Causally Consistent Reversible Choreographies

A Monitors-as-Memories Approach

Claudio Antares Mezzina (IMT Lucca, IT)
Jorge A. Pérez (University of Groningen, NL)

PPDP 2017
Namur, Belgium
October 9, 2017

Roadmap

Reversibility: From Movies to Software Practice

ICT COST Action IC1405

This Work

Reversible computation in models of message-passing concurrency, in particular process calculi

Motivation:

- Rigorous basis for modern programming languages (Go, Erlang)
- Techniques based on type systems and contracts that enforce safety/liveness properties ("protocol conformance")
- Programming abstractions that "undo" computation steps and return to a previous consistent state
- Analysis of workflow management systems with backward and forward "jumps" at runtime

Causal consistency, Informally (Danos & Krivine, CONCUR'04): Reversibility doesn't lead to states not reachable with forward steps

Monitors-as-Memories Approach (JLAMP'17)

The **monitors** that verify protocol actions at runtime
used as
the **memories** needed to reverse communication steps

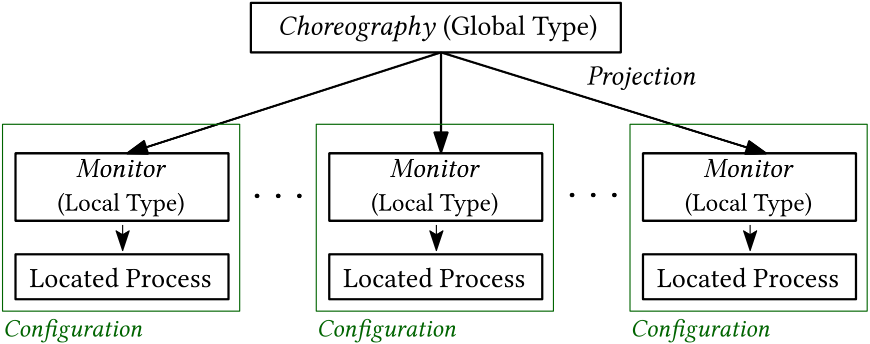
Smooth integration of reversibility into interacting processes:

+ A monitor for each protocol participant, with a session type that describes the intended protocol
+ A cursor in the type marks the protocol state: it moves forwards and backwards (reversing protocol actions)
+ Streamlined proofs of causal consistency

Shortcomings:

- Only protocols between two partners (binary session types)
- Synchronous communication

This Work: From Binary to Multiparty



Highlights:

- Asynchronous communication, declaratively specified
- Higher-order process passing (name passing is representable!)
- Causal consistency

Technical Contributions

1. A new **process model** for reversible, multiparty sessions
   - A concurrent λ-calculus with asynchronous process passing
   - Forward and backward semantics for decoupled rollbacks
   - Monitors-as-memories approach extended to global types, local types, and their process implementations
2. A proof of **causal consistency**
   - Needs an alternative semantics with **atomic rollbacks**, shown equivalent to the decoupled semantics
3. **Formal connection** of reversibility at two levels:
   - Declarative, given by global types
   - Operational, given by monitored processes

Technical Contributions

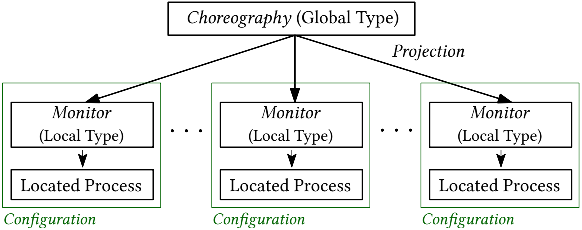
1. A new **process model** for reversible, multiparty sessions
   - A concurrent λ-calculus with asynchronous process passing
   - Forward and backward semantics for decoupled rollbacks
   - Monitors-as-memories approach extended to global types, local types, and their process implementations
2. A proof of **causal consistency**
   - Needs an alternative semantics with **atomic rollbacks**, shown equivalent to the decoupled semantics
3. **Formal connection** of reversibility at two levels:
   - Declarative, given by global types
   - Operational, given by monitored processes

In this talk

The process model and causal consistency, by example.

Roadmap

Governing Protocols: Global and Local Types



Global Types	G, G'	$::=$	$\mathtt{p} \to \mathtt{q} : \langle U \rangle.G \;\mid\; \mu X.G \;\mid\; X \;\mid\; \mathtt{end}$
Value Types	U, U'	$::=$	$\mathtt{bool} \;\mid\; \mathtt{nat} \;\mid\; \cdots \;\mid\; \boxed{T \to \diamond}$
Local Types	T, T'	$::=$	$\mathtt{p}!\langle U \rangle.T \;\mid\; \mathtt{p}?\langle U \rangle.T \;\mid\; \mu X.T \;\mid\; X \;\mid\; \mathtt{end}$

Notice:

- $T \to \diamond$ is the type of abstractions from names to processes
- $G\downarrow_{\mathtt{p}}$ denotes the projection of G onto participant $\mathtt{p}$ (standard)
- Labelled choices can be easily incorporated (see our arXiv TR)

Running Example: A Three-Buyer protocol

Alice ($\mathtt{A}$), Bob ($\mathtt{B}$), and Carol ($\mathtt{C}$) interact with a Vendor ($\mathtt{V}$):

$$
\begin{aligned}
G = \mathtt{A} \to \mathtt{V} : \langle \text{title} \rangle.\ \mathtt{V} \to \{\mathtt{A}, \mathtt{B}\} : \langle \text{price} \rangle. \\
\mathtt{A} \to \mathtt{B} : \langle \text{share} \rangle.\ \mathtt{B} \to \{\mathtt{A}, \mathtt{V}\} : \langle \text{OK} \rangle. \\
\mathtt{B} \to \mathtt{C} : \langle \text{share} \rangle.\ \mathtt{B} \to \mathtt{C} : \langle \{\{\diamond\}\} \rangle. \\
\mathtt{B} \to \mathtt{V} : \langle \text{address} \rangle.\ \mathtt{V} \to \mathtt{B} : \langle \text{date} \rangle.\mathtt{end}
\end{aligned}
$$

where $\{\{\diamond\}\}$ is the type of a **thunk process**:
Bob sends Carol some code with the protocol; she must activate it.

Running Example: A Three-Buyer protocol

Alice ($\mathtt{A}$), Bob ($\mathtt{B}$), and Carol ($\mathtt{C}$) interact with a Vendor ($\mathtt{V}$):

$$\begin{aligned} G = \mathtt{A} \to \mathtt{V} : \langle \text{title} \rangle.\ \mathtt{V} \to \{\mathtt{A}, \mathtt{B}\} : \langle \text{price} \rangle. \\ \mathtt{A} \to \mathtt{B} : \langle \text{share} \rangle.\ \mathtt{B} \to \{\mathtt{A}, \mathtt{V}\} : \langle \text{OK} \rangle. \\ \mathtt{B} \to \mathtt{C} : \langle \text{share} \rangle.\ \mathtt{B} \to \mathtt{C} : \langle \{\{\diamond\}\} \rangle. \\ \mathtt{B} \to \mathtt{V} : \langle \text{address} \rangle.\ \mathtt{V} \to \mathtt{B} : \langle \text{date} \rangle.\mathtt{end} \end{aligned}$$

where $\{\{\diamond\}\}$ is the type of a **thunk process**:
Bob sends Carol some code with the protocol; she must activate it.

