+1} \\ \pi(\nu', \bar{s}') &= \langle \langle \bar{e}'_i \rangle \rangle, \nu'(i) = 1 \\ \\ \text{sinks}(g, w_0) &= p \\ \text{where } p(\bar{s}) &= \langle \rangle \\ g(w_i) &= w_{i+1} \\ \pi(\nu, \bar{s}) &= \langle \bar{a}_i \rangle, \nu(i) = 1 \\ \\ \text{initS}(\bar{r}) &= p \\ \text{where } p(\bar{s}) &= \bar{r} \oplus \bar{s} \\ \nu &= \nu' = 1 \\ \bar{r}, \bar{s} &\in \bar{S} \end{aligned}$$



Unzip Processes

$$\begin{aligned} \text{unzipS}() &= p \\ \text{where } p(\bar{s}) &= \langle \bar{s}', \bar{s}'' \rangle \\ \bar{a}_i &= \begin{cases} \perp & \text{if } \bar{c}_i = \perp \text{ or } \bar{c}_i = (\perp, v_b) \\ v_a & \text{otherwise, where } \bar{c}_i = (v_a, v_b) \end{cases} \\ \bar{b}_i &= \begin{cases} \perp & \text{if } \bar{c}_i = \perp \text{ or } \bar{c}_i = (v_a, \perp) \\ v_b & \text{otherwise, where } \bar{c}_i = (v_a, v_b) \end{cases} \\ \pi(\nu, \bar{s}) &= \langle \bar{c}_i \rangle, \nu(i) = 1 \\ \pi(\nu', \bar{s}') &= \langle \bar{a}_i \rangle, \nu'(i) = 1 \\ \pi(\nu'', \bar{s}'') &= \langle \bar{b}_i \rangle, \nu''(i) = 1 \end{aligned}$$

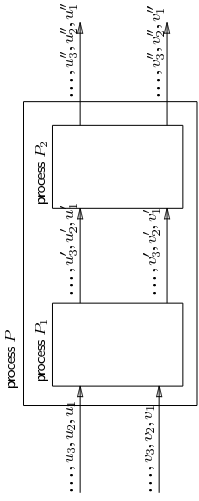


Process Merge

$$\begin{aligned} \text{mapS}(f_1) \circ \text{mapS}(f_2) &= \text{mapS}(f_1 \circ f_2) \\ \text{mealyS}(g_1, f_1, v_0) \circ \text{mealyS}(g_2, f_2, w_0) &= \text{mealyS}(g, f, (v_0, w_0)) \\ \text{where } g((v, w), \bar{e}) &= (g_1(v, f_2(w, \bar{e})), g_2(w, \bar{e})) \\ f((v, w), \bar{e}) &= f_1(v, f_2(w, \bar{e})) \end{aligned}$$



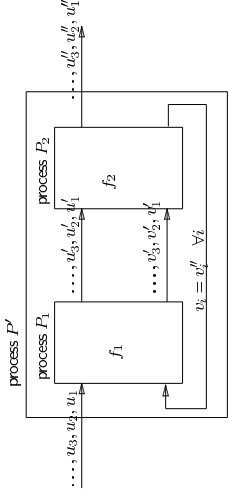
Process Merge Example



$$\begin{aligned}
 P_1 &= \text{maps}(f_1) \\
 P_2 &= \text{maps}(f_2) \\
 f_1((x, y)) &= (x + y, x - y) \\
 f_2((x, y)) &= (x - y, x + y) \\
 P &= \text{maps}(f_P) \\
 f_P((x, y)) &= f_2(f_1((x, y))) = f_2((x + y, x - y)) \\
 &= (x + y - (x - y), x + y + x - y) = (2y, 2x)
 \end{aligned}$$



