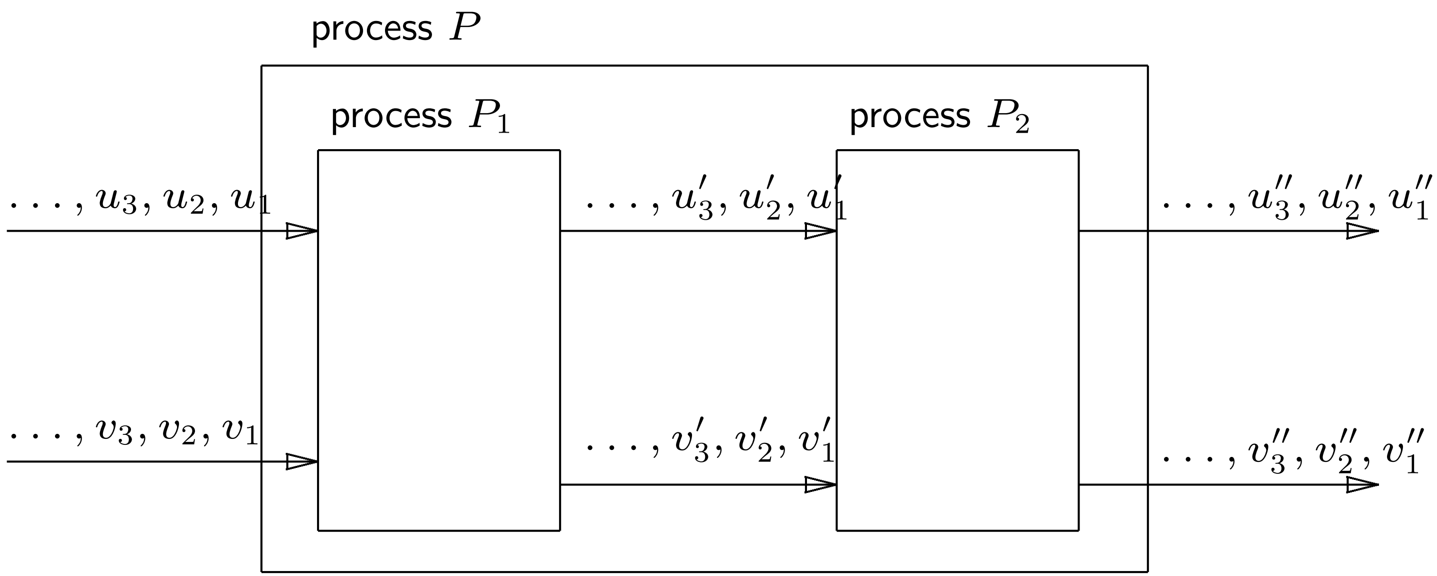
System Modeling

Perfect Synchrony Hypothesis

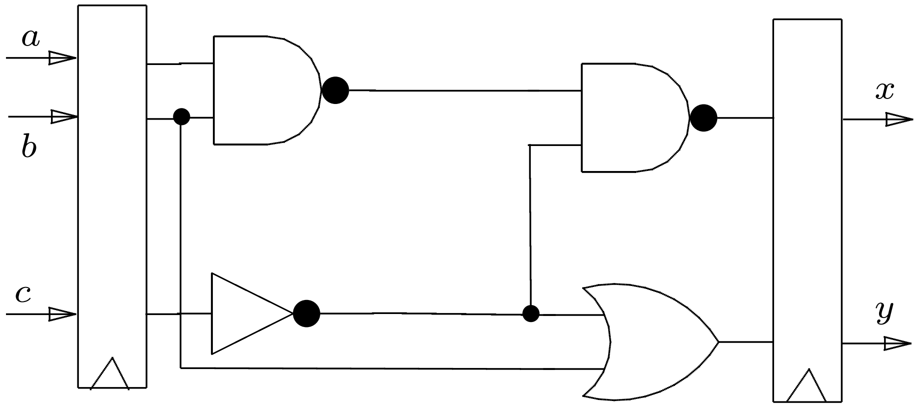
Neither computation nor communication takes time.



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



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

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

Synchronous Processes

Synchronous processes are restricted untimed processes:

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

Map Processes

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

Scan Processes

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

Moore and Mealy Processes

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

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

Zip Processes

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

Unzip Processes

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

Sources and Sinks

$$
\begin{array}{rcl}
\mathtt{sourceS}(g, w_0) & = & p \\
\text{where} & & p() = \bar{s}' \\
& & w_i = \bar{e}'_i \\
& & g(w_i) = w_{i+1} \\
& & \pi(\nu', \bar{s}') = \langle\langle \bar{e}'_i \rangle\rangle,\ \nu'(i) == 1
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{sinkS}(g, w_0) & = & p \\
\text{where} & & p(\bar{s}) = \langle\rangle \\
& & g(w_i) = w_{i+1} \\
& & \pi(\nu, \bar{s}) = \langle \bar{a}_i \rangle,\ \nu(i) = 1
\end{array}
$$

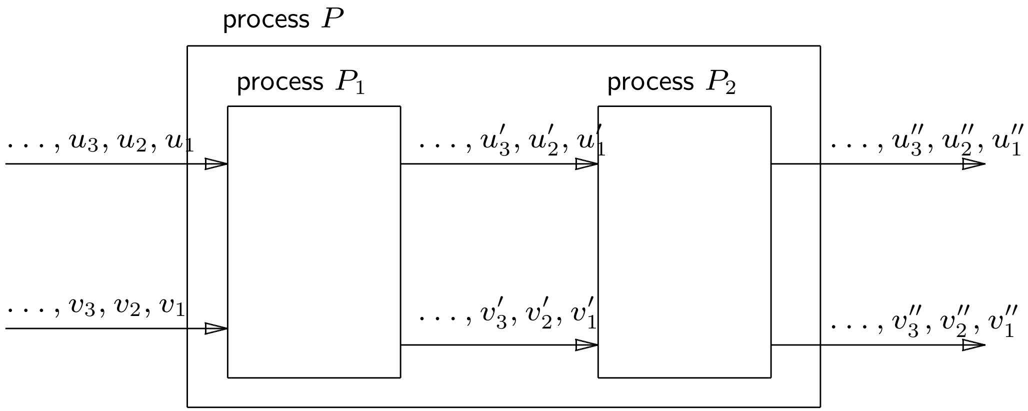
$$
\begin{array}{rcl}
\mathtt{initS}(\bar{r}) & = & p \\
\text{where} & & p(\bar{s}) = \bar{r} \oplus \bar{s} \\
& & \nu = \nu' = 1 \\
& & \bar{r}, \bar{s} \in \bar{S}
\end{array}
$$