Local types for the four participants (projections of G onto $\mathtt{V}, \mathtt{A}, \mathtt{B}, \mathtt{C}$):

$$G\downarrow_{\mathtt{V}} = \mathtt{A}?\langle \text{title} \rangle.\{\mathtt{A}, \mathtt{B}\}!\langle \text{price} \rangle.\mathtt{B}?\langle \text{OK} \rangle.\mathtt{B}?\langle \text{address} \rangle.\mathtt{B}!\langle \text{date} \rangle.\text{end}$$

$$G\downarrow_{\mathtt{A}} = \mathtt{V}!\langle \text{title} \rangle.\mathtt{V}?\langle \text{price} \rangle.\mathtt{B}!\langle \text{share} \rangle.\mathtt{B}?\langle \text{OK} \rangle.\text{end}$$

$$\begin{aligned} G\downarrow_{\mathtt{B}} = {} & \mathtt{V}?\langle \text{price} \rangle.\mathtt{A}?\langle \text{share} \rangle.\{\mathtt{A}, \mathtt{V}\}!\langle \text{OK} \rangle. \\ & \mathtt{C}!\langle \text{share} \rangle.\mathtt{C}!\langle \{\{\diamond\}\} \rangle.\mathtt{V}!\langle \text{address} \rangle.\mathtt{V}?\langle \text{date} \rangle.\text{end} \end{aligned}$$

$$G\downarrow_{\mathtt{C}} = \mathtt{B}?\langle \text{share} \rangle.\mathtt{B}?\langle \{\{\diamond\}\} \rangle.\text{end}$$

Key Ingredients

- **Processes** P, $Q, \ldots$ include point-to-point value communication, recursion, and application
- **Values** V, $W, \ldots$ include shared names and abstractions $\lambda x.\, P$. Name communication (delegation) can be represented.
- **Configurations** M, $N, \ldots$ are compositions of processes which are deployed in localities $\ell, \ell', \ldots$, one per participant $\mathtt{p}, \mathtt{q}, \ldots$

Key Ingredients

- **Processes** $P, Q, \ldots$ include point-to-point value communication, recursion, and application
- **Values** $V, W, \ldots$ include shared names and abstractions $\lambda x.\, P$. Name communication (delegation) can be represented.
- **Configurations** $M, N, \ldots$ are compositions of processes which are deployed in localities $\ell, \ell', \ldots$, one per participant $\mathrm{p}, \mathrm{q}, \ldots$
- There are also **run-time elements**, such as monitors and queues: they are configurations only generated at run-time
- Monitors with local types **with cursors** $\boxed{\wedge}$:

$$
\begin{array}{rcl}
T, T' & ::= & \mathrm{p}!\langle U\rangle.T \;\Big|\; \mathrm{p}?\langle U\rangle.T \;\Big|\; \mu X.T \;\Big|\; X \;\Big|\; \text{end} \quad \text{[as before]} \\
\alpha & ::= & \mathrm{q}?(U) \;\Big|\; \mathrm{q}!\langle U\rangle \\
H, K & ::= & \boxed{\wedge}\, T \;\Big|\; T\,\boxed{\wedge} \;\Big|\; \alpha_1.\cdots.\alpha_n.\boxed{\wedge}\, S
\end{array}
$$

(We need to add cursors also to global types; see the paper).

Syntax

$$
\begin{array}{rrcl}
\text{Names} & n, n' & ::= & a, b \mid s_{[\mathrm{p}]} \qquad u, w \; ::= \; n \mid x, y, z \\
\text{Values} & V, W & ::= & a, b \mid x, y, z \mid \lambda x.\, P \mid \mathtt{tt} \mid \mathtt{ff} \\
\text{Processes} & P, Q & ::= & u!\langle V\rangle.P \mid u?(x).P \mid V\; u \\
 & & \mid & P \mid Q \mid X \mid \mu X.P \mid (\nu\, n)P \mid \mathbf{0} \\
\text{Configurations} & M, N & ::= & \ell\{a!\langle x : T\rangle.P\} \mid \ell\{a?(x : T).P\} \\
 & & \mid & M \mid N \mid (\nu\, n)M \mid \mathbf{0} \\
 & & \mid & \boxed{\ell_{[\mathrm{p}]} : \wr P \int} \\
 & & \mid & \boxed{s_{[\mathrm{p}]} \lfloor H \cdot \widetilde{x} \cdot \sigma \rfloor^{\spadesuit}} \mid \boxed{k \lfloor (V\; u), \ell \rfloor} \\
 & & \mid & \boxed{s : (h_i \star h_o)} \\
\text{Queues} & h & ::= & \epsilon \mid h \circ (\mathrm{p}, \mathrm{q}, V)
\end{array}
$$

The tag ♠ can be ♦ or ♢ if the monitor is involved in a rollback or not

Running Example: Process Implementations

One process per protocol participant:

$$\begin{aligned}
\text{Vendor} &= d!\langle x : G\!\downarrow_{\mathtt{V}}\rangle.x?(t).x!\langle \mathit{price}(t)\rangle.x!\langle \mathit{price}(t)\rangle. \\
&\quad x?(ok).x?(a).x!\langle \mathit{date}\rangle.\mathbf{0} \\
\text{Alice} &= d?(y : G\!\downarrow_{\mathtt{A}}).y!\langle \text{`Logicomix'}\rangle.y?(p).y!\langle h\rangle.y?(ok).\mathbf{0} \\
\text{Bob} &= d?(z : G\!\downarrow_{\mathtt{B}}).z?(p).z?(h).z!\langle ok\rangle.z!\langle ok\rangle.z!\langle h\rangle. \\
&\qquad\qquad z!\Big\langle \{\!\!\{ z!\langle \text{`Lucca, 55100'}\rangle.z?(d).\mathbf{0} \}\!\!\} \Big\rangle.\mathbf{0} \\
\text{Carol} &= d?(w : G\!\downarrow_{\mathtt{C}}).w?(h).w?(\mathit{code}).(\mathit{code} *)
\end{aligned}$$

where $\{\!\!\{ P \}\!\!\} = \lambda x.\, P$, with $x \notin \mathit{fv}(P)$, is a **thunk process**. Upon activation of the thunk, received as code, Carol will send an address and receive a date on Bob's behalf.