Feed-back Loops - cont'd

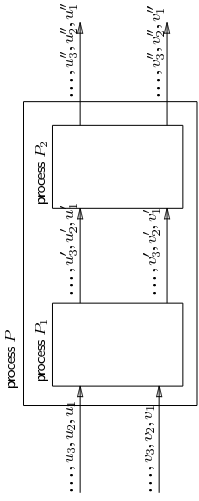


$$\begin{aligned}
 P_1 &= \text{maps}(f_1) \\
 P_2 &= \text{maps}(f_2) \\
 f_1((x, y)) &= (x + y + 1, x) \\
 f_2((x, y)) &= (x + y - 1, x)
 \end{aligned}$$

$$\begin{aligned}
 P' &= \text{maps}(f_{P'}) \\
 f_{P'}((x, y)) &= f_2(f_1((x, y))) = f_2((x + y + 1, x)) \\
 &= (x + y + 1 + x - 1, x + y + 1 - x) = (2x + y, y + 1) \\
 y &= y + 1
 \end{aligned}$$



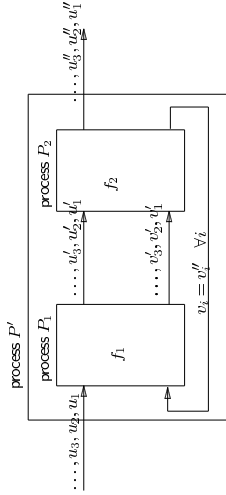
Process Merge Example



$$\begin{aligned}
 P_1 &= \text{maps}(f_1) \\
 P_2 &= \text{maps}(f_2) \\
 f_1((x, y)) &= (x + y, x - y) \\
 f_2((x, y)) &= (x - y, x + y) \\
 P &= \text{maps}(f_P) \\
 f_P((x, y)) &= f_2(f_1((x, y))) = f_2((x + y, x - y)) \\
 &= (x + y - (x - y), x + y + x - y) = (2y, 2x)
 \end{aligned}$$



Feed-back Loops

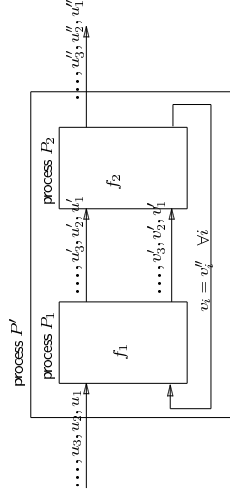


$$\begin{aligned}
 P_1 &= \text{maps}(f_1) \\
 P_2 &= \text{maps}(f_2) \\
 f_1((x, y)) &= (x + y, x - y) \\
 f_2((x, y)) &= (x - y, x + y)
 \end{aligned}$$

$$\begin{aligned}
 P' &= \text{maps}(f_{P'}) \\
 f_{P'}(x) &= z \text{ where} \\
 (z, y) &= f_2(f_1((x, y))) = f_2((x + y, x - y)) \\
 &= (x + y - x + y, x + y + x - y) = (2y, 2x) \\
 y &= 2x \\
 z &= 2y = 4x
 \end{aligned}$$



Feed-back Loops - cont'd



$$\begin{aligned}
 P_1 &= \text{maps}(f_1) \\
 P_2 &= \text{maps}(f_2) \\
 f_1((x, y)) &= (x + y, y) \\
 f_2((x, y)) &= (x + y, y)
 \end{aligned}$$

$$\begin{aligned}
 P' &= \text{maps}(f_{P'}) \\
 f_{P'}((x, y)) &= f_2(f_1((x, y))) = f_2((x + y, y)) \\
 &= (x + y + y, y) = (x + 2y, y) \\
 y &= y \\
 P'((1, 2, 3)) &= (1, 2, 3) \text{ for } y = 0 \\
 P'((1, 2, 3)) &= (3, 4, 5) \text{ for } y = 1 \\
 P'((1, 2, 3)) &= (-1, 0, 1) \text{ for } y = -1 \\
 P'((1, 2, 3)) &= (1, 4, 1) \text{ for } y = 0, 1, -1
 \end{aligned}$$



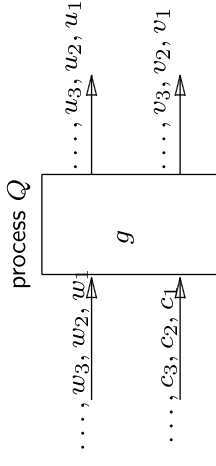
Feed-back Loops - cont'd

Only models with a exactly **one unambiguous solution** are considered valid.



