

System Modeling

Introduction

Rugby Meta-Model

Finite State Machines

Petri Nets

Untimed Model of Computation

Synchronous Model of Computation

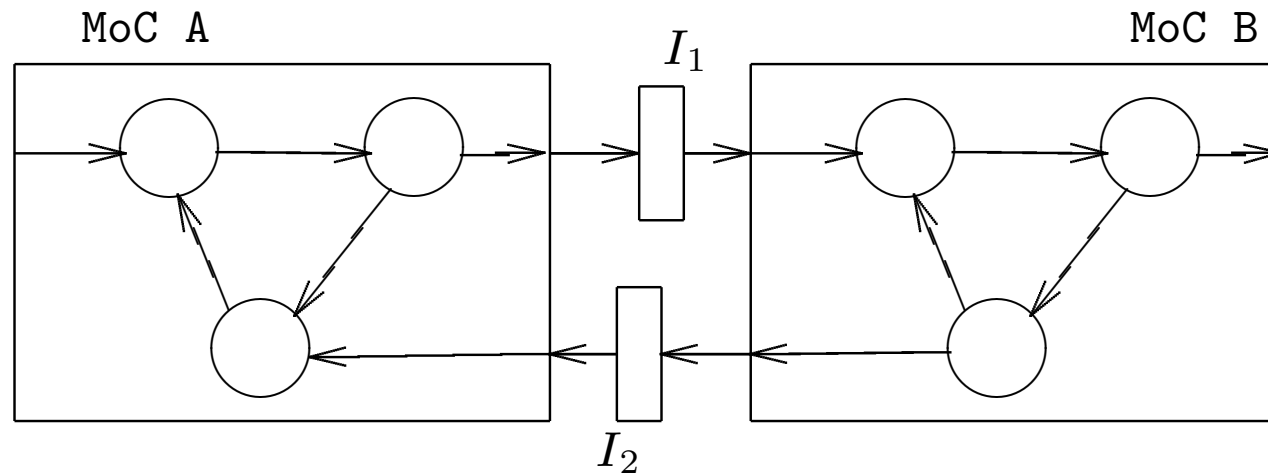
Timed Model of Computation

Integration of Computational Models

Tightly Coupled Process Networks



Interfaces between MoC Domains



If either MoC A or MoC B are synchronous or timed domains, the interfaces define the time relation between the two domains.

Interfaces between MoC Domains of the same Type

$$\mathit{intSup}(r, f) = \mathit{mapU}(1, f)$$

$$\text{with } \mathit{length}(f(\bar{e})) = r$$

$$\bar{e} \in \bar{E}, r \in \mathbb{N}$$

$$\mathit{intSdown}(r, f) = \mathit{mapU}(r, f)$$

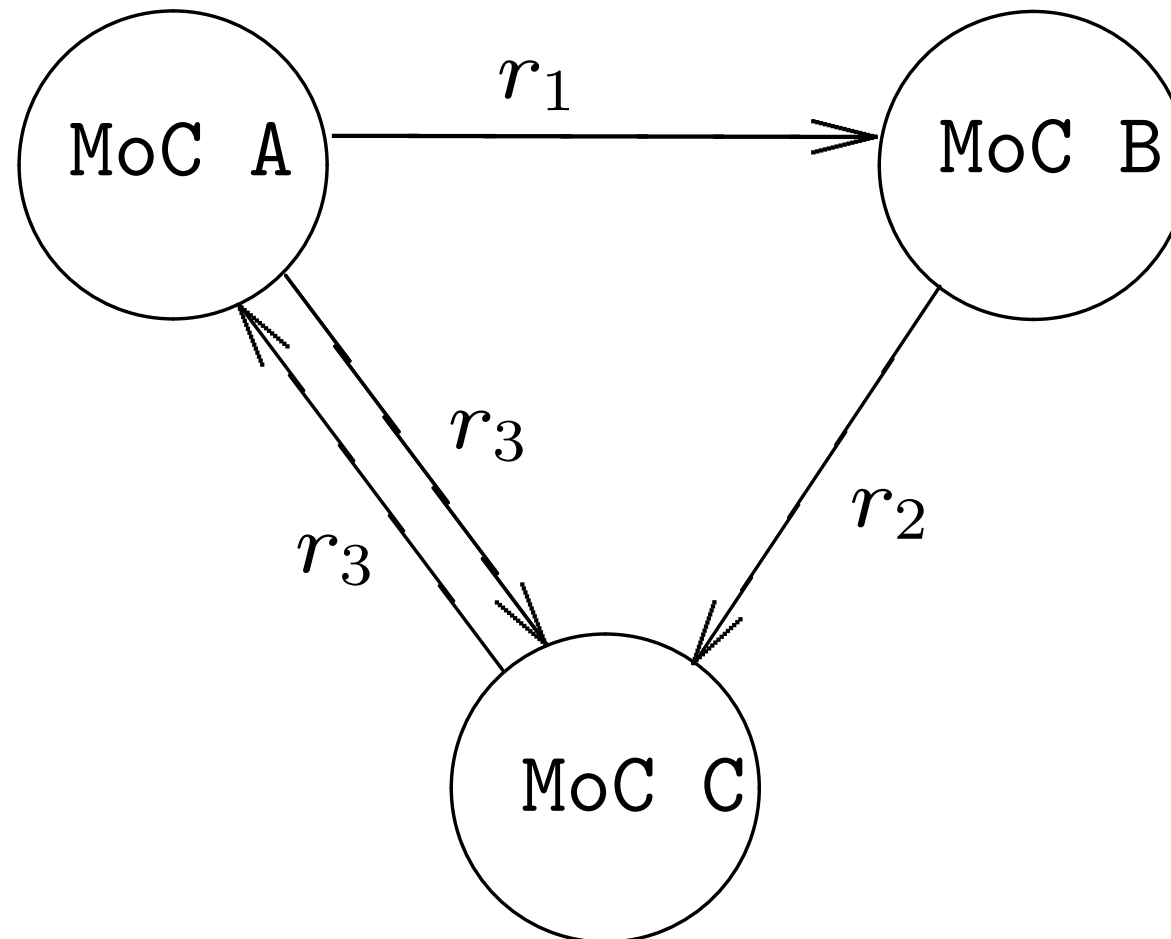
$$\text{with } \mathit{length}(f(\bar{a})) = 1$$

$$\bar{a} \in \bar{S}, r \in \mathbb{N}$$

$$\mathit{intTup} = \mathit{intSup}$$

$$\mathit{intTdown} = \mathit{intSdown}$$

Multiple Connected Domains



Interfaces Between MoC Domains

| from/to | Timed | Synchronous | Untimed |
|-------------|------------------|------------------|-----------------|
| Timed | - | <i>stripT2S</i> | <i>stripT2U</i> |
| Synchronous | <i>insertS2T</i> | - | <i>stripS2U</i> |
| Untimed | <i>insertU2T</i> | <i>insertU2S</i> | - |

Strip Based Interface Processes:

Timed \Rightarrow Untimed and Synchronous \Rightarrow Untimed

$$\text{stripT2U}() = p$$

$$\text{where } p(\hat{s}) = \dot{s}$$

$$\pi(\nu, \hat{s}) = \langle \hat{e}_i \rangle, \nu(i) = 1$$

$$\pi(\nu', \dot{s}) = \langle \dot{a}_i \rangle$$

$$\dot{a}_i = \begin{cases} \langle \rangle & \text{if } \dot{e}_i = \sqcup \\ \langle \dot{e}_i \rangle & \text{otherwise} \end{cases}$$

$$\nu'(i) = \begin{cases} 0 & \text{if } \dot{e}_i = \sqcup \\ 1 & \text{otherwise} \end{cases}$$

$$\text{stripS2U} = \text{stripT2U}$$

Strip Based Interface Processes: **Timed** \Rightarrow **Synchronous**

$$\text{stripT2S}(\lambda) = p$$

$$\text{where } p(\hat{s}) = \bar{s}$$

$$\pi(\nu, \hat{s}) = \langle \hat{a}_i \rangle, \nu(i) = \lambda$$

$$\pi(\nu', \bar{s}) = \langle \bar{e}_i \rangle, \nu'(i) = 1$$

$$\bar{e}_i = \begin{cases} \sqcup & \text{if strip}(\hat{a}_i) = \langle \rangle \\ \text{lastt}(\hat{a}_i) & \text{otherwise} \end{cases}$$

$$\text{for } \lambda \in \mathbb{N}, \hat{s}, \hat{a} \in \hat{S}, \bar{s} \in \bar{S}, \bar{e}_i \in \bar{E}, i \in \mathbb{N}_0$$