A configuration results by placing these processes in four locations:

$$M = \ell_1\,\{\text{Vendor}\} \;\|\; \ell_2\,\{\text{Alice}\} \;\|\; \ell_3\,\{\text{Bob}\} \;\|\; \ell_4\,\{\text{Carol}\}$$

Roadmap

A Monitor-based Session Semantics

The binary case, as in [Kouzapas 2009; Hu et al, ECOOP'10]:

$$\overline{s}\langle v\rangle.P \mid s(x).Q$$

- Output and input processes along dual **endpoints** $\overline{s}$, s

A Monitor-based Session Semantics

The binary case, as in [Kouzapas 2009; Hu et al, ECOOP'10]:

$$\overline{s}\langle v\rangle.P \mid \bar{s}\lfloor !\,U\,.\,T_1 \cdot h_1\rfloor \mid s(x).Q \mid s\lfloor ?\,U\,.\,T_2 \cdot h_2\rfloor$$

- Output and input processes along dual **endpoints** $\overline{s}$, s
- A monitor per endpoint (a type and a message queue h_i)

A Monitor-based Session Semantics

The binary case, as in [Kouzapas 2009; Hu et al, ECOOP'10]:

$$\overline{s}\langle v\rangle.P \mid \bar{s}\lfloor !\,U\,.\,T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor ?\,U\,.\,T_2 \cdot h_2 \rfloor$$

$$\longrightarrow$$

$$P \mid \qquad \mid Q$$

- Output and input processes along dual **endpoints** $\overline{s}$, s
- A monitor per endpoint (a type and a message queue h_i)
- Types enable synchronizations;

A Monitor-based Session Semantics

The binary case, as in [Kouzapas 2009; Hu et al, ECOOP'10]:

$$\overline{s}\langle v\rangle.P \mid \overline{s}\lfloor !\,U\,.\,T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor ?\,U\,.\,T_2 \cdot h_2 \rfloor$$

$$\longrightarrow$$

$$P \mid \overline{s}\lfloor\, T_1 \cdot h_1 \rfloor \mid Q \mid s\lfloor\, T_2 \cdot h_2, v\, \rfloor$$

- Output and input processes along dual **endpoints** $\overline{s}$, s
- A monitor per endpoint (a type and a message queue h_i)
- Types enable synchronizations; processes/types are **consumed**

Our Idea: Cursors for Reversibility (JLAMP'17)

$$\overline{s}\langle v\rangle.P \mid \bar{s}\lfloor\ \ !U.T_1 \cdot h_1\rfloor \mid s(x).Q \mid s\lfloor\ \ ?U.T_2 \cdot h_2\rfloor$$

- Keep protocol information with a **cursor** on types

Our Idea: Cursors for Reversibility (JLAMP'17)

$$\overline{s}\langle v\rangle.P \mid \bar{s}\lfloor \boxed{\wedge}\,!U.T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor \boxed{\wedge}\,?U.T_2 \cdot h_2 \rfloor$$

- Keep protocol information with a **cursor** on types, written $\boxed{\wedge}$

Our Idea: Cursors for Reversibility (JLAMP'17)

$$\overline{s}\langle v\rangle.P \mid \bar{s}\lfloor \boxed{\wedge}\,!U.T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor \boxed{\wedge}\,?U.T_2 \cdot h_2 \rfloor$$

$\twoheadrightarrow$ [forward reduction]

$$P \mid \bar{s}\lfloor !U.\boxed{\wedge}\,T_1 \cdot h_1 \rfloor \mid Q \mid s\lfloor ?U.\boxed{\wedge}\,T_2 \cdot h_2, v \rfloor$$

- Keep protocol information with a **cursor** on types, written $\boxed{\wedge}$
- The monitor allows us to move forward

Our Idea: Cursors for Reversibility (JLAMP'17)

$$\overline{s}\langle v\rangle.P \mid \overline{s}\lfloor \boxed{\wedge}\, !U.T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor \boxed{\wedge}\, ?U.T_2 \cdot h_2 \rfloor$$

$\twoheadrightarrow$ [forward reduction]

$$P \mid \overline{s}\lfloor !U.\boxed{\wedge}\, T_1 \cdot h_1 \rfloor \mid Q \mid s\lfloor ?U.\boxed{\wedge}\, T_2 \cdot h_2, v \rfloor$$

$\rightsquigarrow$ [backwards reduction]

$$\overline{s}\langle v\rangle.P \mid \overline{s}\lfloor \boxed{\wedge}\, !U.T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor \boxed{\wedge}\, ?U.T_2 \cdot h_2 \rfloor$$

- Keep protocol information with a **cursor** on types, written $\boxed{\wedge}$
- The monitor allows us to move forward and backwards

Our Idea: Cursors for Reversibility (JLAMP'17)

$$\overline{s}\langle v\rangle.P \mid \overline{s}\lfloor \boxed{\wedge}\, !U.T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor \boxed{\wedge}\, ?U.T_2 \cdot h_2 \rfloor$$

$\twoheadrightarrow$ [forward reduction]

$$P \mid \overline{s}\lfloor !U.\boxed{\wedge}\, T_1 \cdot h_1 \rfloor \mid Q \mid s\lfloor ?U.\boxed{\wedge}\, T_2 \cdot h_2, v \rfloor$$

$\rightsquigarrow$ [backwards reduction]

$$\overline{s}\langle v\rangle.P \mid \overline{s}\lfloor \boxed{\wedge}\, !U.T_1 \cdot h_1 \rfloor \mid s(x).Q \mid s\lfloor \boxed{\wedge}\, ?U.T_2 \cdot h_2 \rfloor$$

- Keep protocol information with a **cursor** on types, written $\boxed{\wedge}$
- The monitor allows us to move forward and backwards
- Two operational semantics, denoted $\twoheadrightarrow$ and $\rightsquigarrow$

Forward Semantics ($\twoheadrightarrow$): Key Ideas

Our proposal for multiparty communication — in five rules:

- Session initiation checks that distributed participants implement compatible protocols, and sets up all the run-time machinery
- Session communication is asynchronous, mediated by the queue, in two reduction steps
  The types in the monitors have to enable these steps
- β-reduction requires a special memory ("running function") to record the exact part of the process where it occurs
- Parallel composition is supporteds by splitting running processes

Forward Reduction (1/5)

(INIT)

$$\frac{\mathrm{pa}(G) = \{\mathrm{p}_1, \cdots, \mathrm{p}_n\} \qquad \forall \mathrm{p}_i \in \mathrm{pa}(G).\ G\downarrow_{\mathrm{p}_i} = T_i}{\ell_1\,\{a!\langle x_1 : T_1\rangle.P_1\} \mid \prod_{i\in\{2,\cdots,n\}} \ell_i\,\{a?(x_i : T_i).P_i\}}$$

$$\twoheadrightarrow$$

$$(\nu\, s)(\prod_{i\in\{1,\cdots,n\}} \ell_{i[\mathrm{p}_i]} : \wr P_i\{^{s[\mathrm{p}_i]}/x_i\} \int \mid s_{[\mathrm{p}_i]} \lfloor \boxed{\wedge}\, T_i \cdot x_i \cdot [x_i \mapsto a] \rfloor^{\diamondsuit}$$

$$\mid s : (\epsilon \star \epsilon))$$

- One request (at ℓ_1) and $n-1$ accepts (at $\ell_2, \ldots, \ell_n$)
- Reduction sets up the n-party session: fresh session names, running processes, monitors (with tag $\diamondsuit$), and the empty queue
- Cursors within the monitors are placed at the beginning