A. Jantsch, KTH

Feed-back Loops - cont'd



$g(w, c) = (u, v)$ where
 $u =$ if c then v else w ;
 $v =$ if c then w else u ;

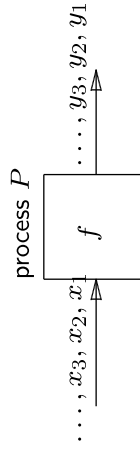
has the unambiguous solution

$$u = v = w$$



A. Jantsch, KTH

Feed-back Loops - cont'd



The implicit equation

$$f(x) = y = y^2 - 4x^4 - 2x^2$$

has the solution

$$f(x) = 2x^2 + 1$$



A. Jantsch, KTH

The Synchronous Model of Computation

Definition: The **Synchronous Model of Computation (Synchronous MoC)** is defined as Synchronous MoC= (C, O) , where

$$C = \{mapS, mapStrict, scans, scandS, mooreS, mealyS, zipS, unzipS, zipWithS, sourceS, sinkS, initS\}$$

$$O = \{||, o, FBs\}$$

In other words, a process or a process network belongs to the **Synchronous MoC Domain** iff all its processes and process compositions are constructed either by one of the named process constructors or by one of the composition operators. We call such processes **S-MoC processes**.



A. Jantsch, KTH

Clocked Synchrony Hypothesis

There is a global clock signal controlling the start of each computation in the system. Communication takes no time and computation takes one clock cycle.



The Clocked Synchronous Model of Computation

Definition: The **Clocked Synchronous Model of Computation** is defined as Clocked Synchronous MoC= (C, O) , where

$$C = \{\Delta, \text{mapCS}, \text{scanCS}, \text{mooreCS}, \text{mealyCS}, \text{zipCS}, \text{unzipCS}, \text{sourceCS}, \text{sinkCS}, \text{initCS}\}$$

$$O = \{\|, \circ, \mathbf{FBP}\}$$

In other words, a process or a process network belongs to the **Clocked Synchronous MoC Domain** iff all its processes and process compositions are constructed either by one of the named process constructors or by one of the composition operators. We call such processes **CS-MoC processes**.

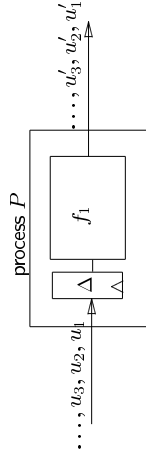


Clocked Synchrony Hypothesis

There is a global clock signal controlling the start of each computation in the system. Communication takes no time and computation takes one clock cycle.



Clocked Synchronous Processes



$$\Delta = \text{scandS}(g, \perp) \text{ where } g(w, \bar{e}) = \bar{e}$$

$$\begin{aligned} \text{mapCS}(f) &= \text{maps}(f) \circ \Delta \\ \text{scanCS}(g, w_0) &= \text{scans}(g, w_0) \circ \Delta \\ \text{mealyCS}(g, f, w_0) &= \text{mealyS}(g, f, w_0) \circ \Delta \\ \text{zipCS}(\bar{s}_1, \bar{s}_2) &= \text{zips}(\Delta(\bar{s}_1), \Delta(\bar{s}_2)) \\ \text{unzipCS}() &= \text{unzips}() \circ \Delta \\ \text{sourceCS} &= \text{sourceS} \\ \text{sinkCS} &= \text{sinkS} \\ \text{initCS} &= \text{initS} \end{aligned}$$



Extended Characteristic Function - 1

$$p = \text{mapS}(f)$$

$$\bar{s}' = p(\bar{s})$$

$$\bar{s} = \langle \bar{e}_i \rangle$$

$$\bar{s}' = \langle \bar{e}'_i \rangle$$

$$\bar{e}'_i = \bar{e}_{i-1} \quad \forall i \in \mathbb{N}$$

$$\bar{e}'_0 = \perp$$

$$\bar{e}'_i = f(\bar{e}_i) \quad \forall i \in \mathbb{N}_0$$

Perfectly Synchronous

Clocked Synchronous



Extended Characteristic Function - 2

Definition: Let p a CS-MoC process.

$$f_{p,\bar{s}}(\bar{e}_i) = \bar{e}'_j,$$

with $p(\bar{s}) = \bar{s}'$,

$$\bar{s}, \bar{s}' \in \bar{S},$$

$$\langle \bar{e}_i \rangle = \bar{s},$$

$$\langle \bar{e}'_j \rangle = \bar{s}'$$

is an **extended characteristic function** of process p if it is defined in terms of an arbitrary functional expression of $\bar{e}_i, i \leq j$.