Process Merge

$$\texttt{mapS}(f_1) \circ \texttt{mapS}(f_2) = \texttt{mapS}(f_1 \circ f_2)$$

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

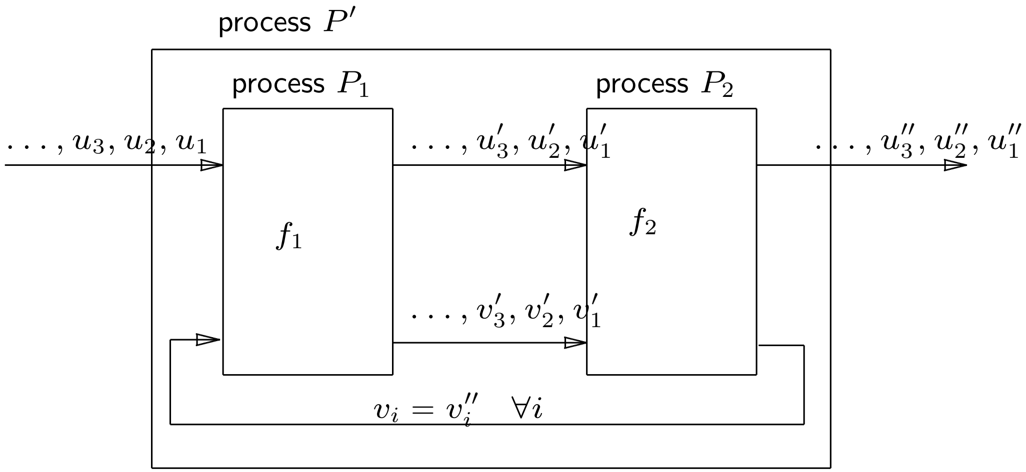
Process Merge Example



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

$$\begin{aligned}
P &= \texttt{mapS}(f_P)\\
f_P((x,y) &= f_2(f_1((x,y))) = f_2((x+y, x-y))\\
&= (x+y-(x-y), x+y+x-y) = (2y, 2x)
\end{aligned}$$

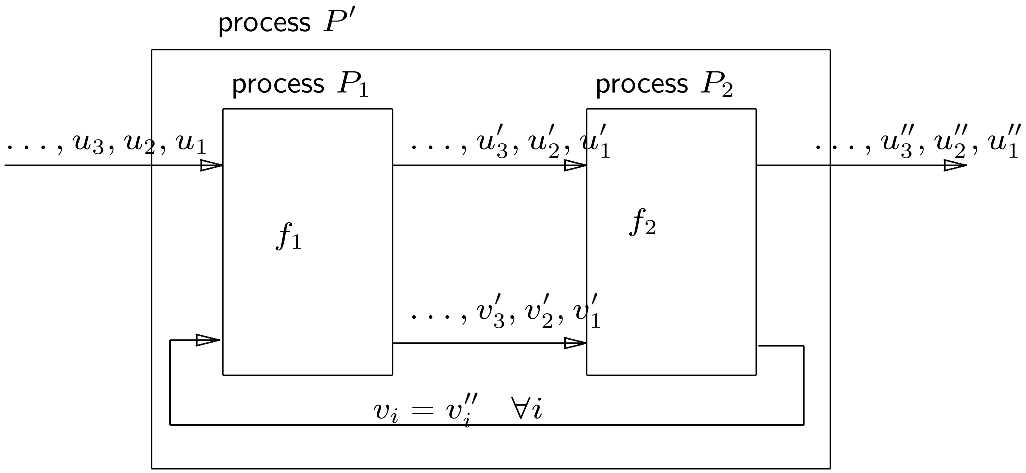
Feed-back Loops



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

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

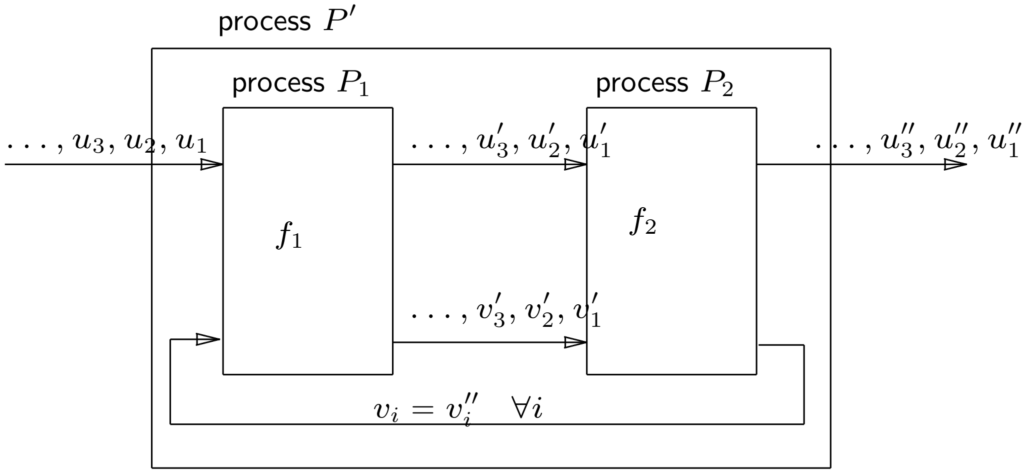
Feed-back Loops - cont'd



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

$$
\begin{aligned}
P' &= \texttt{mapS}(f_{P'}) \\
f_{P'}((x,y) &= f'_2(f'_1((x,y))) = f'_2((x+y+1,x)) \\
&= (x+y+1+x-1, x+y+1-x) = (2x+y, y+1) \\
y &= y+1
\end{aligned}
$$