Forward Reduction (2/5)

$$\text{(OUT)}\ \frac{\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)}{\begin{array}{c}\ell_{[\mathtt{r}]} : \wr s_{[\mathtt{p}]}!\langle V\rangle.P \int \ \mid s_{[\mathtt{p}]}\lfloor \mathbb{T}\left[\boxed{\wedge}\, \mathtt{q}!\langle U\rangle.S\right] \cdot \tilde{x} \cdot \sigma \rfloor \\ \mid s : (h_i \star h_o) \\ \twoheadrightarrow \\ \ell_{[\mathtt{r}]} : \wr P \int \ \mid s_{[\mathtt{p}]}\lfloor \mathbb{T}\left[\mathtt{q}!\langle U\rangle.\boxed{\wedge}\, S\right] \cdot \tilde{x} \cdot \sigma \rfloor \\ \mid s : (h_i \star h_o \circ (\mathtt{p}\,,\, \mathtt{q}\,,\, \sigma(V)))\end{array}}$$

- $\mathbb{T}$ is a type context (with past protocol actions)
- Premise $\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)$ allows performing actions on names previously received via abstraction-passing.

Forward Reduction (3/5)

$$(\textsc{In})\ \frac{\mathtt{p} = \mathtt{r} \ \vee\ \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)}{\begin{array}{c}\ell_{[\mathtt{r}]} : \wr s_{[\mathtt{p}]}?(y).P \int \ \mid s_{[\mathtt{p}]} \lfloor \mathbb{T}\left[\boxed{\wedge}\, \mathtt{q}?\langle U \rangle.S\right] \cdot \tilde{x} \cdot \sigma \rfloor \\ \mid s : (h_i \star (\mathtt{q}\,,\, \mathtt{p}\,,\ V) \circ h_o) \\ \twoheadrightarrow \\ \ell_{[\mathtt{r}]} : \wr P \int \ \mid s_{[\mathtt{p}]} \lfloor \mathbb{T}\left[\mathtt{q}?\langle U \rangle.\boxed{\wedge}\, S\right] \cdot \tilde{x}, y \cdot \sigma[y \mapsto\ V] \rfloor \\ \mid s : (h_i \circ (\mathtt{q}\,,\, \mathtt{p}\,,\ V) \star h_o)\end{array}}$$

- The queue actually implements a history, separated by $\star$
- $\mathbb{T}$ is a type context (with past protocol actions)
- Premise $\mathtt{p} = \mathtt{r} \ \vee\ \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)$ allows performing actions on names previously received via abstraction-passing.

Forward Reduction (4/5)

(BETA)

$$\frac{\sigma(V) = \lambda x.\, P}{\ell_{[\mathrm{p}]} : \wr (V\ w) \int \mid s_{[\mathrm{p}]} \lfloor \mathbb{T}\left[\boxed{\wedge}\, S\right] \cdot \tilde{x} \cdot \sigma \rfloor \twoheadrightarrow (\nu\, k)\ \left(\ell_{[\mathrm{p}]} : \wr P\{^{\sigma(w)}/_{x}\} \int \mid k \lfloor (V\ w),\, \ell \rfloor \mid s_{[\mathrm{p}]} \lfloor \mathbb{T}\left[k.\boxed{\wedge}\, S\right] \cdot \tilde{x} \cdot \sigma \rfloor\right)}$$

- A fresh k is used in the running function and the monitor.

Forward Reduction (5/5)

$$(\textsc{Spawn})\ \frac{}{\begin{array}{c}\ell_{[\mathrm{p}]} : \wr P \mid Q \int \ \mid s_{[\mathrm{p}]} \lfloor \mathbb{T}\left[\boxed{\wedge}\, S\right] \cdot \tilde{x} \cdot \sigma \rfloor \\ \twoheadrightarrow \\ (\nu\, \ell_1, \ell_2)\, (\ell_{[\mathrm{p}]} : \wr \mathbf{0} \int \ \mid \ell_{1[\mathrm{p}]} : \wr P \int \ \mid \ell_{2[\mathrm{p}]} : \wr Q \int \\ \mid s_{[\mathrm{p}]} \lfloor \mathbb{T}\left[(\ell, \ell_1, \ell_2).\, \boxed{\wedge}\, S\right] \cdot \tilde{x} \cdot \sigma \rfloor)\end{array}}$$

- Location ℓ is split into running processes with fresh ℓ_1, ℓ_2. This is recorded in the monitor

Backward Semantics ($\rightsquigarrow$): Key Ideas

- The rollback of a synchronization is **decoupled**:
  The two involved monitors are first jointly tagged (from $\Diamond$ to $\blacklozenge$); then, each participant independently undoes its behavior
- Undoing a forward session communication uses two backward reductions
- Again, tagging and undoing steps have to be enabled by the type
- In contrast, β-reductions and parallel processes are rollbacked atomically

Backward Semantics (1/2)

$$(\textsc{RollS})\ \frac{}{\begin{array}{c} s_{[\mathrm{p}]}\lfloor \mathbb{T}\left[\mathrm{q}?\langle U\rangle.\boxed{\wedge}\ T\right] \cdot \tilde{x} \cdot \sigma_1 \rfloor^{\Diamond} \\ \mid s_{[\mathrm{q}]}\lfloor \mathbb{S}\left[\mathrm{p}!\langle U\rangle.\boxed{\wedge}\ S\right] \cdot \tilde{y} \cdot \sigma_2 \rfloor^{\Diamond} \\ \mid s : (h_i \star h_o) \\ \rightsquigarrow \\ s_{[\mathrm{p}]}\lfloor \mathbb{T}\left[\mathrm{q}?\langle U\rangle.\boxed{\wedge}\ T\right] \cdot \tilde{x} \cdot \sigma_1 \rfloor^{\blacklozenge} \\ \mid s_{[\mathrm{q}]}\lfloor \mathbb{S}\left[\mathrm{p}!\langle U\rangle.\boxed{\wedge}\ S\right] \cdot \tilde{y} \cdot \sigma_2 \rfloor^{\blacklozenge} \\ \mid s : (h_i \star h_o) \end{array}}$$

- Starts to undo a synchronization between p and q by **tagging** their monitors
- The two monitor types must be complementary
- Once tagged, reversing input/output actions can occur independently