Example:

$$f_{\Delta,s}(\bar{e}_i) = \bar{e}_{i-1}, i > 0$$

$$f_{\Delta,s}(\bar{e}_0) = \perp$$



Clocked Synchronous Example - cont'd

$$f_1((\perp, \perp)) = (0, 0)$$

$$f_1((x, \perp)) = (x, x)$$

$$f_1((\perp, y)) = (y, -y)$$

$$f_1((x, y)) = (x + y, x - y)$$

$$f_2((\perp, \perp)) = (0, 0)$$

$$f_2((x, \perp)) = (x, x)$$

$$f_2((\perp, y)) = (-y, y)$$

$$f_2((x, y)) = (x - y, x + y)$$

$$f_P((x_i, y_i)) = f_2(f_\Delta(f_1(f_\Delta((x_i, y_i))))))$$

$$= f_2(f_\Delta(f_1((x_{i-1}, y_{i-1}))))$$

$$= f_2(f_\Delta((x_{i-1} + y_{i-1}, x_{i-1} - y_{i-1})))$$

$$= f_2((x_{i-2} + y_{i-2}, x_{i-2} - y_{i-2}))$$

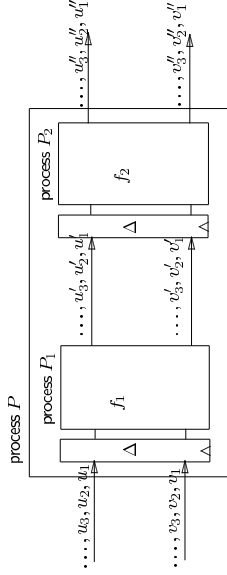
$$= (x_{i-2} + y_{i-2} - x_{i-2} + y_{i-2} + y_{i-2} + x_{i-2} - y_{i-2})$$