Feed-back Loops - cont'd



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

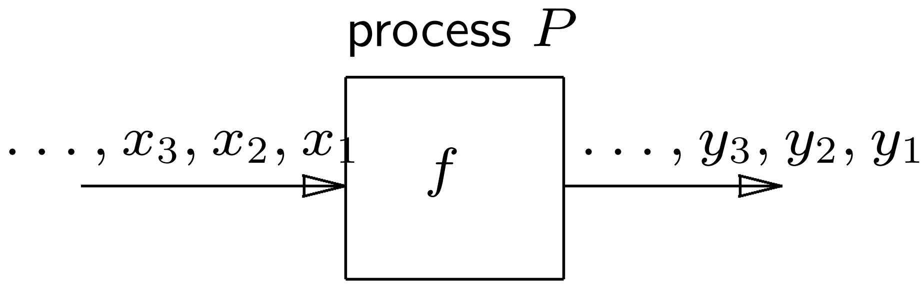
$$
\begin{aligned}
P' &= \texttt{mapS}(f_{P'}) \\
f_{P'}((x,y) &= f_2''(f_1''((x,y))) = f_2'((x+y,y)) \\
&= (x+y+y),y) = (x+2y,y) \\
y &= y
\end{aligned}
$$

$$
\begin{aligned}
P'(\langle 1,2,3\rangle) &= \langle 1,2,3\rangle && \text{for} \quad y = 0 \\
P'(\langle 1,2,3\rangle) &= \langle 3,4,5\rangle && \text{for} \quad y = 1 \\
P'(\langle 1,2,3\rangle) &= \langle -1,0,1\rangle && \text{for} \quad y = -1 \\
P'(\langle 1,2,3\rangle) &= \langle 1,4,1\rangle && \text{for} \quad y = 0,1,-1
\end{aligned}
$$

Feed-back Loops - cont'd

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

Feed-back Loops - cont'd



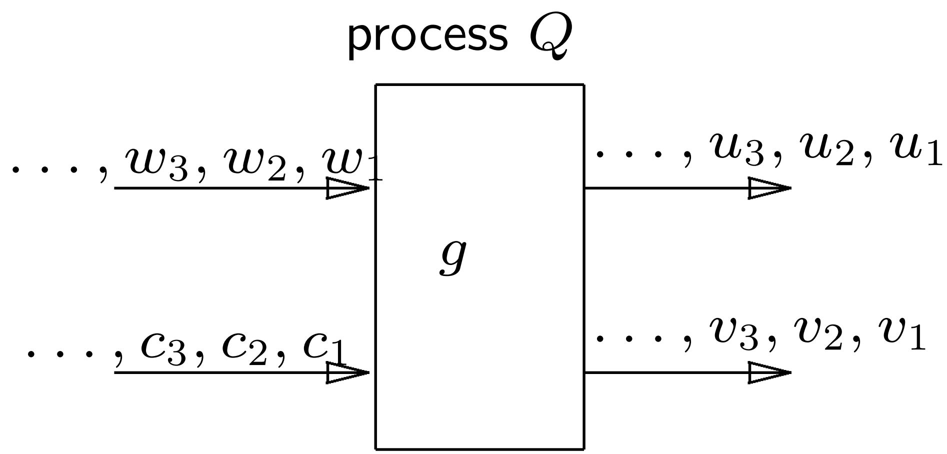
The implicit equation

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

has the solution

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

Feed-back Loops - cont'd



$$\begin{aligned} g(w,c) = (u,v) & \text{ where} \\ u = & \text{ if } c \text{ then } v \text{ else } w; \\ v = & \text{ if } c \text{ then } w \text{ else } u; \end{aligned}$$

has the unambiguous solution

$$u = v = w$$

The Synchronous Model of Computation

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

$$C = \{\mathtt{mapS}, \mathtt{mapSstrict}, \mathtt{scanS}, \mathtt{scandS}, \mathtt{mooreS}, \mathtt{mealyS}, \mathtt{zipS}, \mathtt{unzipS}, \mathtt{zipWithS}, \mathtt{sourceS}, \mathtt{sinkS}, \mathtt{initS}\}$$

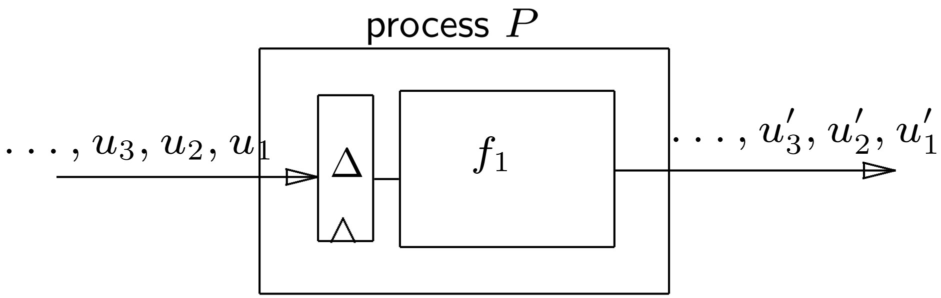
$$O = \{\, \|, \circ, \mathbf{FB_S}\}$$

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.

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



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

$$\begin{aligned}
\mathtt{mapCS}(f) &= \mathtt{mapS}(f) \circ \Delta \\
\mathtt{scanCS}(g, w_0) &= \mathtt{scanS}(g, w_0) \circ \Delta \\
\mathtt{mealyCS}(g, f, w_0) &= \mathtt{mealyS}(g, f, w_0) \circ \Delta \\
\mathtt{zipCS}()(\bar{s}_1, \bar{s}_2) &= \mathtt{zipS}()(\Delta(\bar{s}_1), \Delta(\bar{s}_2)) \\
\mathtt{unzipCS}() &= \mathtt{unzipS}() \circ \Delta \\
\mathtt{sourceCS} &= \mathtt{sourceS} \\
\mathtt{sinkCS} &= \mathtt{sinkS} \\
\mathtt{initCS} &= \mathtt{initS}
\end{aligned}$$

The Clocked Synchronous Model of Computation

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

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

$$O = \{ \parallel, \circ, \mathbf{FB_P}\}$$

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.