Backward Semantics (2/2)

$$(\text{ROUT})\ \frac{\mathrm{p} = \mathrm{r} \vee \mathrm{p} \in \mathbf{roles}(\mathrm{r}, h_i)}{\begin{array}{c}\ell_{[\mathrm{r}]} : \wr P \int \ \mid\ s_{[\mathrm{p}]} \lfloor \mathbb{T}\left[\mathrm{q}?\langle U\rangle.\boxed{\wedge} S\right] \cdot \tilde{x}, y \cdot \sigma \rfloor^{\blacklozenge} \mid \\ s : (h_i \circ (\mathrm{q}, \mathrm{p}, V) \star h_o) \\ \rightsquigarrow \\ \ell_{[\mathrm{r}]} : \wr s_{[\mathrm{p}]}?(y).P \int \ \mid\ s_{[\mathrm{p}]} \lfloor \mathbb{T}[\wedge \mathrm{q}?\langle U\rangle.S] \cdot \tilde{x} \cdot \sigma \setminus y \rfloor^{\lozenge} \mid \\ s : (h_i \star (\mathrm{q}, \mathrm{p}, V) \circ h_o)\end{array}}$$

- Rule RIN is symmetric to Rule IN and only enabled when the monitor is tagged as $\blacklozenge$
- Other rules for $\rightsquigarrow$ are as expected

Running Example: The Semantics At Work

Recall the configuration:

$$M = \ell_1 \{\text{Vendor}\} \mid \ell_2 \{\text{Alice}\} \mid \ell_3 \{\text{Bob}\} \mid \ell_4 \{\text{Carol}\}$$

where processes are as follows:

$$\begin{aligned}
\text{Vendor} &= d!\langle x : G\downarrow_{\mathtt{V}}\rangle.x?(t).x!\langle \mathit{price}(t)\rangle.x!\langle \mathit{price}(t)\rangle. \\
&\quad x?(ok).x?(a).x!\langle \mathit{date}\rangle.\mathbf{0} \\
\text{Alice} &= d?(y : G\downarrow_{\mathtt{A}}).y!\langle \text{`Logicomix'}\rangle.y?(p).y!\langle h\rangle.y?(ok).\mathbf{0} \\
\text{Bob} &= d?(z : G\downarrow_{\mathtt{B}}).z?(p).z?(h).z!\langle ok\rangle.z!\langle ok\rangle.z!\langle h\rangle. \\
&\qquad z!\left\langle \{\!\!\{ z!\langle \text{`Lucca, 55100'}\rangle.z?(d).\mathbf{0} \}\!\!\} \right\rangle.\mathbf{0} \\
\text{Carol} &= d?(w : G\downarrow_{\mathtt{C}}).w?(h).w?(\mathit{code}).(\mathit{code} *)
\end{aligned}$$

Let's examine some forward and backward reductions from M.

Forward Semantics: Session Initialization

$$M = \ell_1 \{\text{Vendor}\} \mid \ell_2 \{\text{Alice}\} \mid \ell_3 \{\text{Bob}\} \mid \ell_4 \{\text{Carol}\}$$

$$\begin{aligned} M \twoheadrightarrow (\nu\, s)\Big(& \ell_{1[\mathtt{V}]} : \wr V_1\{^{s[\mathtt{V}]}/x\} \int \mid s_{[\mathtt{V}]} \lfloor \boxed{\wedge}\, G\downarrow_{\mathtt{V}} \cdot x \cdot [x \mapsto d] \rfloor^{\Diamond} \\ & \mid \ell_{2[\mathtt{A}]} : \wr A_1\{^{s[\mathtt{A}]}/y\} \int \mid s_{[\mathtt{A}]} \lfloor \boxed{\wedge}\, G\downarrow_{\mathtt{A}} \cdot y \cdot [y \mapsto d] \rfloor^{\Diamond} \\ & \mid \ell_{3[\mathtt{B}]} : \wr B_1\{^{s[\mathtt{B}]}/z\} \int \mid s_{[\mathtt{B}]} \lfloor \boxed{\wedge}\, G\downarrow_{\mathtt{B}} \cdot z \cdot [z \mapsto d] \rfloor^{\Diamond} \\ & \mid \ell_{4[\mathtt{C}]} : \wr C_1\{^{s[\mathtt{C}]}/w\} \int \mid s_{[\mathtt{C}]} \lfloor \boxed{\wedge}\, G\downarrow_{\mathtt{C}} \cdot w \cdot [w \mapsto d] \rfloor^{\Diamond} \\ & \mid s : (\epsilon \star \epsilon) \Big) = M_1 \end{aligned}$$

- Each monitor type is initialized with $\boxed{\wedge}$
- A queue with empty memory is created
- At this point we could either undo the initialization, or proceed further with the protocol

Forward Semantics: Asynchronous Output

The first action of Alice is to send ‘Logicomix’ to Vendor:

$$M_1 \twoheadrightarrow (\nu\ s)(\ \ell_{2[\mathtt{A}]} : \wr s_{[\mathtt{A}]}?(p).s_{[\mathtt{A}]}!\langle h\rangle.s_{[\mathtt{A}]}?(ok).\mathbf{0}\int$$

$$\mid s_{[\mathtt{A}]}\lfloor \mathtt{V}!\langle \text{title}\rangle.\boxed{\wedge}\,\mathtt{V}?\langle \text{price}\rangle.\mathtt{B}!\langle \text{share}\rangle.\mathtt{B}?\langle \text{OK}\rangle.\text{end} \cdot y \cdot [y \mapsto d]\rfloor^{\Diamond}$$

$$\mid N_2 \mid s : (\epsilon \star (\mathtt{A}, \mathtt{V}, \text{‘Logicomix’}))) = M_2$$

$\twoheadrightarrow$

Forward Semantics: Asynchronous Output

The first action of Alice is to send 'Logicomix' to Vendor:

$$M_1 \twoheadrightarrow (\nu\, s)(\, \ell_{2[\mathtt{A}]} : \wr s_{[\mathtt{A}]}?(p).s_{[\mathtt{A}]}!\langle h\rangle.s_{[\mathtt{A}]}?(ok).\mathbf{0} \int$$
$$\mid s_{[\mathtt{A}]}\lfloor \mathtt{V}!\langle \text{title}\rangle.\, \boxed{\wedge}\, \mathtt{V}?\langle \text{price}\rangle.\mathtt{B}!\langle \text{share}\rangle.\mathtt{B}?\langle \text{OK}\rangle.\text{end} \,\cdot\, y \,\cdot\, [y \mapsto d]\rfloor^{\lozenge}$$
$$\mid N_2 \mid s : (\epsilon \star (\mathtt{A}\,,\, \mathtt{V}\,,\, \text{`Logicomix'})\,)\,) = M_2$$

$$\twoheadrightarrow$$

$$(\nu\, s)(\, \ell_{1[\mathtt{V}]} : \wr s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}?(ok).$$
$$s_{[\mathtt{V}]}?(a).s_{[\mathtt{V}]}!\langle date\rangle.\mathbf{0} \int$$
$$\mid s_{[\mathtt{V}]}\lfloor \mathtt{A}?\langle \text{title}\rangle.\, \boxed{\wedge}\, \{\mathtt{A},\mathtt{B}\}!\langle \text{price}\rangle.\, T_{\mathtt{V}} \,\cdot\, x, t \,\cdot\, \sigma_3 \rfloor^{\lozenge} \mid N_3$$
$$\mid s : ((\mathtt{A}\,,\, \mathtt{V}\,,\, \text{`Logicomix'}) \star \epsilon)\,)\,) = M_3$$