$$= (2y_{i-2}, 2x_{i-2}) \text{ for } i > 1$$

$$= 0 \text{ for } 0 \leq i \leq 1$$



Clocked Synchronous Example



$$\text{map2CS}(f) = \text{mapS}(f) \circ \text{zipS}() \circ \Delta$$

$$P_1 = \text{map2CS}(f_1)$$

$$P_2 = \text{mapCS}(f_2)$$

$$f_1((\perp, \perp)) = (0, 0)$$

$$f_1((x, \perp)) = (x, x)$$

$$f_1((\perp, y)) = (y, -y)$$

$$f_1((x, y)) = (x + y, x - y)$$

$$f_2((\perp, \perp)) = (0, 0)$$

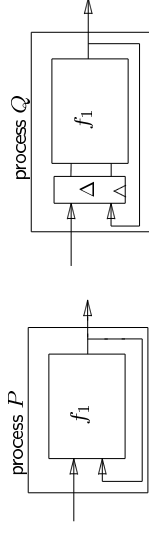
$$f_2((x, \perp)) = (x, x)$$

$$f_2((\perp, y)) = (-y, y)$$

$$f_2((x, y)) = (x - y, x + y)$$



Clocked Synchronous with Feed-back



$$f_1(x, y) = 2y - 2x$$

$$P = \text{mapS}(f_1) \circ \text{zip}()$$

$$f_P(x) = z \text{ where}$$

$$z = f_1(x, z)$$

$$= 2z - 2x$$

$$-z = -2x$$

$$z = 2x$$

$$Q = \text{map2CS}(f_1)$$

$$f_Q(x_i) = z_i \text{ where}$$

$$z_i = f_1(f_\Delta(x_i, z_i))$$

$$= f_1(x_{i-1}, z_{i-1})$$

$$= 2z_{i-1} - 2x_{i-1} \text{ for } i > 0$$

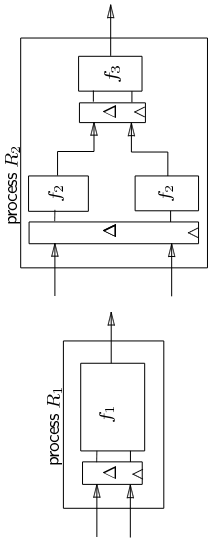
$$z_0 = f_1(0, 0) = 0$$

$$P([0, 1, 2, 3]) = [0, 2, 4, 6]$$

$$Q([0, 1, 2, 3]) = [0, 0, -2, -8, -22]$$



Process Substitution Example



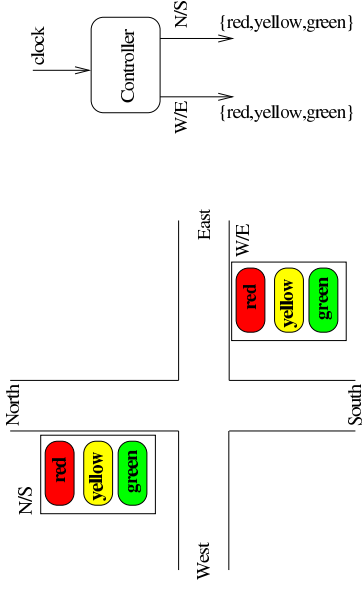
$$f_1(x, y) = 2y - 2x$$

$$f_2(x) = 2x$$

$$f_3(x, y) = y - x$$



The Traffic Light Controller



$$mooreS(nsf, outif, (rr1, 0)) = tlctrl$$

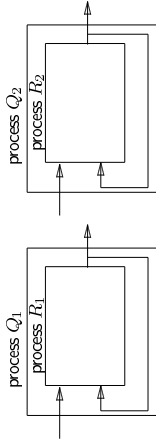
where

$$tlctrl(s_c) = s_l$$

$$s_l = \langle \langle n_i, e_i \rangle \rangle$$



Process Substitution Example - cont'd



$$f_{Q_1}(x_i) = z_i \text{ where}$$

$$z_i = f_{R_1}(x_i, z_i)$$

$$= f_1(f_\Delta(x_i, z_i))$$

$$= f_1(x_{i-1}, z_{i-1})$$

$$= 2z_{i-1} - 2x_{i-1} \text{ for } i \geq 1$$

$$z_0 = 0$$

$$f_{Q_2}(x_0) = 0$$

$$f_{Q_2}(x_1) = 0$$

$$f_{Q_2}(x_i) = z_i \text{ where}$$

$$z_i = f_3(f_\Delta(f_2(f_\Delta(x_i)), f_2(f_\Delta(z_i))))$$

$$= f_3(f_\Delta(f_2(x_{i-1}), f_2(z_{i-1})))$$

$$= f_3(f_\Delta(2x_{i-1}, 2z_{i-1}))$$

$$= f_3(2x_{i-2}, 2z_{i-2})$$

$$= 2z_{i-2} - 2x_{i-2}$$

$$z_i = 2z_{i-2} - 2x_{i-2} \text{ for } i \geq 2$$

$$Q_1([0, 1, 2, 3]) = [0, 0, -2, -8, -22]$$

$$Q_2([0, 1, 2, 3]) = [0, 0, 0, -2, -4, -10]$$



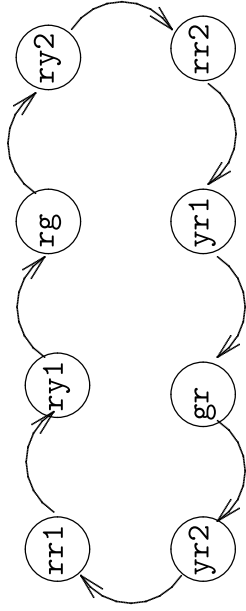
The Traffic Light Controller - cont'd

$$nsf :: Clock \rightarrow State$$

$nsf \ c$	$(rr1, cnt)$	$cnt < 3$	$\rightarrow State = (rr1, cnt + 1)$
		otherwise	$= (ry1, 0)$
$nsf \ c$	$(ry1, cnt)$	$cnt < 3$	$\rightarrow State = (ry1, cnt + 1)$
		otherwise	$= (rg, 0)$
$nsf \ c$	(rg, cnt)	$cnt < 60$	$\rightarrow State = (rg, cnt + 1)$
		otherwise	$= (ry2, 0)$
$nsf \ c$	$(ry2, cnt)$	$cnt < 3$	$\rightarrow State = (ry2, cnt + 1)$