$\text{lastt}(\hat{s})$ denotes the last non-absent event in signal \hat{s}

Insert Based Interface Processes: **Untimed** \Rightarrow **Synchronous**

$$\text{insert}U2S(\lambda) = p$$

$$\text{where } p(\dot{s}) = \bar{s}$$

$$\pi(\nu, \dot{s}) = \langle \dot{e}_i \rangle, \nu(i) = 1$$

$$\pi(\nu', \bar{s}) = \langle \bar{a}_i \rangle, \nu'(i) = \lambda$$

$$\bar{a}_i = \langle \dot{e}_i \rangle \oplus \langle \sqcup \rangle^{\lambda-1}$$

$$\text{for } \lambda \in \mathbb{N}, \dot{s} \in \dot{S}, \bar{s}, \bar{a}_i \in \bar{S}, \dot{e}_i \in \dot{E}, i \in \mathbb{N}_0$$

Insert Based Interface Processes: **Synchronous** \Rightarrow **Timed**

$$\text{insertS2T}(\lambda) = p$$

$$\text{where } p(\bar{s}) = \hat{s}$$

$$\pi(\nu, \bar{s}) = \langle \bar{e}_i \rangle, \nu(i) = 1$$

$$\pi(\nu', \hat{s}) = \langle \hat{a}_i \rangle, \nu'(i) = \lambda$$

$$\hat{a}_i = \langle \bar{e}_i \rangle \oplus \langle \sqcup \rangle^{\lambda-1}$$

$$\text{for } \lambda \in \mathbb{N}, \bar{s} \in \bar{S}, \hat{s}, \hat{a} \in \hat{S}, \bar{e}_i \in \bar{E}, i \in \mathbb{N}_0$$

Hierarchical Model of Computation

Definition: A **Hierarchical Model of Computation (HMoC)** is a 3-tuple $\text{HMoC} = (M, C, O)$, where

M is a set of HMoCs or MoCs, each capable of instantiating processes;

C is a set of process constructors, each of which, when given constructor specific parameters, instantiates a process;

O is a set of process composition operators, each of which, when given processes as arguments, instantiates a new process.

With process we mean either an elementary process or a process network.

The Integrated Model of Computation

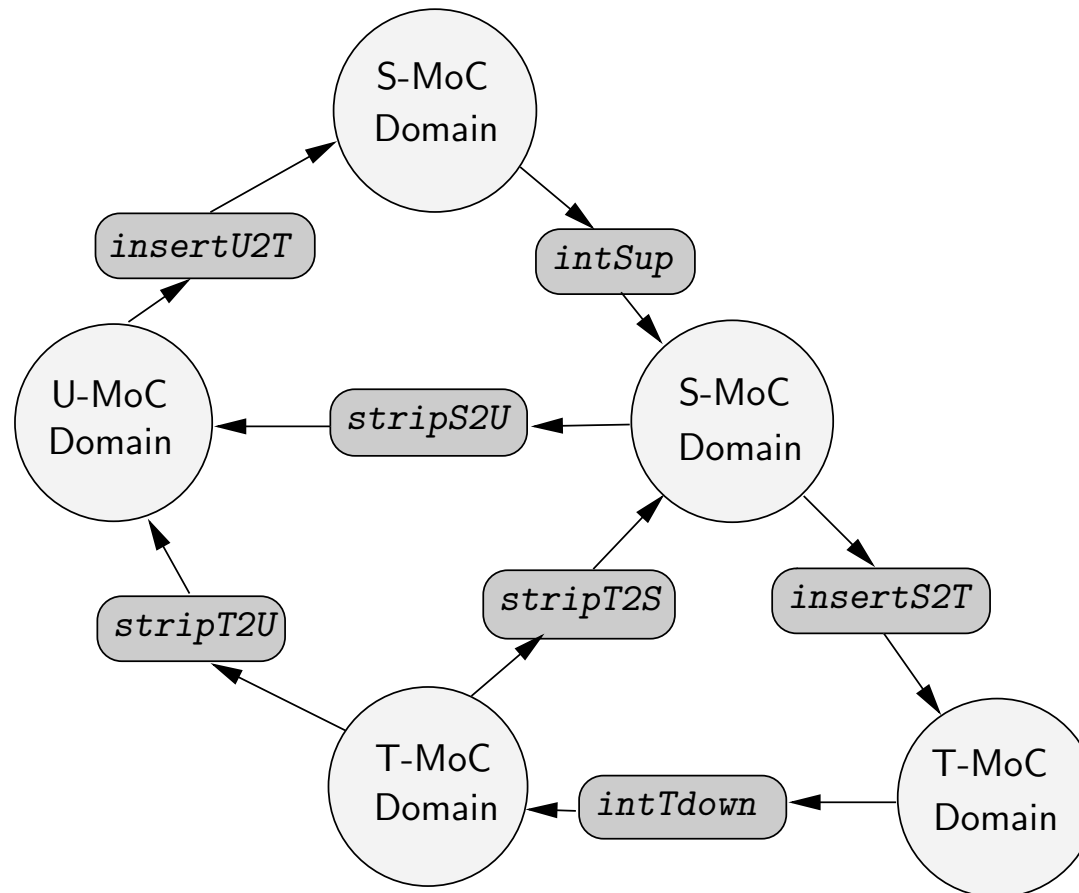
Definition: The **Integrated Model of Computation (Integrated MoC)** is defined as Integrated HMoC= (M, C, O) , where

$$M = \{U\text{-MoC}, S\text{-MoC}, T\text{-MoC}\}$$

$$C = \{ \textit{intSup}, \textit{intSdown}, \textit{intTup}, \textit{intTdown}, \\ \textit{stripT2S}, \textit{stripT2U}, \textit{stripS2U}, \\ \textit{insertS2T}, \textit{insertU2T}, \textit{insertU2S} \}$$

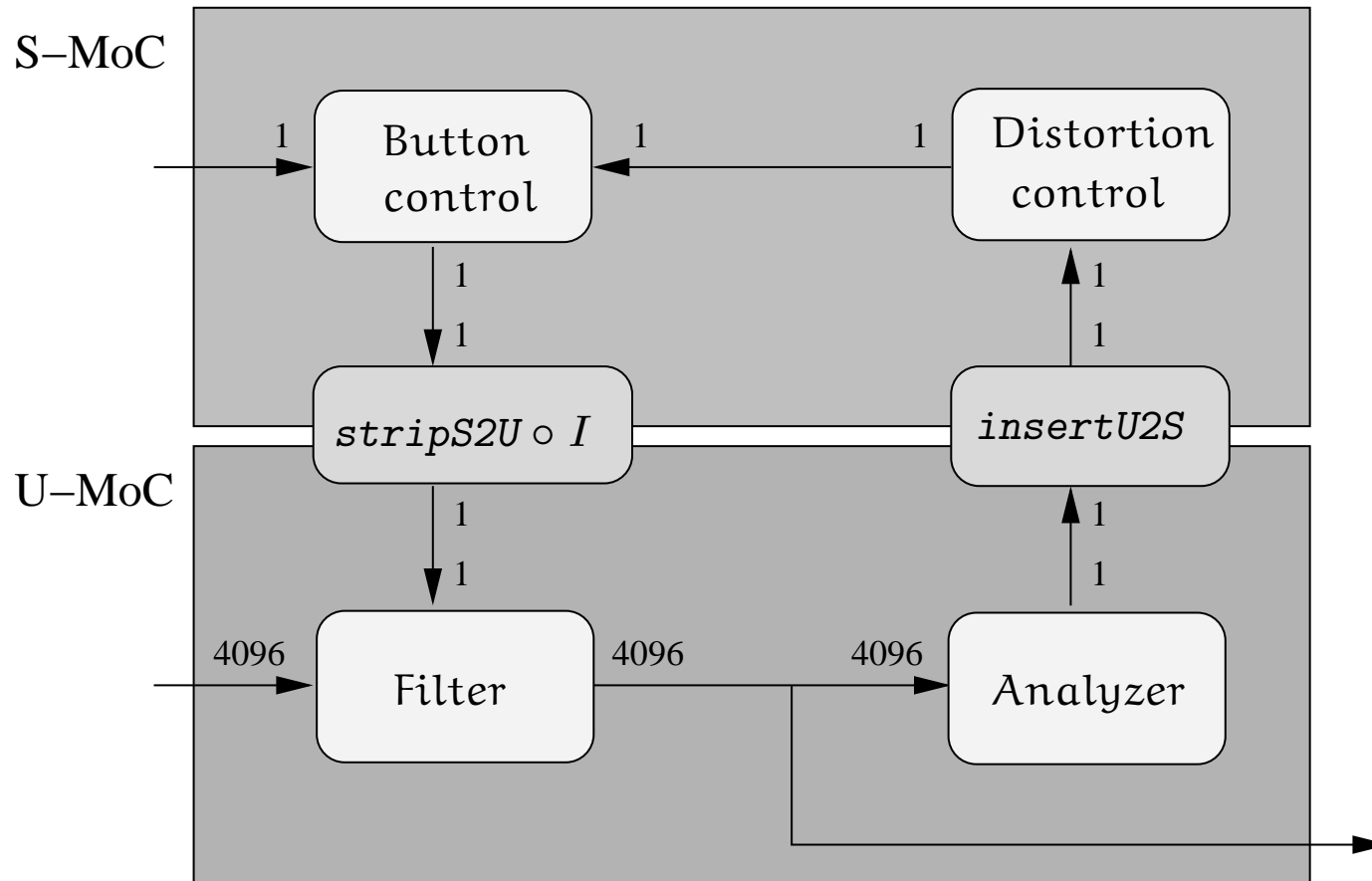
$$O = \{ \parallel, \circ, \mathbf{FB}_P \}$$