Forward Semantics: Asynchronous Output

The first action of Alice is to send ‘Logicomix’ to Vendor:

$$M_1 \twoheadrightarrow (\nu\, s)(\, \ell_{2[\mathtt{A}]} : \wr s_{[\mathtt{A}]}?(p).s_{[\mathtt{A}]}!\langle h\rangle.s_{[\mathtt{A}]}?(ok).\mathbf{0} \int$$
$$\mid s_{[\mathtt{A}]} \lfloor \mathtt{V}!\langle \text{title}\rangle.\boxed{\wedge}\,\mathtt{V}?\langle \text{price}\rangle.\mathtt{B}!\langle \text{share}\rangle.\mathtt{B}?\langle \text{OK}\rangle.\text{end} \cdot y \cdot [y \mapsto d] \rfloor^{\Diamond}$$
$$\mid N_2 \mid s : (\epsilon \star (\mathtt{A}\,,\, \mathtt{V}\,,\, \text{‘Logicomix’}))\,) = M_2$$

$$\twoheadrightarrow$$

$$(\nu\, s)(\, \ell_{1[\mathtt{V}]} : \wr s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}?(ok).$$
$$s_{[\mathtt{V}]}?(a).s_{[\mathtt{V}]}!\langle date\rangle.\mathbf{0} \int$$
$$\mid s_{[\mathtt{V}]} \lfloor \mathtt{A}?\langle \text{title}\rangle.\boxed{\wedge}\,\{\mathtt{A},\mathtt{B}\}!\langle \text{price}\rangle.T_{\mathtt{V}} \cdot x, t \cdot \sigma_3 \rfloor^{\Diamond} \mid N_3$$
$$\mid s : ((\mathtt{A}\,,\, \mathtt{V}\,,\, \text{‘Logicomix’}) \star \epsilon)\,) = M_3$$

In M_3 we have:

- $\sigma_3 = [x \mapsto d], [t \mapsto \text{‘Logicomix’}]$ is the resulting store
- $T_{\mathtt{V}} = \mathtt{B}?\langle \text{OK}\rangle.\mathtt{B}?\langle \text{address}\rangle.\mathtt{B}!\langle \text{date}\rangle.\text{end}$
- the message from $\mathtt{A}$ to $\mathtt{V}$ has now been moved to the input queue

Decoupled Rollback (1/3)

Returning to M_1 starting from M_3.
We need to apply Rules (RollS), (RIn), and (ROut).

$$
\begin{aligned}
M_3 \rightsquigarrow (\nu\, s)(\, & \ell_{1[\mathtt{V}]} : \wr s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}?(ok). \\
& \quad s_{[\mathtt{V}]}?(a).s_{[\mathtt{V}]}!\langle date\rangle.\mathbf{0} \int \\
& \mid s_{[\mathtt{V}]} \lfloor \mathtt{A}?\langle \text{title}\rangle.\boxed{\wedge}\{\mathtt{A},\mathtt{B}\}!\langle \text{price}\rangle.T_{\mathsf{B}} \cdot x, t \cdot \sigma_3 \rfloor^{\blacklozenge} \\
& \mid \ell_{2[\mathtt{A}]} : \wr s_{[\mathtt{A}]}?(p).s_{[\mathtt{A}]}!\langle h\rangle.s_{[\mathtt{A}]}?(ok).\mathbf{0} \int \\
& \mid s_{[\mathtt{A}]} \lfloor \mathbb{T}_4 \left[\boxed{\wedge}\, \mathtt{V}?\langle \text{price}\rangle.\mathtt{B}!\langle \text{share}\rangle.\mathtt{B}?\langle \text{OK}\rangle.\text{end} \right] \cdot y \cdot [y \mapsto d] \rfloor^{\blacklozenge} \\
& \mid N_4 \mid s : ((\mathtt{A}\,,\, \mathtt{V}\,,\, \text{‘Logicomix’}) \star \epsilon)\,) = M_4
\end{aligned}
$$

Notice:

- By applying (RollS) monitors for V and A have now tag ♦
- In the monitor for A, we have $\mathbb{T}_4\,[\bullet] = \mathtt{V}!\langle \text{title}\rangle.\bullet$
- M_4 has several possible forward and backward reductions.

Decoupled Rollback (2/3)

Using Rule (RIN) to first undo the input at $\mathtt{V}$:

$$
\begin{aligned}
M_4 \rightsquigarrow (\nu\, s)(\, & \ell_{1[\mathtt{V}]} : \wr s_{[\mathtt{V}]}?(t).s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}!\langle price(t)\rangle. \\
& \quad s_{[\mathtt{V}]}?(ok).s_{[\mathtt{V}]}?(a).s_{[\mathtt{V}]}!\langle date\rangle.\mathbf{0}\int \\
& \mid s_{[\mathtt{V}]}\lfloor \boxed{\wedge}\, \mathtt{A}?\langle \mathsf{title}\rangle.\{\mathtt{A},\mathtt{B}\}!\langle \mathsf{price}\rangle.\, T_{\mathsf{B}} \cdot x \cdot [x \mapsto d]\rfloor^{\diamondsuit} \\
& \mid \ell_{2[\mathtt{A}]} : \wr s_{[\mathtt{A}]}?(p).s_{[\mathtt{A}]}!\langle h\rangle.s_{[\mathtt{A}]}?(ok).\mathbf{0}\int \\
& \mid s_{[\mathtt{A}]}\lfloor \mathbb{T}_4\left[\boxed{\wedge}\, \mathtt{V}?\langle \mathsf{price}\rangle.\mathtt{B}!\langle \mathsf{share}\rangle.\mathtt{B}?\langle \mathsf{OK}\rangle.\mathsf{end}\right] \cdot y \cdot [y \mapsto d]\rfloor^{\blacklozenge} \\
& \mid N_4 \mid s : (\epsilon \star (\mathtt{A}\,,\ \mathtt{V}\,,\ \text{‘Logicomix’}))\,) = M_5
\end{aligned}
$$

- The input at $\mathtt{V}$ has been undone, as witnessed by the modified cursor and tag $\diamondsuit$
- The output at $\mathtt{A}$ still needs to be reversed (hence the tag $\blacklozenge$); this can take place from M_5 at any time

Decoupled Rollback (3/3)