		otherwise	$= (rr2, 0)$
$nsf \ c$	$(rr2, cnt)$	$cnt < 3$	$\rightarrow State = (rr2, cnt + 1)$
		otherwise	$= (yr1, 0)$
$nsf \ c$	$(yr1, cnt)$	$cnt < 3$	$\rightarrow State = (yr1, cnt + 1)$
		otherwise	$= (gr, 0)$
$nsf \ c$	(gr, cnt)	$cnt < 60$	$\rightarrow State = (gr, cnt + 1)$
		otherwise	$= (yr2, 0)$
$nsf \ c$	$(yr2, cnt)$	$cnt < 3$	$\rightarrow State = (yr2, cnt + 1)$
		otherwise	$= (rr1, 0)$



Traffic Light Controller - cont'd



Traffic Light Controller - cont'd

```

outf2 :: State
outf2 (rr1, cnt) | cnt == 0 = (red, ⊥)
                 | otherwise = (⊥, ⊥)
outf2 (rr2, cnt) | cnt == 0 = (⊥, red)
                 | otherwise = (⊥, ⊥)
outf2 (ry1, cnt) | cnt == 0 = (⊥, yellow)
                 | otherwise = (⊥, ⊥)
outf2 (ry2, cnt) | cnt == 0 = (⊥, yellow)
                 | otherwise = (⊥, ⊥)
outf2 (rg, cnt)  | cnt == 0 = (⊥, green)
                 | otherwise = (⊥, ⊥)
outf2 (yr1, cnt) | cnt == 0 = (yellow, ⊥)
                 | otherwise = (⊥, ⊥)
outf2 (yr2, cnt) | cnt == 0 = (yellow, ⊥)
                 | otherwise = (⊥, ⊥)
outf2 (gr, cnt)  | cnt == 0 = (green, ⊥)
                 | otherwise = (⊥, ⊥)
    
```



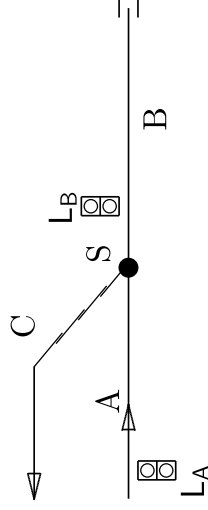
Traffic Light Controller - cont'd

```

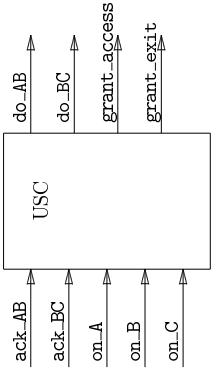
outf :: State
outf (rr1, cnt) = (red, red)
outf (rr2, cnt) = (red, red)
outf (ry1, cnt) = (red, yellow)
outf (ry2, cnt) = (red, yellow)
outf (rg, cnt)  = (red, green)
outf (yr1, cnt) = (yellow, red)
outf (yr2, cnt) = (yellow, red)
outf (gr, cnt)  = (green, red)
    
```



Validation Example: U-turn Section Controller



U-turn Section Controller - cont'd



- `ackAB` and `ackBC` indicate the status of the switch.
- `onA`, `onB`, and `onC` are signals from sensors from the three sections. They are True if a train is in the corresponding section.
- `doAB` and `doBC` are requests to the switch to connect the corresponding sections.
- `grantAccess` and `grantExit` are control signals for traffic lights. `grantAccess` controls traffic light L_A ; `grantExit` controls traffic light L_B .

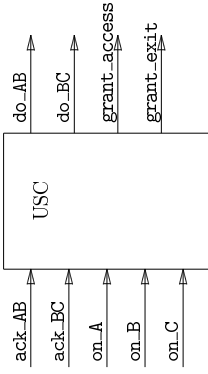


Safety Properties for the USC

- No collision:** A train is never granted access when another train is in one of the sections A, B or C.
- No conflict:** No conflicting control events are ever sent to the switch, i.e. say that it should connect both A to B and B to C.
- No derail:** The switch shall never be requested to change its state when a train is driving on the switch region.