Systems with Multiple Sub-domains



Proper interfaces between domains are required.

A Digital Equalizer with two Sub-domains



Connecting MoC Domains Relates Time Structures

- Only U-MoC - U-MoC coupling does not couple time structures;
- Connecting an U-MoC to a S-MoC imposes the S-MoC time structure on the U-MoC domain.
- Interfaces can be modeled to define the time relation.
- Interface delays can be modeled stochastically or nondeterministically
 - ★ Channel behaviour becomes more realistic;
 - ★ Time structure relation becomes complex;
 - ★ Time structure coupling cannot be avoided;

MoC Interface Refinement

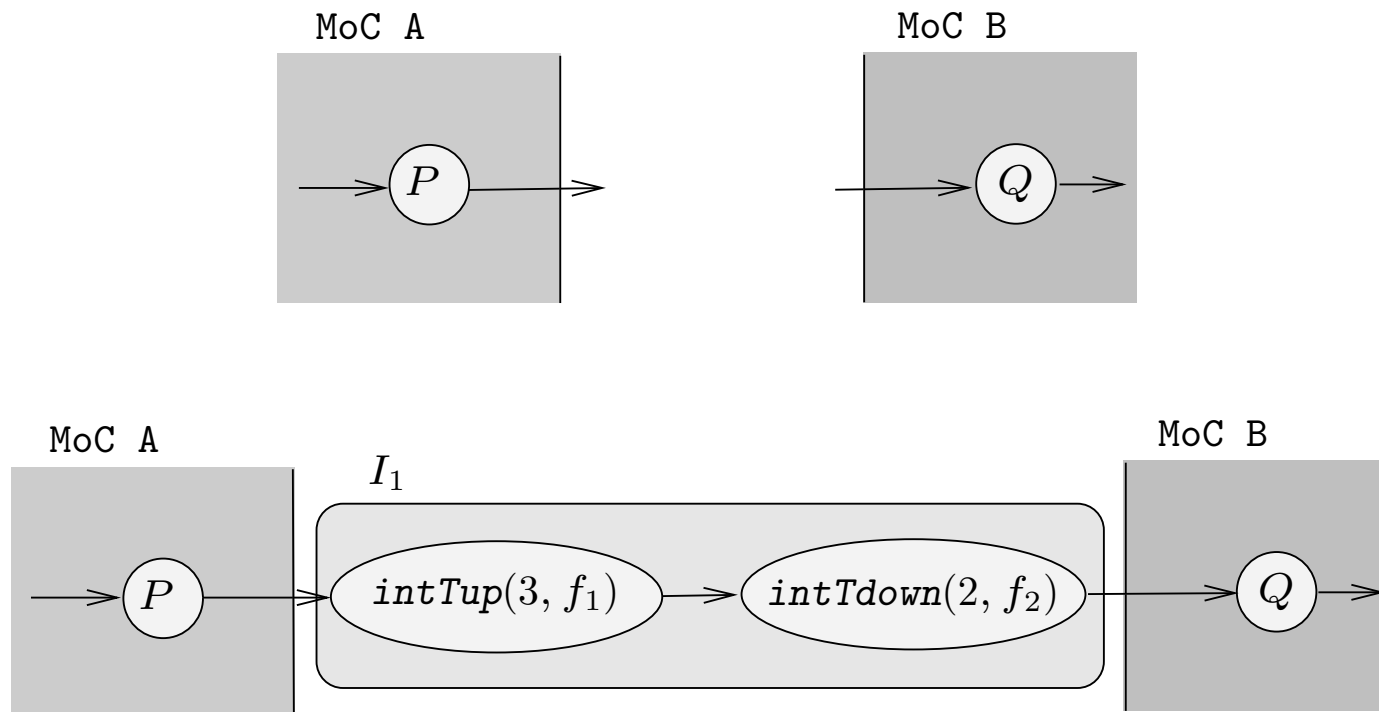
Add time interface: to precisely define the time structure relation. The relation can be constant, cyclic, deterministic, or stochastic.

Refine the protocol: Define and refine a protocol which allows for reliable communication across the domain boundary with the given time relation.

Model the channel delay: If desirable, model the channel delay deterministically or stochastically.