Extended Characteristic Function - 1

$$p = \texttt{mapS}(f)$$
$$\bar{s}' = p(\bar{s})$$
$$\bar{s} = \langle \bar{e}_i \rangle$$
$$\bar{s}' = \langle \bar{e}'_i \rangle$$

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

Perfectly Synchronous

$$\bar{s}' = \Delta(\bar{s})$$
$$\bar{s} = \langle \bar{e}_i \rangle$$
$$\bar{s}' = \langle \bar{e}'_i \rangle$$
$$\bar{e}'_i = \bar{e}_{i-1} \qquad \forall i \in \mathbb{N}$$
$$\bar{e}'_0 = \perp$$

Clocked Synchronous

Extended Characteristic Function -2

Definition: Let p a CS-MoC process.

$$
\begin{aligned}
f_{p,\bar{s}=\langle\bar{e}_i\rangle}(\bar{e}_j) &= \bar{e}'_j, \\
\text{with } p(\bar{s}) &= \bar{s}', \\
\bar{s}, \bar{s}' &\in \bar{S}, \\
\langle\bar{e}_i\rangle &= \bar{s}, \\
\langle\bar{e}'_i\rangle &= \bar{s}'
\end{aligned}
$$

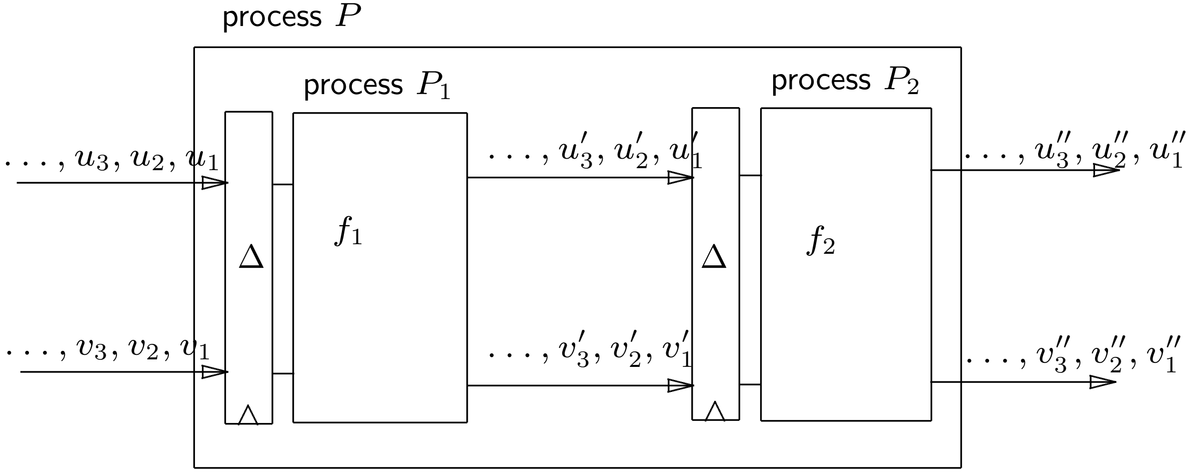
is an extended characteristic function of process p if it is defined in terms of an arbitrary functional expression of $\bar{e}_i, i \leq j$.

Example:

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

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

Clocked Synchronous Example



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

$$P_1 = \mathtt{map2CS}(f_1)$$
$$P_2 = \mathtt{mapCS}(f_2)$$

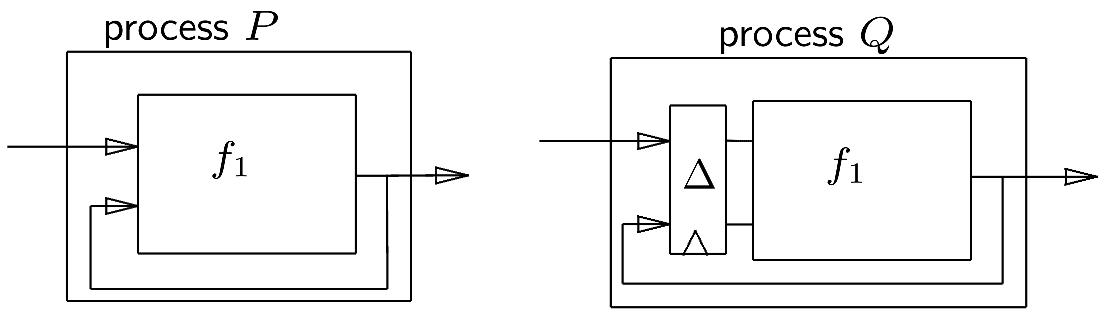
$$f_1((\bot, \bot)) = (0, 0) \qquad f_2((\bot, \bot)) = (0, 0)$$
$$f_1((x, \bot)) = (x, x) \qquad f_2((x, \bot)) = (x, x)$$
$$f_1((\bot, y)) = (y, -y) \qquad f_2((\bot, y)) = (-y, y)$$
$$f_1((x, y)) = (x + y, x - y) \qquad f_2((x, y)) = (x - y, x + y)$$