U-turn Section Controller - cont'd



$USC = \text{mapS}(f)$
 where $f(\text{onA}, \text{onB}, \text{onC}, \text{ackAB}, \text{ackBC}) = (\text{grantAccess}, \text{grantExit}, \text{doAB}, \text{doBC})$
 $\text{grantAccess} = \text{emptySection} \wedge \text{ackAB}$
 $\text{grantExit} = \text{onlyOnB} \wedge \text{ackBC}$
 $\text{doAB} = \neg \text{ackAB} \wedge \text{emptySection}$
 $\text{doBC} = \neg \text{ackBC} \wedge \text{onlyOnB}$
 $\text{emptySection} = \neg(\text{onA} \vee \text{onB} \vee \text{onC})$
 $\text{onlyOnB} = \text{onB} \wedge \neg(\text{onA} \vee \text{onC})$



Monitor Processes

Definition: A **monitor process** for a logic property emits True when the property holds, and False otherwise.

$\text{mon1S}(f_1) = \text{mapS}$
 where $f_1 : \text{Boolean} \rightarrow \text{Boolean}$
 $\text{mon2S}(f_2) = \text{zipWithS}$
 where $f_2 : (\text{Boolean}, \text{Boolean}) \rightarrow \text{Boolean}$
 $\text{mon1sS}(g_3, f_3, w_0) = \text{mealyS}(g_3, f_3, w_0)$
 where $g_3, f_3 : (V, \text{Boolean}) \rightarrow \text{Boolean}$
 $\text{mon2sS}(g_4, f_4, w_0) = \text{mealyS}(g_3, f_3, w_0) \circ \text{zipS}$
 where $g_4, f_4 : (V, \text{Boolean}, \text{Boolean}) \rightarrow \text{Boolean}$



Monitor Processes - cont'd

or is true if either one of the inputs is true:

$$\text{or} = \text{mon2S}(f) \\ \text{where } f(b_1, b_2) = b_1 \vee b_2$$

implies is true if b_1 implies b_2 :

$$\text{implies} = \text{mon2S}(f) \\ \text{where } f(b_1, b_2) = \neg b_1 \vee b_2$$



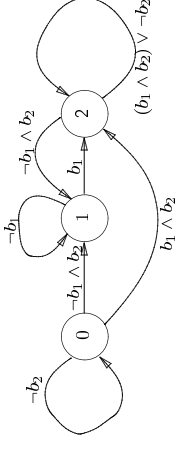
Monitor Processes - cont'd

onceSince: if b_1 has been true at least once since the last time b_2 has been true:

$$\text{onceSince} = \text{mon2sS}(g, f, 0)$$

$$\text{where } g(w, b_1, b_2) = \begin{cases} 0 & \text{if } w = 0 \wedge \neg b_2 \\ 1 & \text{if } (w = 0 \wedge \neg b_1 \wedge b_2) \vee (w = 1 \wedge \neg b_1) \\ & \vee (w = 2 \wedge \neg b_1 \wedge b_2) \\ 2 & \text{if } (w = 0 \wedge b_1 \wedge b_2) \vee (w = 1 \wedge b_1) \\ & \vee (w = 2 \wedge ((b_1 \wedge b_2) \vee \neg b_2)) \end{cases}$$

$$f(w, b_1, b_2) = (w = 2)$$



Monitor Processes - cont'd

or is true if either one of the inputs is true:

$$\text{or} = \text{mon2S}(f) \\ \text{where } f(b_1, b_2) = b_1 \vee b_2$$

implies is true if b_1 implies b_2 :

$$\text{implies} = \text{mon2S}(f) \\ \text{where } f(b_1, b_2) = \neg b_1 \vee b_2$$



Monitor Processes - cont'd

after is true after the input b has been true once:

$$\text{after} = \text{mon1sS}(g, f, \text{False}) \\ \text{where } g(w, b) = w \vee b \\ f(w, b) = w \vee b$$

alwaysSince is true if b_2 has been continuously true after b_1 has been true once:

$$\text{alwaysSince} = \text{mon2sS}(g, f, \text{False}) \\ \text{where } g(w, b_1, b_2) = (w \wedge b_1) \vee (b_1 \wedge b_2) \\ f(w, b_1, b_2) = (w \wedge b_1) \vee (b_1 \wedge b_2)$$