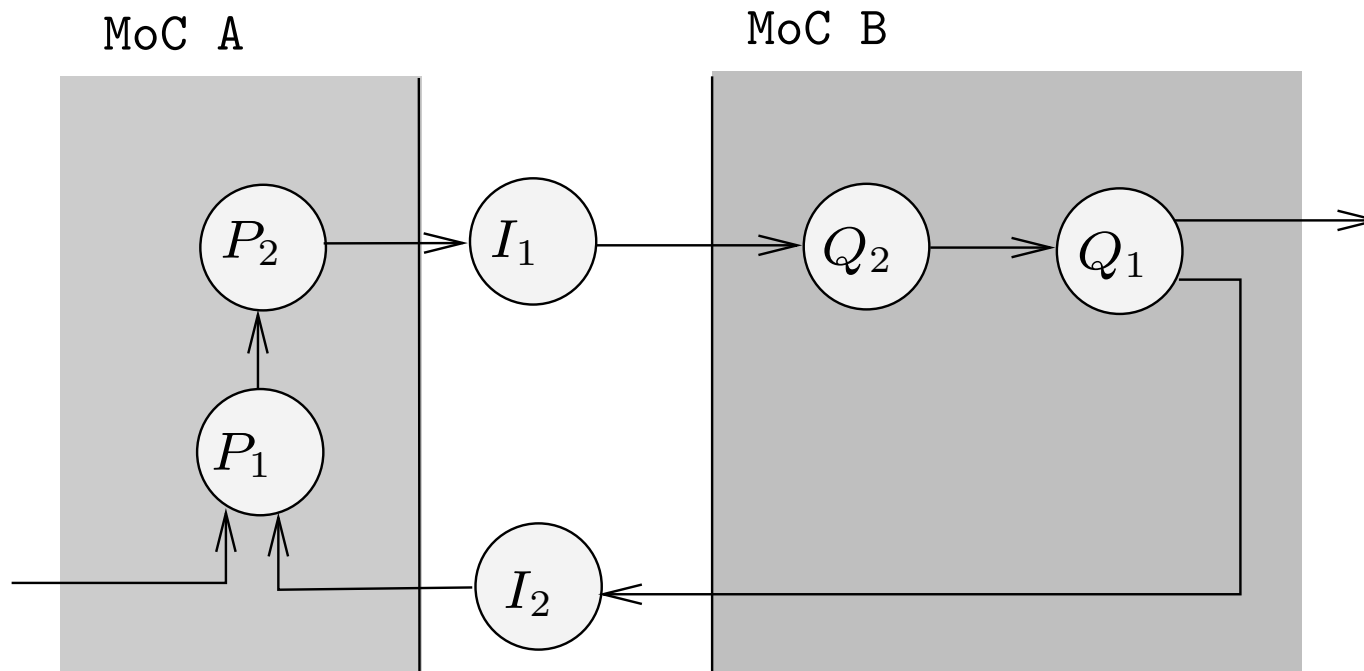
MoC Interface Refinement Example

Step 1 - Add time interface

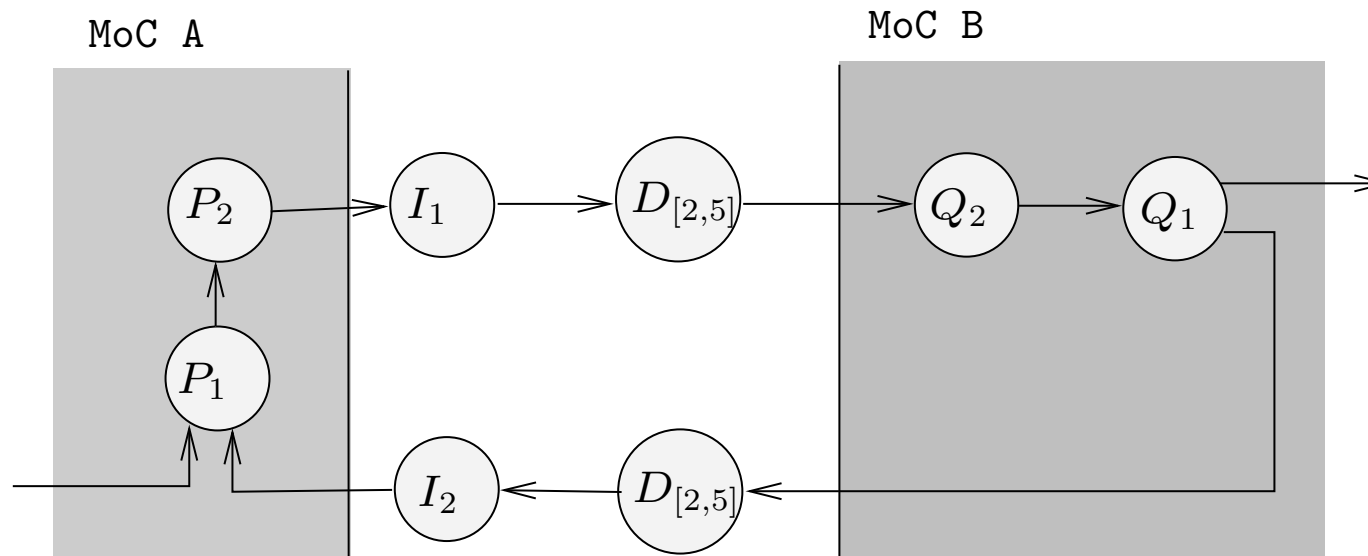


MoC Interface Refinement Example

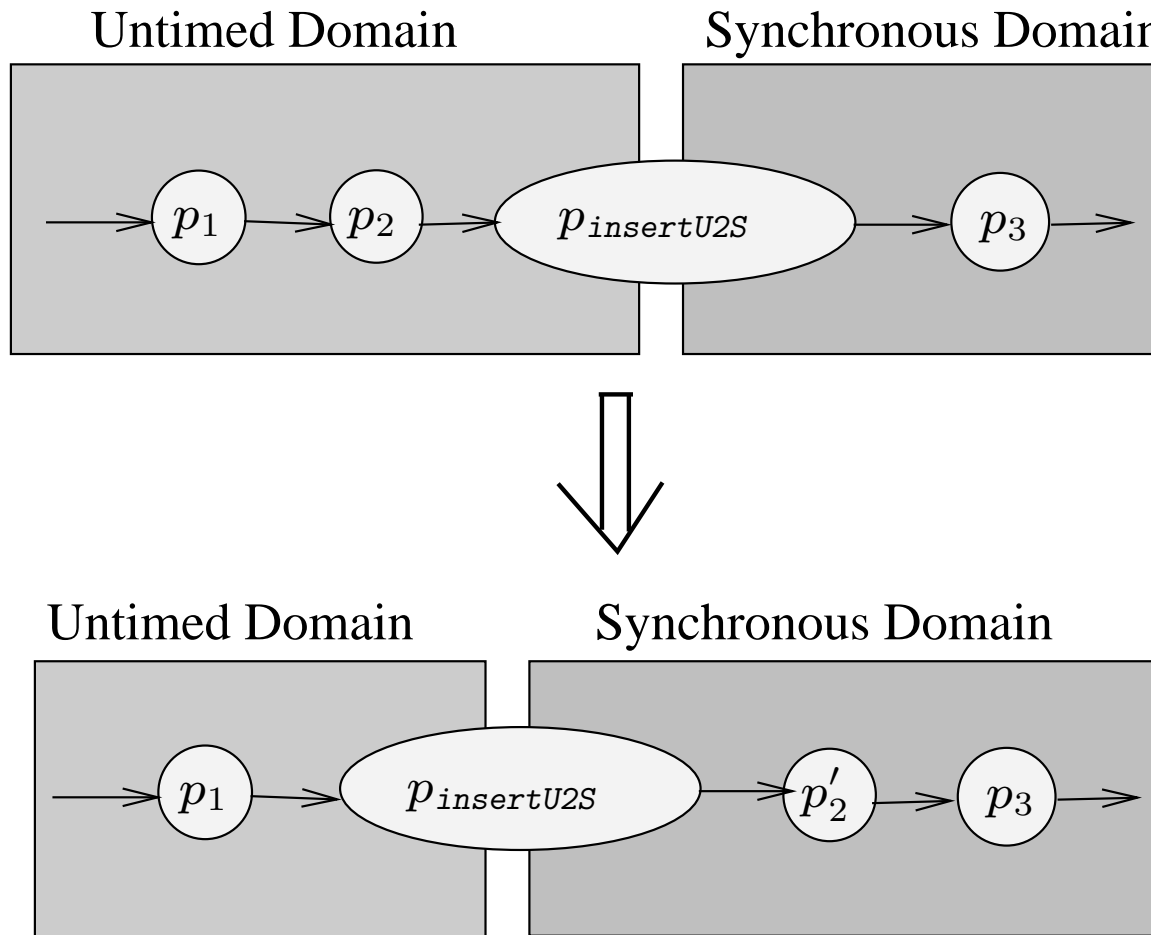
Step 2 - Refine the Protocol



MoC Interface Refinement Example Step 3 - Model the Channel Delay



Process Migration between MoC Domains



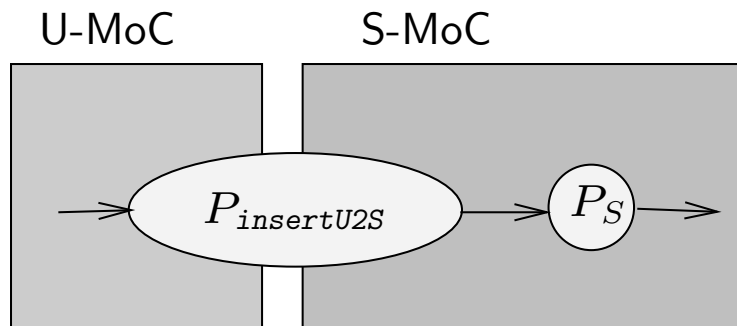
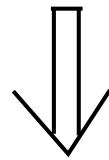
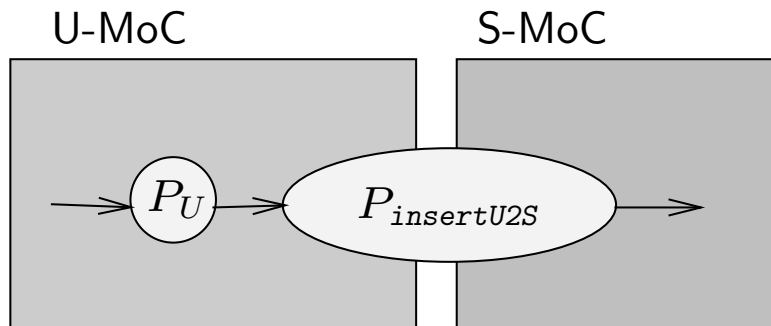
Process Migration Cases