Clocked Synchronous Example - cont'd

$$
\begin{aligned}
f_1((\perp,\perp)) &= (0,0) & f_2((\perp,\perp)) &= (0,0)\\
f_1((x,\perp)) &= (x,x) & f_2((x,\perp)) &= (x,x)\\
f_1((\perp,y)) &= (y,-y) & f_2((\perp,y)) &= (-y,y)\\
f_1((x,y)) &= (x+y,x-y) & f_2((x,y)) &= (x-y,x+y)
\end{aligned}
$$

$$
\begin{aligned}
f_P((x_i,y_i)) &= f_2(f_\Delta(f_1(f_\Delta((x_i,y_i)))))\\
&= f_2(f_\Delta(f_1((x_{i-1},y_{i-1}))))\\
&= f_2(f_\Delta((x_{i-1}+y_{i-1},x_{i-1}-y_{i-1})))\\
&= f_2((x_{i-2}+y_{i-2},x_{i-2}-y_{i-2}))\\
&= (x_{i-2}+y_{i-2}-x_{i-2}+y_{i-2},x_{i-2}+y_{i-2}+x_{i-2}-y_{i-2})\\
&= (2y_{i-2},2x_{i-2}) \text{ for } i>1\\
f_P((x_i,y_i)) &= 0 \text{ for } 0\le i\le 1
\end{aligned}
$$

Clocked Synchronous with Feed-back



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

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

$$\begin{aligned} f_P(x) &= z \text{ where} \\ z &= f_1(x, z) \\ &= 2z - 2x \\ -z &= -2x \\ z &= 2x \end{aligned}$$

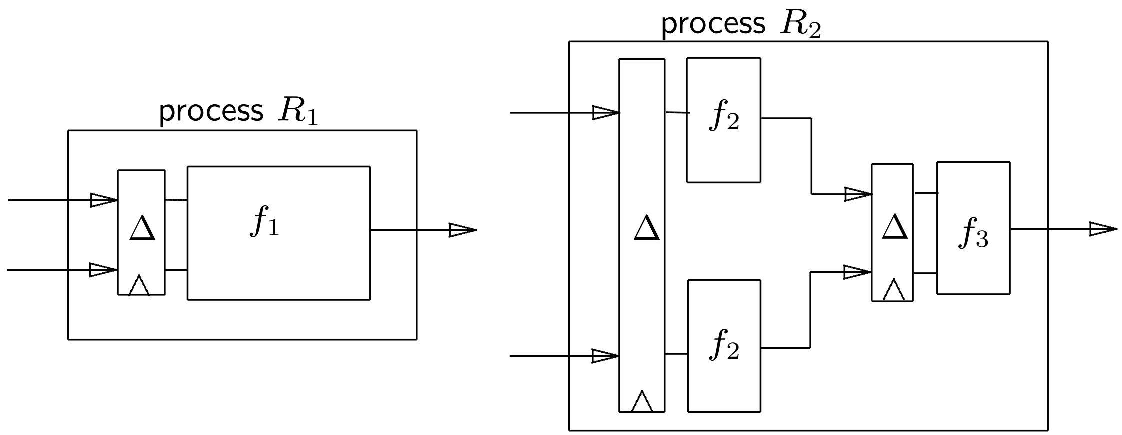
$$P([0, 1, 2, 3]) = [0, 2, 4, 6]$$

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

$$\begin{aligned} f_Q(x_i) &= z_i \text{ where} \\ z_i &= f_1(f_\Delta(x_i, z_i)) \\ &= f_1(x_{i-1}, z_{i-1}) \\ &= 2z_{i-1} - 2x_{i-1} \text{ for } i > 0 \\ z_0 &= f_1(0, 0) = 0 \end{aligned}$$