Monitors for the USC

$$M_{eS}(s_1, s_2, s_3) = \text{mon2S}(f_1)(\text{zipS})(s_1, s_2), s_3) \\ \text{where } f_1((b_1, b_2), b_3) = \neg(b_1 \vee b_2 \vee b_3)$$

is True when all sections are empty;

$$M_{oB}(s_1, s_2, s_3) = \text{mon2S}(f_2)(\text{zipS})(s_1, s_2), s_3) \\ \text{where } f_2((b_1, b_2), b_3) = b_2 \wedge \neg(b_1 \vee b_3)$$

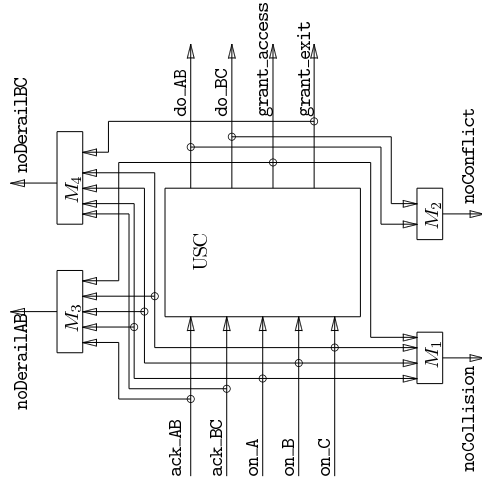
is True when there is a train only on section B;

$$M_d(s_1, s_2, s_3) = \text{implies}(\text{after}(s_2), \text{or}(\text{alwaysSince}(s_1, s_2), \text{onceSince}(s_3, s_2)))$$

is True when the derail condition is False.



Monitors for Safety Properties of the USC



A. Jantsch, KTH

Monitors for Safety Properties of the USC

No collision: noCollision, generated by M_1 , is always True provided that access to a train is only granted when the U-turn area is empty.

$$M_1(\text{grantAccess}, \text{onA}, \text{onB}, \text{onC}) = \text{implies}(\text{grantAccess}, M_{eS}(\text{onA}, \text{onB}, \text{onC})).$$

No conflict: noConflict tells if conflicting requests are sent to the switch.

$$M_2(\text{doAB}, \text{doBC}) = \text{non2S}(f)(\text{doAB}, \text{doBC}) \text{ where } f(b_1, b_2) = \neg(b_1 \wedge b_2)$$

No derail AB: M_3 (noDerailAB) checks the derail condition when the switch connects A to B.

$$M_3(\text{ackAB}, \text{onA}, \text{onB}, \text{onC}, \text{grantAccess}) = M_d(\text{ackAB}, \text{grantAccess}, M_{oB}(\text{onA}, \text{onB}, \text{onC}))$$

No derail BC: M_4 (noDerailBC) checks the derail condition when the switch connects B and C.

$$M_4(\text{ackBC}, \text{onA}, \text{onB}, \text{onC}, \text{grantExit}) = M_d(\text{ackBC}, \text{grantExit}, M_{oB}(\text{onA}, \text{onB}, \text{onC}))$$



A. Jantsch, KTH

Validation with Monitors

- Simulation
 - ★ Monitors can validate properties for given scenarios.
 - ★ Monitors can validate properties for random input stimuli.
- Formal verification
 - ★ Property checkers can prove that properties always hold.
 - ★ Formal verification requires to make all assumptions explicit, e.g.
 - * The switch remains in a given position until the controller requests a change.
 - * Initially there is no train in any of the sections A, B or C.
 - * Trains obey traffic lights.
 - * When a train leaves A it is on B; when a train leaves B it is either on A or on C.



A. Jantsch, KTH

Summary

- Perfect synchronous MoC
- Clocked synchronous MoC
- Feed-back loops
- Process merging
- Intermediate time abstraction allows to separate time and functionality
- Monitor/assertion based validation



A. Jantsch, KTH