| | | |
|------|----------------------------------|---|
| 1.a. | Untimed to Synchronous/downwards | $P_U \circ P_{insertU2S} \Rightarrow P_{insertU2S} \circ P_S$ |
| b. | Synchronous to Untimed/Upwards | $P_{insertU2S} \circ P_S \Rightarrow P_U \circ P_{insertU2S}$ |
| 2.a. | Untimed to Timed/downwards | $P_U \circ P_{insertU2T} \Rightarrow P_{insertU2T} \circ P_T$ |
| b. | Timed to Untimed/upwards | $P_{insertU2T} \circ P_T \Rightarrow P_U \circ P_{insertU2T}$ |
| 3.a. | Synchronous to Timed/downwards | $P_S \circ P_{insertS2T} \Rightarrow P_{insertS2T} \circ P_T$ |
| b. | Timed to Synchronous/upwards | $P_{insertS2T} \circ P_T \Rightarrow P_S \circ P_{insertS2T}$ |
| 4.a. | Timed to Synchronous/downwards | $P_T \circ P_{stripT2S} \Rightarrow P_{stripT2S} \circ P_S$ |
| b. | Synchronous to Timed/upwards | $P_{stripT2S} \circ P_S \Rightarrow P_T \circ P_{stripT2S}$ |
| 5.a. | Timed to Untimed/downwards | $P_T \circ P_{stripT2U} \Rightarrow P_{stripT2U} \circ P_U$ |
| b. | Untimed to Timed/upwards | $P_{stripT2U} \circ P_U \Rightarrow P_T \circ P_{stripT2U}$ |
| 6.a. | Synchronous to Untimed/downwards | $P_S \circ P_{stripS2U} \Rightarrow P_{stripS2U} \circ P_U$ |
| b. | Untimed to Synchronous/upwards | $P_{stripS2U} \circ P_U \Rightarrow P_S \circ P_{stripS2U}$ |

Helper Processes

$$\begin{array}{l}
 \text{par_c} = \text{mealyS}(g, f, \langle \rangle) \\
 \text{where } g(e, w) = \begin{cases} w \oplus e & \text{if } \text{length}(w) < c \\ \langle \rangle & \text{otherwise} \end{cases} \\
 f(e, w) = \begin{cases} w \oplus e & \text{if } \text{length}(w) = c \\ \perp & \text{otherwise} \end{cases} \\
 \hline
 \text{par} = p_2 \circ p_1 \\
 p_1 = \text{zipS}() \\
 p_2 = \text{mealyS}(g, f, (\langle \rangle, 0)) \\
 \text{where } g((e, c), (w, d)) = \begin{cases} (w \oplus e, d) & \text{if } \text{length}(w) < d \\ (\langle \rangle, c) & \text{otherwise} \end{cases} \\
 f((e, c), (w, d)) = \begin{cases} w \oplus e & \text{if } \text{length}(w) = d \\ \perp & \text{otherwise} \end{cases} \\
 \hline
 \text{ser} = \text{mooreS}(g, f, \langle \rangle) \\
 \text{where } g(e, w) = \begin{cases} \text{tail}(w) & \text{if } e = \perp \\ \text{tail}(w) \oplus e & \text{otherwise} \end{cases} \\
 f(w) = \begin{cases} \text{head}(w) & \text{if } w \neq \langle \rangle \\ \perp & \text{otherwise} \end{cases}
 \end{array}$$

Case 1.a: Untimed to Synchronous/downward - Stateless Process



Map based process:

$$P_U = \text{mapU}(c, f_1)$$

$$p_I = \text{insertU2S}(1)$$

$$P_S = q_3 \circ q_2 \circ q_1$$

$$q_1 = \text{par_c}$$

$$q_3 = \text{ser}$$

$$q_2 = \text{mapS}(f_2)$$

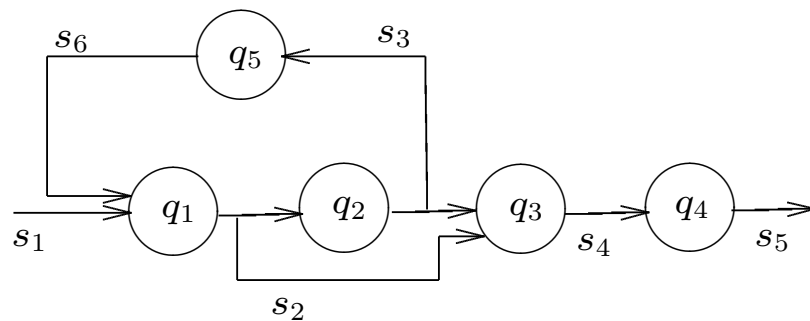
$$f_2(e) = \begin{cases} \perp & \text{if } e = \perp \\ \perp & \text{if } f_1(e) = \langle \rangle \\ f_1(e) & \text{otherwise} \end{cases}$$

Case 1.a: Untimed to Synchronous/downward - Statefull Process

Mealy based process:

$$p_U = \text{mealyU}(\gamma, g, f, w_0)$$

$$p_I = \text{insertU2S}(1)$$



$$P_S(s_1) = s_5$$

where

$$s_5 = q_4(s_4)$$

$$s_4 = q_3(s_3, s_2)$$

$$s_3 = q_2(s_2)$$

$$s_2 = q_1(s_1, s_6)$$

$$s_6 = q_5(s_3)$$

$$q_1 = \text{par}$$

$$q_2 = \text{scanS}(f_2, w_0)$$

where

$$g_2(e, w) = \begin{cases} \perp & \text{if } e = \perp \\ g(e, w) & \text{otherwise} \end{cases}$$

$$q_3 = \text{mapS}(f_3)$$

where

$$f_3(e, w) = \begin{cases} \perp & \text{if } e = \perp \\ f(e, w) & \text{otherwise} \end{cases}$$

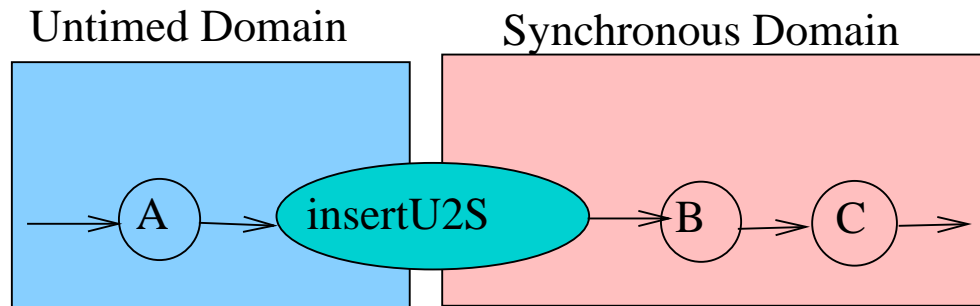
$$q_4 = \text{ser}$$

$$q_5 = \text{mapS}(\gamma_5)$$

where

$$\gamma_5(e) = \begin{cases} \perp & \text{if } e = \perp \\ \gamma(e) & \text{otherwise} \end{cases}$$

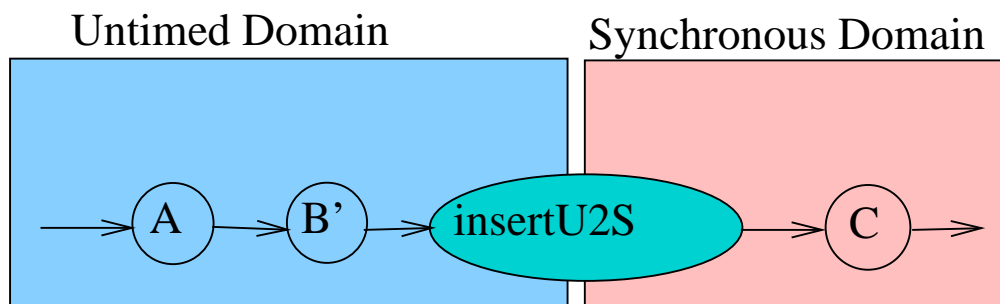
Case 1.b: Synchronous to Untimed/upwards