A particular reduction from M_5 undoes the output at $\mathtt{A}$:

$$
\begin{aligned}
M_5 \rightsquigarrow (\nu\, s)(\, &\ell_{1[\mathtt{V}]} : \wr s_{[\mathtt{V}]}?(t).s_{[\mathtt{V}]}!\langle price(t)\rangle.s_{[\mathtt{V}]}!\langle price(t)\rangle. \\
&\quad s_{[\mathtt{V}]}?(ok).s_{[\mathtt{V}]}?(a).s_{[\mathtt{V}]}!\langle date\rangle.\mathbf{0}\wr \\
&\mid s_{[\mathtt{V}]} \llfloor \boxed{\wedge}\, \mathtt{A}?\langle \text{title}\rangle.\{\mathtt{A},\mathtt{B}\}!\langle\text{price}\rangle.T_{\mathsf{B}} \cdot x \cdot [x \mapsto d] \rrfloor^{\lozenge} \\
&\mid \ell_{2[\mathtt{A}]} : \wr s_{[\mathtt{A}]}!\langle\text{‘Logicomix’}\rangle.s_{[\mathtt{A}]}?(p).s_{[\mathtt{A}]}!\langle h\rangle.s_{[\mathtt{A}]}?(ok).\mathbf{0}\wr \\
&\mid s_{[\mathtt{A}]} \llfloor \boxed{\wedge}\, \mathtt{V}!\langle\text{title}\rangle.\mathtt{V}?\langle\text{price}\rangle.\mathtt{B}!\langle\text{share}\rangle.\mathtt{B}?\langle\text{OK}\rangle.\text{end} \cdot y \cdot [y \mapsto d] \rrfloor^{\lozenge} \\
&\mid N_4 \mid s : (\epsilon \star \epsilon)) = M_6
\end{aligned}
$$

- Clearly, $M_6 = M_1$.
- Summing up, the synchronization realized by the sequence $M_1 \twoheadrightarrow M_2 \twoheadrightarrow M_3$ can be reversed by the sequence $M_3 \rightsquigarrow M_4 \rightsquigarrow M_5 \rightsquigarrow M_6$

Forward Abstraction Passing (1/3)

Assume that M_3 follows a sequence of forward reductions until M_7:

$$M_7 = (\nu\, s)(\, \ell_{3_{[\mathtt{B}]}} : \wr s_{[\mathtt{B}]}!\Big\langle \{\!\!\{ s_{[\mathtt{B}]}!\langle \text{‘Lucca, 55100’}\rangle.s_{[\mathtt{B}]}?(d).\mathbf{0} \}\!\!\} \Big\rangle.\mathbf{0} \int$$

$$\mid s_{[\mathtt{B}]} \lfloor \mathbb{T}_7 \Big[\boxed{\wedge}\, \mathtt{C}!\langle\{\{\diamond\}\}\rangle.\mathtt{V}!\langle\text{address}\rangle.\mathtt{V}?\langle\text{date}\rangle.\text{end} \Big] \cdot z, p, h \cdot \sigma_7 \rfloor^{\Diamond}$$

$$\mid \ell_{4_{[\mathtt{C}]}} : \wr s_{[\mathtt{C}]}?(code).(code *) \int$$

$$\mid s_{[\mathtt{C}]} \lfloor \mathbb{T}_8 \Big[\boxed{\wedge}\, \mathtt{B}?\langle\{\{\diamond\}\}\rangle.\text{end} \Big] \cdot w, h \cdot \sigma_8 \rfloor^{\Diamond} \mid N_5 \mid s : (h_7 \star \epsilon))$$

Forward Abstraction Passing (1/3)

Assume that M_3 follows a sequence of forward reductions until M_7:

$$M_7 = (\nu\, s)(\, \ell_{3[\mathtt{B}]} : \wr s_{[\mathtt{B}]}!\Big\langle \{\!\{ s_{[\mathtt{B}]}!\langle\text{‘Lucca, 55100’}\rangle.s_{[\mathtt{B}]}?(d).\mathbf{0} \}\!\} \Big\rangle.\mathbf{0} \int$$

$$\mid s_{[\mathtt{B}]} \lfloor \mathbb{T}_7 \left[\boxed{\wedge}\, \mathtt{C}!\langle\{\{\diamond\}\}\rangle.\mathtt{V}!\langle\text{address}\rangle.\mathtt{V}?\langle\text{date}\rangle.\text{end} \right] \cdot z, p, h \cdot \sigma_7 \rfloor^{\Diamond}$$

$$\mid \ell_{4[\mathtt{C}]} : \wr s_{[\mathtt{C}]}?(code).(code *) \int$$

$$\mid s_{[\mathtt{C}]} \lfloor \mathbb{T}_8 \left[\boxed{\wedge}\, \mathtt{B}?\langle\{\{\diamond\}\}\rangle.\text{end} \right] \cdot w, h \cdot \sigma_8 \rfloor^{\Diamond} \mid N_5 \mid s : (h_7 \star \epsilon))$$

where $\mathbb{T}_7\,[\bullet]$, σ_7, $\mathbb{T}_8\,[\bullet]$, σ_8, and h_7 capture prior steps as follows:

$$\mathbb{T}_7\,[\bullet] = \mathtt{V}?\langle\text{price}\rangle.\mathtt{A}?\langle\text{share}\rangle.\{\mathtt{A}, \mathtt{V}\}!\langle\text{OK}\rangle.\mathtt{C}!\langle\text{share}\rangle.\bullet$$

$$\sigma_7 = [z \mapsto d], [p \mapsto price(\text{‘Logicomix’})], [h \mapsto 120]$$

$$\mathbb{T}_8\,[\bullet] = \mathtt{B}?\langle\text{share}\rangle.\bullet \qquad \sigma_8 = [w \mapsto d], [h \mapsto 120]$$

$$\begin{aligned} h_7 = {} & (\mathtt{A}, \mathtt{V}, \text{‘Logicomix’}) \\ & \circ (\mathtt{V}, \mathtt{A}, price(\text{‘Logicomix’})) \circ (\mathtt{V}, \mathtt{B}, price(\text{‘Logicomix’})) \\ & \circ (\mathtt{A}, \mathtt{B}, 120) \circ (\mathtt{B}, \mathtt{A}, \text{‘ok’}) \circ (\mathtt{B}, \mathtt{V}, \text{‘ok’}) \circ (\mathtt{B}, \mathtt{C}, 120) \end{aligned}$$

Forward Abstraction Passing (2/3)