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

Process Substitution Example

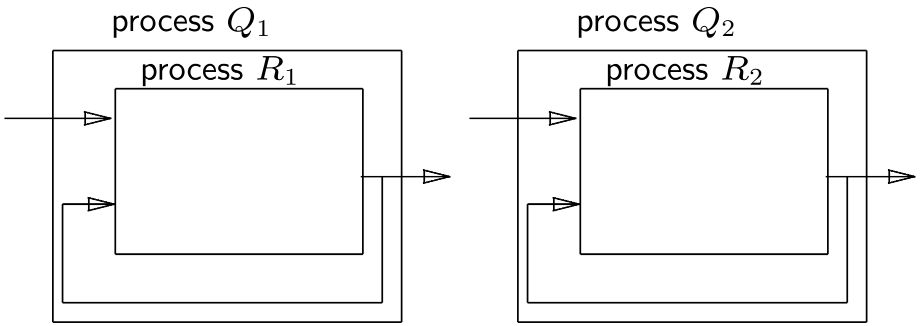


$$f_1(x, y) = 2y - 2x$$

$$f_2(x) = 2x$$

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

Process Substitution Example - cont'd



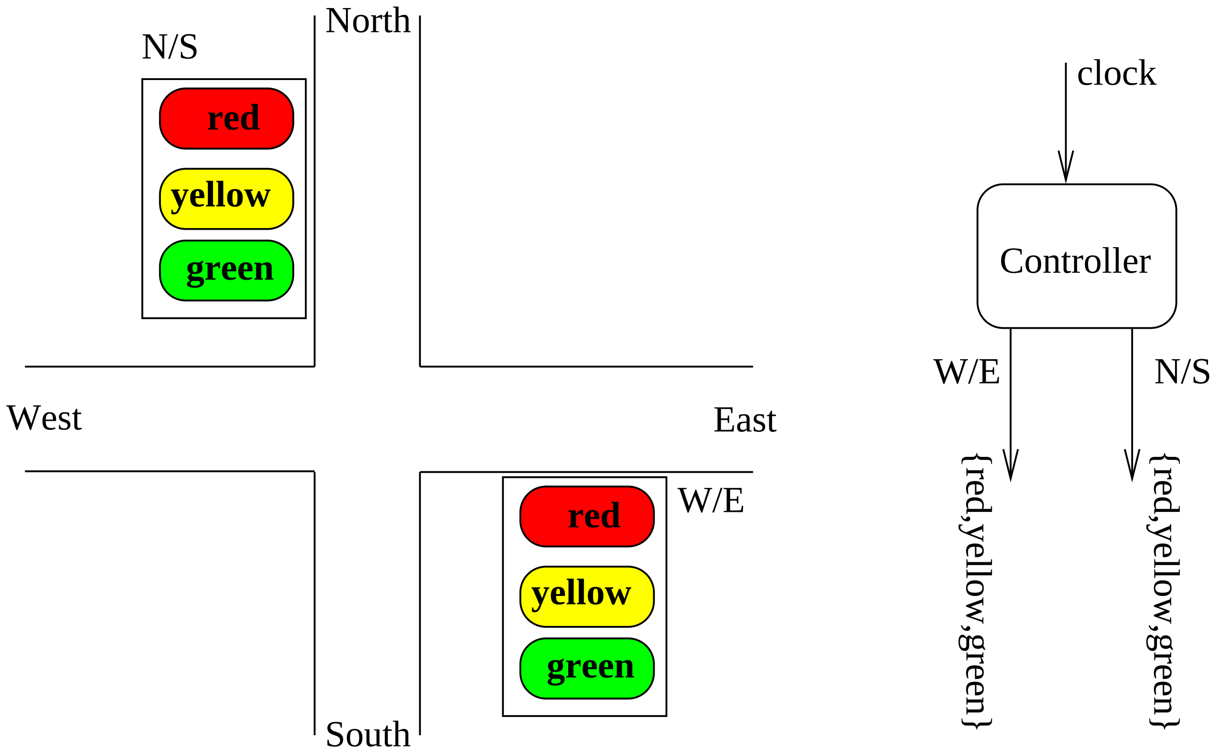
$$
\begin{aligned}
f_{Q_1}(u_i) &= z_i \quad \text{where} \\
z_i &= f_{R_1}(x_i, z_i) \\
&= f_1(f_\Delta(x_i, z_i)) \\
&= f_1(x_{i-1}, z_{i-1}) \\
&= 2z_{i-1} - 2x_{i-1} \quad \text{for } i \leq 1 \\
z_0 &= 0
\end{aligned}
$$

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

$$
\begin{aligned}
f_{Q_2}(u_0) &= 0 \\
f_{Q_2}(u_1) &= 0 \\
f_{Q_2}(u_i) &= z_i \quad \text{where} \\
z_i &= f_3(f_\Delta((f_2(f_\Delta(u_i)), f_2(f_\Delta(v_i))))) \\
&= f_3(f_\Delta((f_2(u_{i-1}), f_2(v_{i-1})))) \\
&= f_3(f_\Delta((2u_{i-1}, 2v_{i-1}))) \\
&= f_3((2u_{i-2}, 2v_{i-2})) \\
&= 2v_{i-2} - 2u_{i-2} \\
z_i &= v_i \\
z_i &= 2v_{i-2} - 2u_{i-2} \quad \text{for } i \geq 2
\end{aligned}
$$

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

The Traffic Light Controller



$$mooreS(nsf, outf, (\mathtt{rr1}, 0)) = \mathtt{tlctrl}$$

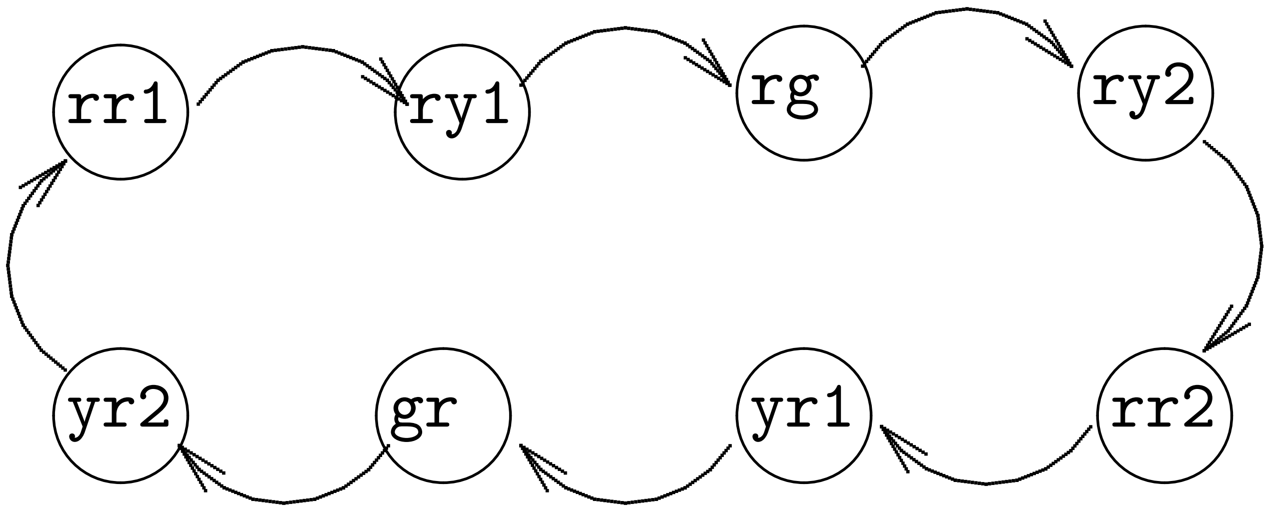
where $\mathtt{tlctrl}(s_c) = s_l$

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

The Traffic Light Controller - cont'd

$$
\begin{array}{llllll}
nsf & :: Clock & \rightarrow State & & & \rightarrow State \\
nsf & c & (\texttt{rr1}, \texttt{cnt}) & | \ \texttt{cnt} < 3 & = & (\texttt{rr1}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{ry1}, \ 0) \\
nsf & c & (\texttt{ry1}, \texttt{cnt}) & | \ \texttt{cnt} < 3 & = & (\texttt{ry1}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{rg}, \ 0) \\
nsf & c & (\texttt{rg}, \texttt{cnt}) & | \ \texttt{cnt} < 60 & = & (\texttt{rg}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{ry2}, \ 0) \\
nsf & c & (\texttt{ry2}, \texttt{cnt}) & | \ \texttt{cnt} < 3 & = & (\texttt{ry2}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{rr2}, \ 0) \\
nsf & c & (\texttt{rr2}, \texttt{cnt}) & | \ \texttt{cnt} < 3 & = & (\texttt{rr2}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{yr1}, \ 0) \\
nsf & c & (\texttt{yr1}, \texttt{cnt}) & | \ \texttt{cnt} < 3 & = & (\texttt{yr1}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{gr}, \ 0) \\
nsf & c & (\texttt{gr}, \texttt{cnt}) & | \ \texttt{cnt} < 60 & = & (\texttt{gr}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{yr2}, \ 0) \\
nsf & c & (\texttt{yr2}, \texttt{cnt}) & | \ \texttt{cnt} < 3 & = & (\texttt{yr2}, \ \texttt{cnt} + 1) \\
 & & & | \ otherwise & = & (\texttt{rr1}, \ 0)
\end{array}
$$