Map based process:

$$B = \text{mealyS}(g, f, w_0)$$

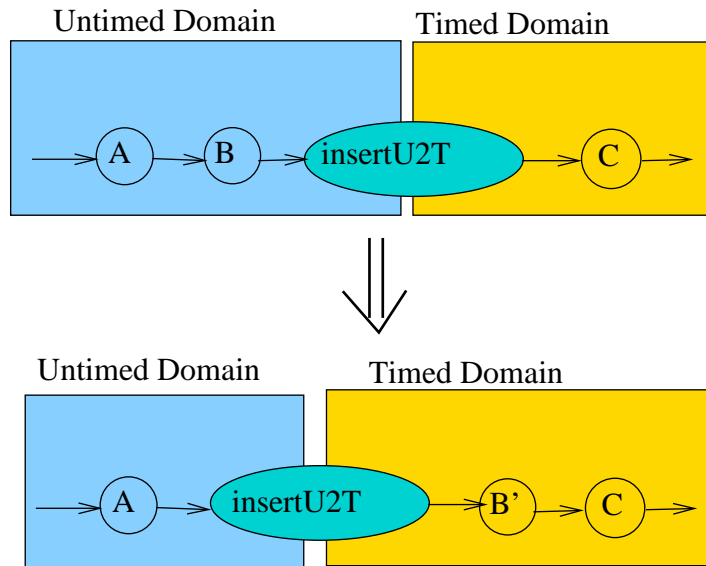
$$p_I = \text{insertU2S}(1)$$



$$B' = \text{mealyU}(1, g, f, w_0)$$

with $\nexists w, \dot{e} : f(\dot{e}, w) = \perp$

Case 2.a: Untimed to Timed/downwards



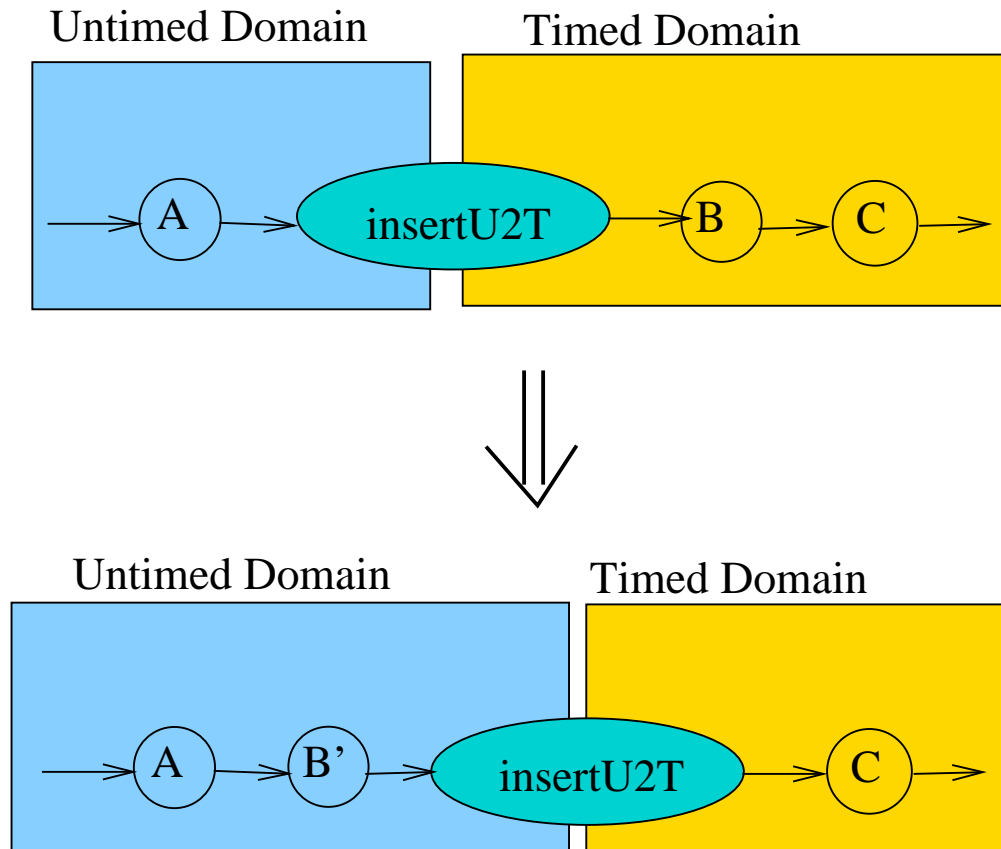
For the special case $\lambda = 1$:

$$B = \text{mealy}U(\gamma, g, f, w_0)$$

$$p_I = \text{insert}U2T(1)$$

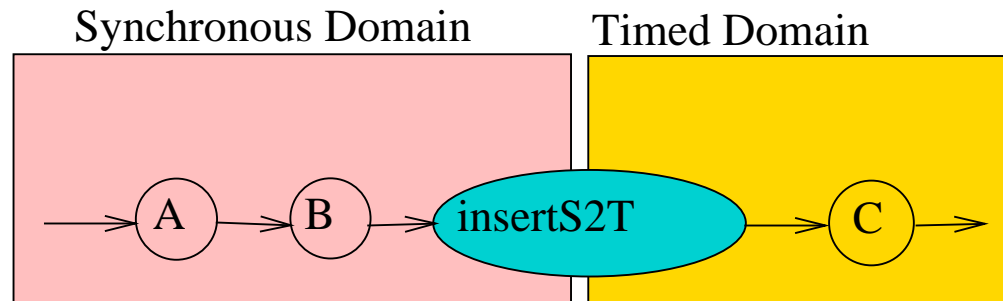
$$B' = \text{mealy}T(\gamma, g, f, w_0)$$

Case 2.b: Timed to Untimed/upwards



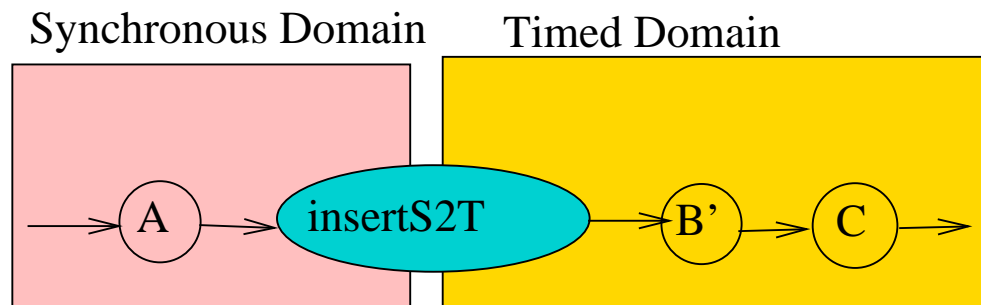
- Cannot be done in general due to lack of information
- B could be merged with the interface process
- Special cases could be considered

Case 3.a: Synchronous to Timed/downwards



$$B = \text{mealyS}(g, f, w_0)$$

$$p_I = \text{insertS2T}(\lambda)$$

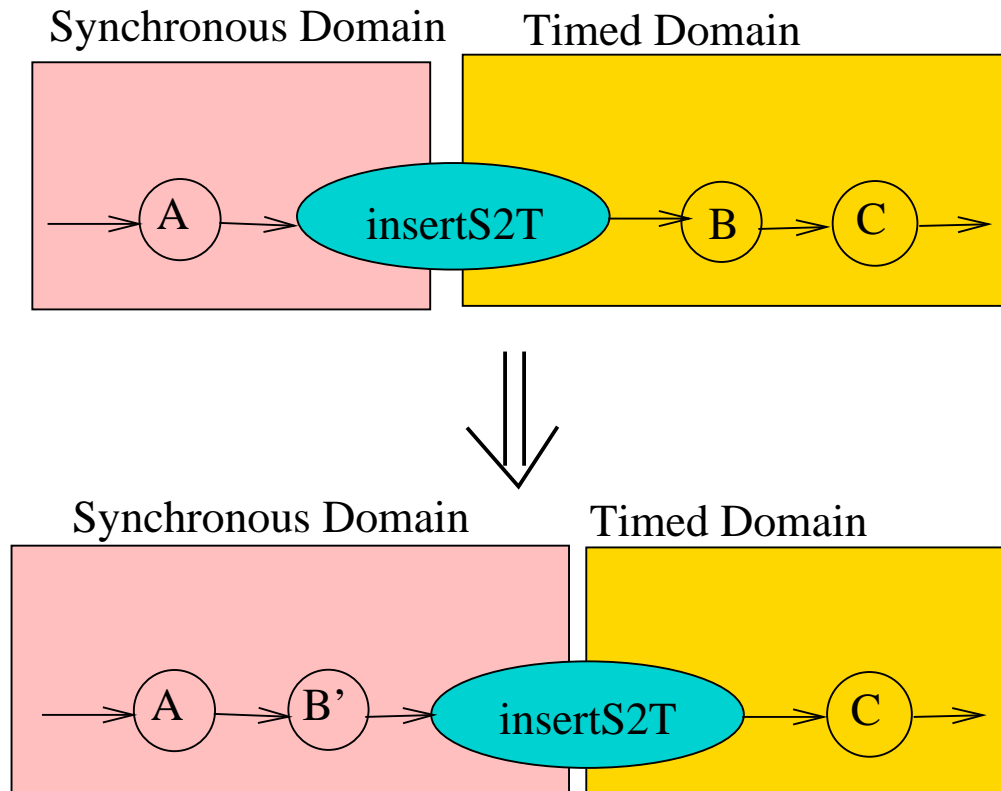


$$B' = \text{mealyT}(1, g, f', w_0)$$

where

$$f'(w, \hat{e}) = \langle f(w, \hat{e}) \rangle \oplus \langle \sqcup \rangle^{\lambda-1}$$