If $M_7 \twoheadrightarrow \twoheadrightarrow M_8$ by using Rules (OUT) and (IN), then we would have a higher-order communication:

$$
\begin{aligned}
M_8 = \ & (\nu\, s)(\, \ell_{3[\mathtt{B}]} : \wr \mathbf{0} \int \\
& \mid s_{[\mathtt{B}]} \lfloor \mathbb{T}_7 \left[\mathtt{C}!\langle \{\{\diamond\}\} \rangle . \boxed{\wedge}\, \mathtt{V}!\langle \text{address} \rangle . \mathtt{V}?\langle \text{date} \rangle . \text{end} \right] \cdot z, p, h \cdot \sigma_7 \rfloor^{\lozenge} \\
& \mid \ell_{4[\mathtt{C}]} : \wr (\mathit{code} \, *) \int \\
& \mid s_{[\mathtt{C}]} \lfloor \mathbb{T}_8 \left[\mathtt{B}?\langle \{\{\diamond\}\} \rangle . \boxed{\wedge}\, \text{end} \right] \cdot w, h, \mathit{code} \cdot \sigma_9 \rfloor^{\lozenge} \\
& \mid N_5 \mid s : (h_7 \circ (\mathtt{B}\,,\ \mathtt{C}\,,\ \{\!\!\{ s_{[\mathtt{B}]}!\langle \text{`Lucca, 55100'} \rangle . s_{[\mathtt{B}]}?(d).\mathbf{0} \}\!\!\}) \star \epsilon))
\end{aligned}
$$

where $\sigma_9 = \sigma_8[\mathit{code} \mapsto \{\!\!\{ s_{[\mathtt{B}]}!\langle \text{`Lucca, 55100'} \rangle . s_{[\mathtt{B}]}?(d).\mathbf{0} \}\!\!\}]$.

Forward Abstraction Passing (3/3)

We may apply Rule (BETA) to obtain the code sent from $\mathtt{B}$ to $\mathtt{C}$:

$$M_8 \twoheadrightarrow (\nu\, s)(\nu\, k)(\, \ell_{4[\mathtt{C}]} : \wr s_{[\mathtt{B}]}!\langle\text{‘Lucca, 55100’}\rangle.s_{[\mathtt{B}]}?(d).\mathbf{0} \int \mid N_6$$
$$\mid s_{[\mathtt{B}]}\lfloor \mathbb{T}_7 \left[\mathtt{C}!\langle\{\{\diamond\}\}\rangle.\boxed{\wedge}\,\mathtt{V}!\langle\text{address}\rangle.\mathtt{V}?\langle\text{date}\rangle.\text{end}\right] \cdot z, p, h \cdot \sigma_7 \rfloor^{\Diamond}$$
$$\mid k\lfloor(code\, *)\,,\, \ell_4\rfloor \mid s_{[\mathtt{C}]}\lfloor \mathbb{T}_8 \left[\mathtt{B}?\langle\{\{\diamond\}\}\rangle.k.\,\boxed{\wedge}\,\text{end}\right] \cdot w, h, code \cdot \sigma_9 \rfloor^{\Diamond}$$
$$\mid s : (h_7 \circ (\mathtt{B}\,,\, \mathtt{C}\,,\, \{\!\!\{ s_{[\mathtt{B}]}!\langle\text{‘Lucca, 55100’}\rangle.s_{[\mathtt{B}]}?(d).\mathbf{0} \}\!\!\}) \star \epsilon)\,) = M_9$$

- This reduction added a running function on a fresh k, which is also used within the monitor $s_{[\mathtt{C}]}$.
- Reduction $M_8 \twoheadrightarrow M_9$ completes the code mobility from $\mathtt{B}$ to $\mathtt{C}$: the now active thunk will run $\mathtt{B}$'s implementation from $\mathtt{C}$'s location.
- Observe that Bob's identity $\mathtt{B}$ is “hardwired” in the sent thunk. This justifies the premise $\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)$.
- Reductions from M_9 will modify the cursor in the type stored in monitor $s_{[\mathtt{B}]}$ based on the process located at $\ell_{4[\mathtt{C}]}$.

Roadmap

Causal Consistency (1/3)

Intuitively, causal consistency characterizes rollbacks which are:

- **Consistent**: do not lead to past unreachable configurations
- **Flexible**: admit the rearrangement of reversed actions

Thus, the set of states reached by a backward step could have been reached by performing only forward computations.

Causal Consistency (1/3)

Intuitively, causal consistency characterizes rollbacks which are:

- **Consistent**: do not lead to past unreachable configurations
- **Flexible**: admit the rearrangement of reversed actions

Thus, the set of states reached by a backward step could have been reached by performing only forward computations.

Causal consistency is a property of **traces of transitions** between configurations. **Causal equivalence** $\asymp$ ensures:

- given two *concurrent* transitions, the traces obtained by swapping their execution order are equivalent
- a trace consisting of opposing transitions is equivalent to the empty trace

Causal Consistency (2/3)

The flexibility of the decoupled semantics makes proving **causal consistency** difficult

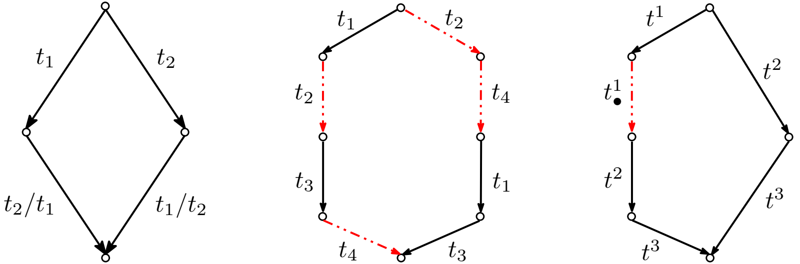
- We move to a synchronous semantics with atomic rollbacks and communications (“atomic semantics”)
- The decoupled and atomic semantics are tightly related via
  (1) a bi-directional operational correspondence and
  (2) a back-and-forth barbed bisimilarity
- It then suffices to prove causal consistency on the atomic semantics!

Causal Consistency (3/3)

Theorem (Causal consistency)

Let ρ_1 and ρ_2 be coinitial traces of transitions.
Then $\rho_1 \asymp \rho_2$ if and only if ρ_1 and ρ_2 are cofinal.

We follow the "recipe" by Danos & Krivine, using three lemmas:



(a) Square Lemma (b) Rearranging Lemma (c) Shortening Lemma

Black, solid arrows represent forward reductions;
red, dashed arrows represent backward reductions.

Roadmap

Final Remarks

A process framework of reversible, multiparty asynchronous communication

- built upon session-based concurrency
- flexible decoupled rollbacks
- causally consistent

Future work

- Add control to reversibility, via enhanced (monitor) types
  - types with modalities
  - types with logical conditions

  Different monitors for the same process enact different behaviors
- Compare with recent work on “Concurrent Reversible Sessions” by Castellani, Dezani-Ciancaglini & Giannini (CONCUR’17)
- Implement practical support for process specifications with reversibility (building upon CaReDeb)

Causally Consistent Reversible Choreographies

A Monitors-as-Memories Approach

Claudio Antares Mezzina (IMT Lucca, IT)
Jorge A. Pérez (University of Groningen, NL)

PPDP 2017
Namur, Belgium
October 9, 2017