Traffic Light Controller - cont'd



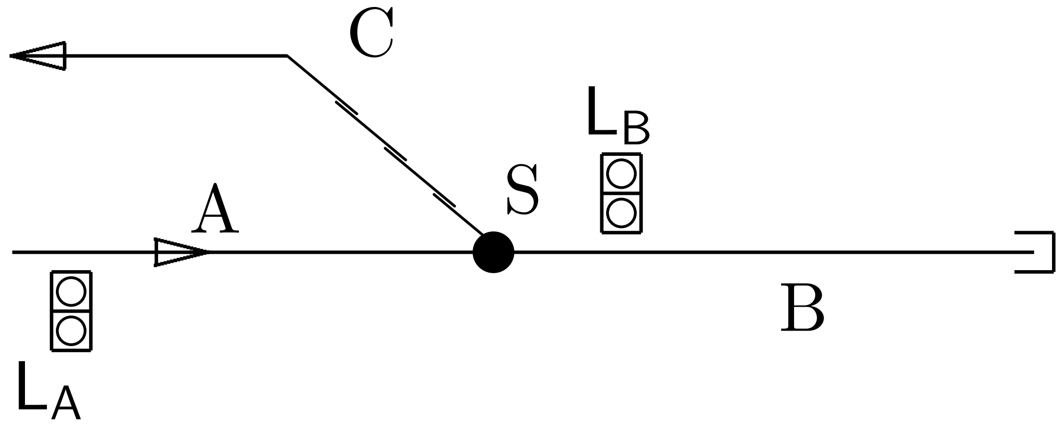
Traffic Light Controller - cont'd

$$
\begin{array}{llcl}
\mathit{outf} & ::\mathit{State} & \rightarrow & (\mathit{Color}, \mathit{Color}) \\
\mathit{outf} & (\texttt{rr1}, \texttt{cnt}) & = & (\texttt{red},\ \texttt{red}) \\
\mathit{outf} & (\texttt{rr2}, \texttt{cnt}) & = & (\texttt{red},\ \texttt{red}) \\
\mathit{outf} & (\texttt{ry1}, \texttt{cnt}) & = & (\texttt{red},\ \texttt{yellow}) \\
\mathit{outf} & (\texttt{ry2}, \texttt{cnt}) & = & (\texttt{red},\ \texttt{yellow}) \\
\mathit{outf} & (\texttt{rg},\ \texttt{cnt}) & = & (\texttt{red},\ \texttt{green}) \\
\mathit{outf} & (\texttt{yr1}, \texttt{cnt}) & = & (\texttt{yellow},\ \texttt{red}) \\
\mathit{outf} & (\texttt{yr2}, \texttt{cnt}) & = & (\texttt{yellow},\ \texttt{red}) \\
\mathit{outf} & (\texttt{gr},\ \texttt{cnt}) & = & (\texttt{green},\ \texttt{red})
\end{array}
$$

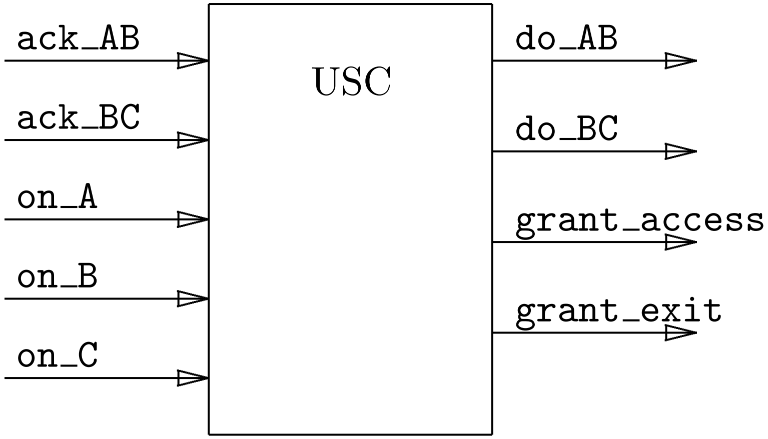
Traffic Light Controller - cont'd

$$
\begin{array}{llllll}
outf2 & ::State & & & \rightarrow & (Color, Color) \\
outf2 & (\mathtt{rr1}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\mathtt{red}, \bot) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{rr2}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\bot, \mathtt{red}) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{ry1}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\bot, \mathtt{yellow}) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{ry2}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\bot, \mathtt{yellow}) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{rg}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\bot, \mathtt{green}) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{yr1}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\mathtt{yellow}, \bot) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{yr2}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\mathtt{yellow}, \bot) \\
 & & | & otherwise & = & (\bot, \bot) \\
outf2 & (\mathtt{gr}, \mathtt{cnt}) & | & \mathtt{cnt} == 0 & = & (\mathtt{green}, \bot) \\
 & & | & otherwise & = & (\bot, \bot)
\end{array}
$$

Validation Example: U-turn Section Controller

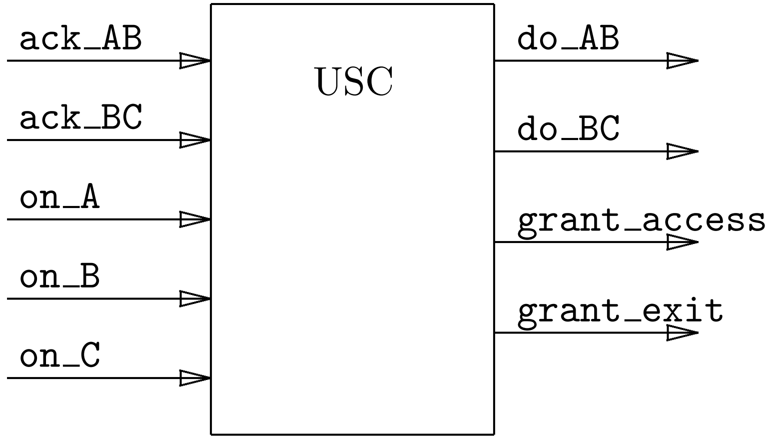


U-turn Section Controller - cont'd



- 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.