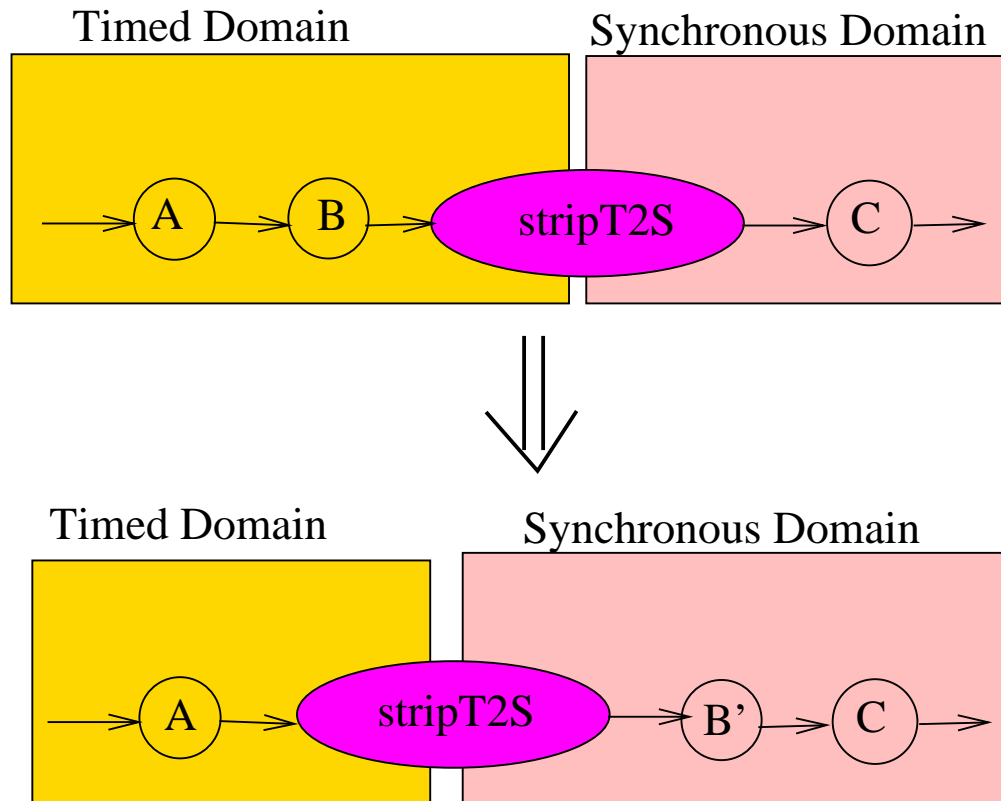
Case 3.b: Timed to Synchronous/upwards



This case is handled similarly to case 1a.

The input to the new process B' must be parallelized and the output must be sequentialized.

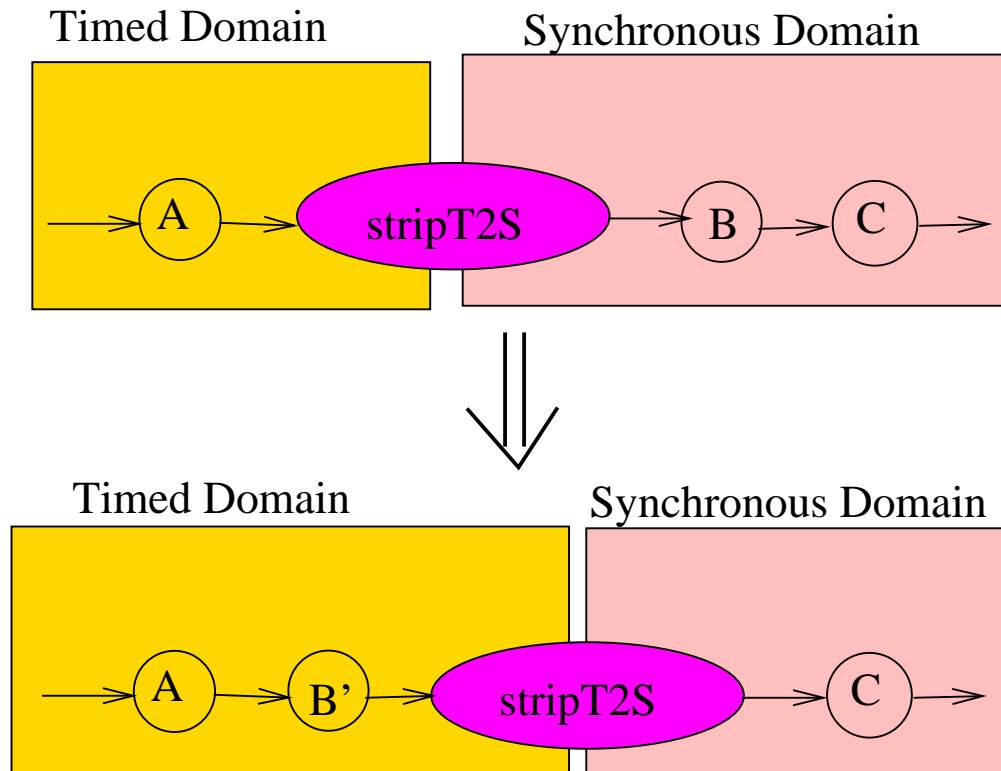
Case 4.a: Timed to Synchronous/downwards



This case is handled similarly to cases 1a and 3b.

The input to the new process B' must be parallelized and the output must be sequentialized.

Case 4.b: Synchronous to Timed/upwards



$$B = \text{mealyS}(g, f, w_0)$$

$$p_I = \text{stripT2S}(\lambda)$$

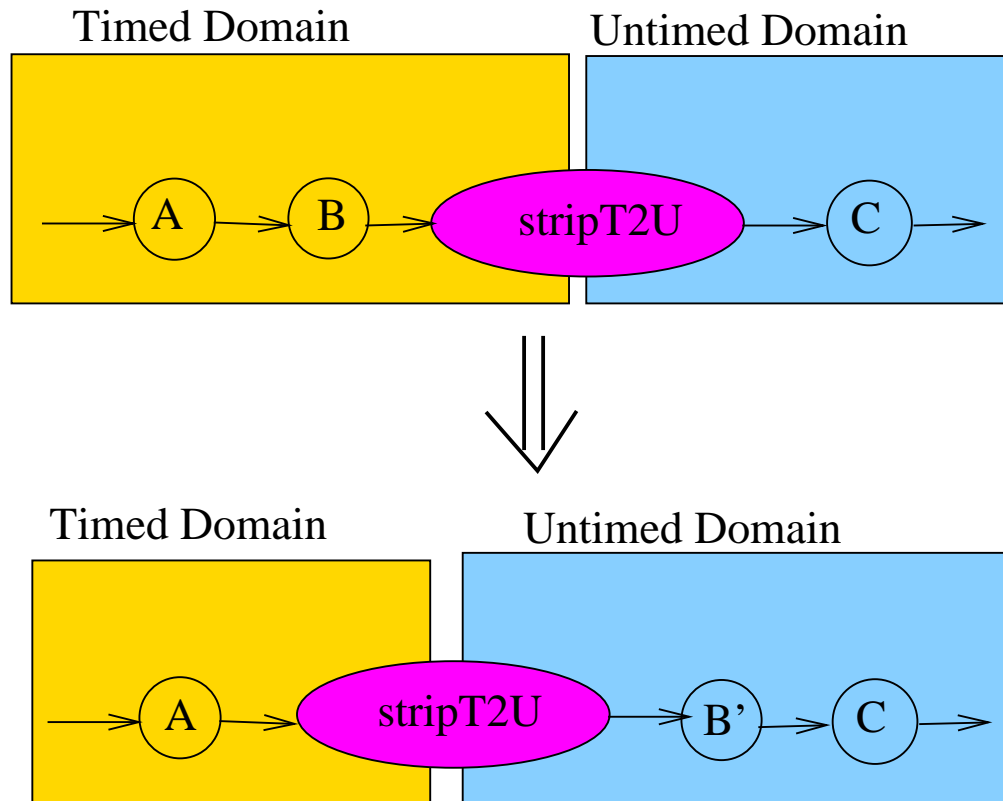
$$B' = \text{mealyT}(\lambda, g', f', w_0)$$

$$\text{where } g'(w, \hat{a}) = g(w, \text{lastt}(\hat{a}))$$

$$f'(w, \hat{a}) = f(w, \text{lastt}(\hat{a}))$$

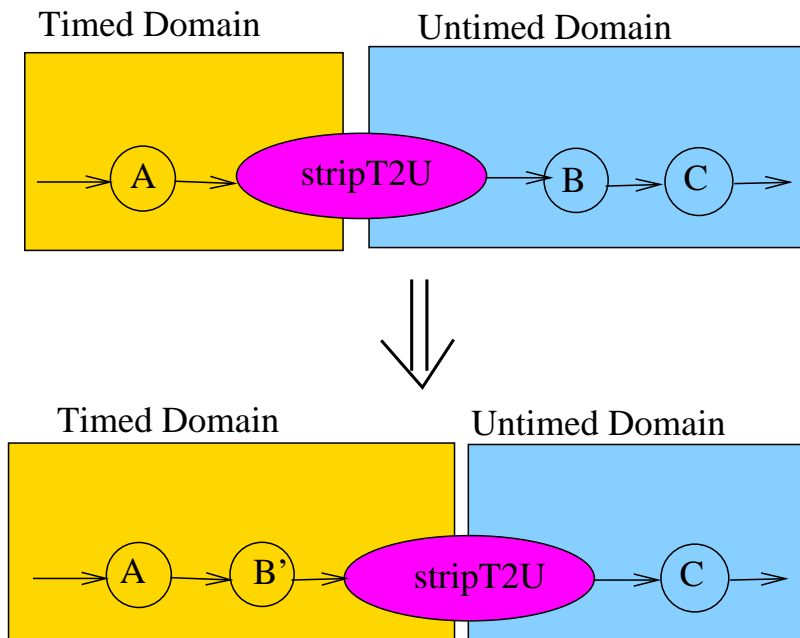
$$p'_I = \text{stripT2S}(1)$$

Case 5.a: Timed to Untimed/downwards



- Cannot be done in general due to lack of information
- B could be merged with the interface process
- Special cases could be considered

Case 5.b: Untimed to Timed/upwards



$$B = \text{mealyU}(\gamma, g, f, w_0)$$

$$p_I = \text{stripT2U}()$$

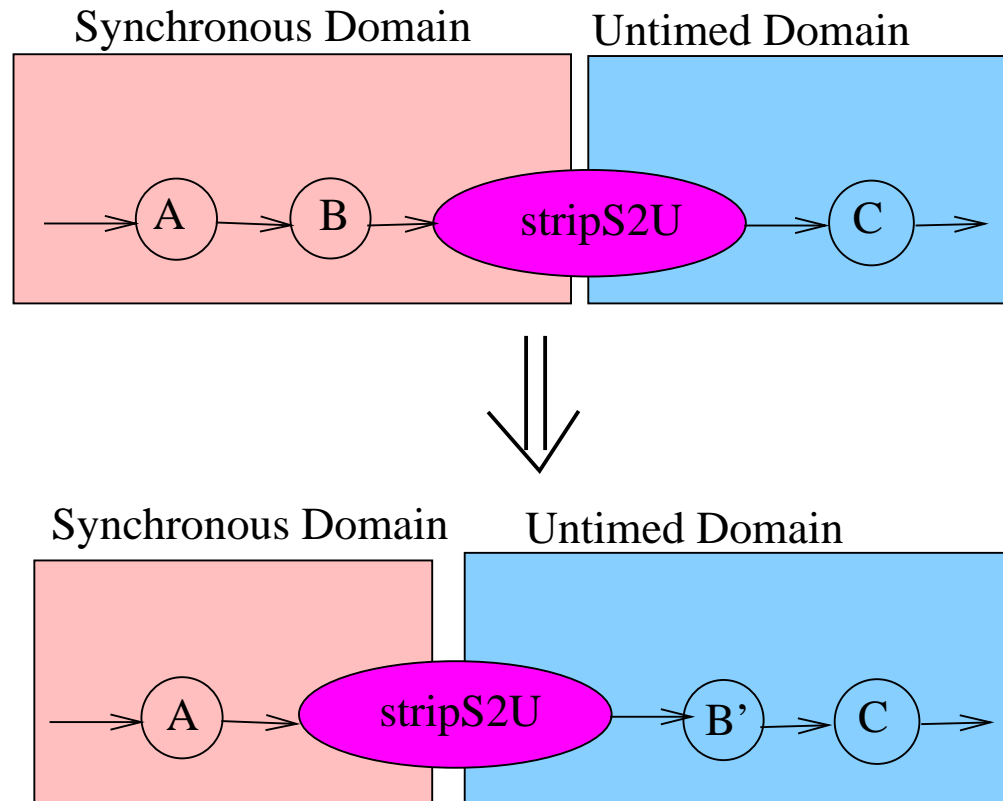
$$B' = q_1 \circ q_2$$

$$q_2 = \text{mapT}(1, f')$$

$$\text{where } f'(\hat{e}) = \begin{cases} \langle \rangle & \text{if } \hat{e} = \sqcup \\ \langle \hat{e} \rangle & \text{otherwise} \end{cases}$$

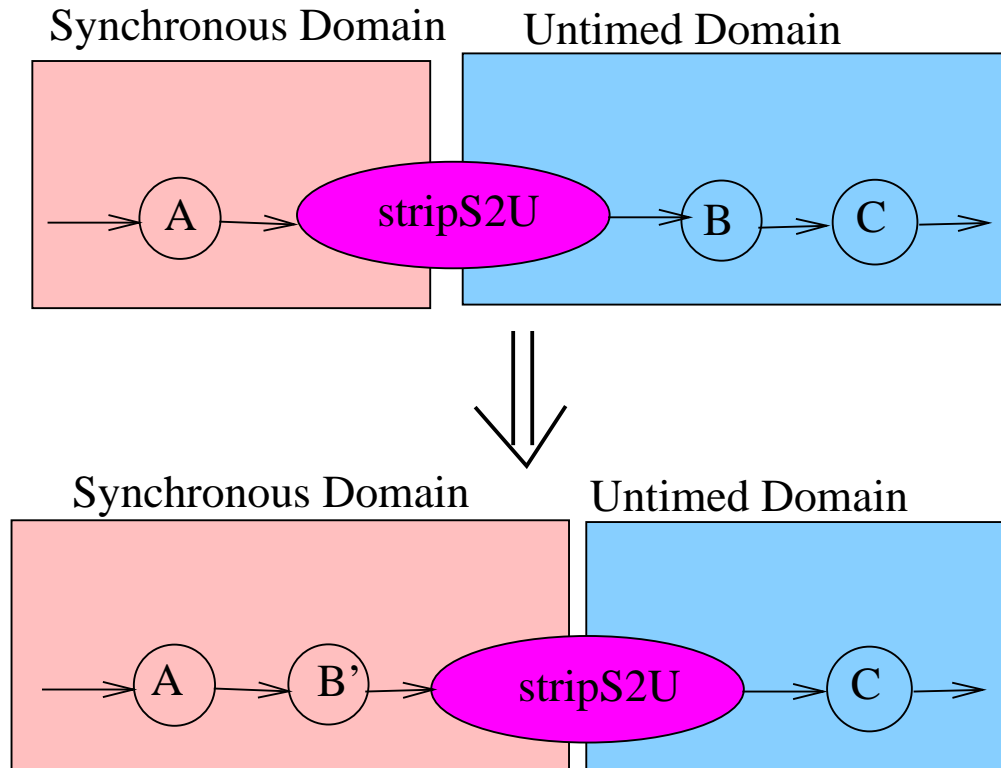
$$q_1 = \text{mealyT}(\gamma, g, f, w_0)$$

Case 6.a: Synchronous to Untimed/downwards



- Cannot be done in general due to lack of information
- B could be merged with the interface process
- Special cases could be considered

Case 6.b: Untimed to Synchronous/upwards



- This case is handled similarly to cases 1a, 3b and 4a.
- The input to the new process B' must be parallelized and the output must be sequentialized.
- In addition, absent events must be filtered out.