U-turn Section Controller - cont'd



$$\mathrm{USC} = mapS(f)$$

$$\text{where } f(\mathtt{onA}, \mathtt{onB}, \mathtt{onC}, \mathtt{ackAB}, \mathtt{ackBC}) = (\mathtt{grantAccess}, \mathtt{grantExit}, \mathtt{doAB}, \mathtt{doBC})$$

$$\mathtt{grantAccess} = \mathtt{emptySection} \wedge \mathtt{ackAB}$$

$$\mathtt{grantExit} = \mathtt{onlyOnB} \wedge \mathtt{ackBC}$$

$$\mathtt{doAB} = \neg\mathtt{ackAB} \wedge \mathtt{emptySection}$$

$$\mathtt{doBC} = \neg\mathtt{ackBC} \wedge \mathtt{onlyOnB}$$

$$\mathtt{emptySection} = \neg(\mathtt{onA} \vee \mathtt{onB} \vee \mathtt{onC})$$

$$\mathtt{onlyOnB} = \mathtt{onB} \wedge \neg(\mathtt{onA} \vee \mathtt{onC})$$

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.

Monitor Processes

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

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

Monitor Processes - cont'd

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

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

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

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

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

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

Monitors for the USC

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

is `True` when all sections are empty;

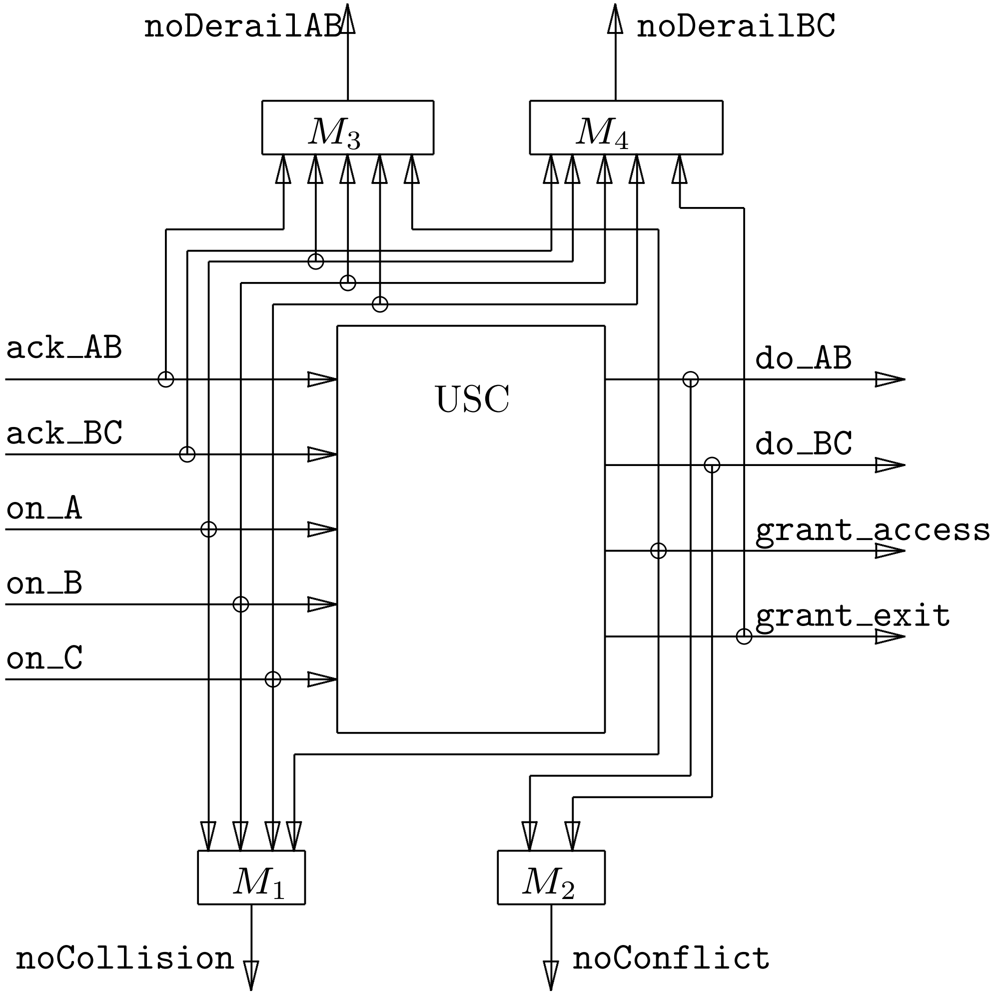
$$M_{oB}(s_1, s_2, s_3) = \texttt{mon2S}(f_2)(\texttt{zipS}()(s_1, s_2), s_3)$$
$$\text{where } f_2((b_1, b_2), b_3) = b_2 \wedge \neg(b_1 \vee b_3)$$

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

$$M_d(s_1, s_2, s_3) = implies(after(s_2), or(alwaysSince(s_1, s_2), onceSince(s_3, s_2)))$$

is `True` when the derail condition is `False`.

Monitors for Safety Properties of the US



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.

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

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

$$M_2(\texttt{doAB}, \texttt{doBC}) = \mathit{mon2S}(f)(\texttt{doAB}, \texttt{doBC}) \text{ where } f(b_1, b_2) = \neg(b_1 \wedge b_2)$$

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

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

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

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

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.