Causally Consistent Reversible Choreographies*

Claudio Antares Mezzina
IMT School for Advanced Studies Lucca, Italy

Jorge A. Pérez
University of Groningen & CWI, Amsterdam, The Netherlands

March 20, 2017

Abstract

Under a reversible semantics, computation steps can be undone. This paper addresses the integration of reversible semantics into process languages for communication-centric systems, equipped with *behavioral types*. In prior work, we introduced a *monitors-as-memories* approach to seamlessly integrate reversible semantics into a process model in which concurrency is governed by *session types* (a class of behavioral types), covering binary (two-party) protocols with synchronous communications. Although such a model offers a simple setting for showcasing our approach, its expressiveness is rather limited. Here we substantially extend our approach, and use it to define reversible semantics for a very expressive process model that accounts for multiparty (n-party) protocols (*choreographies*), asynchronous communication, decoupled rollbacks, and process passing. As main technical result, we prove that our multiparty, reversible semantics is *causally-consistent*.

1 Introduction

This paper is about *reversible computation* in the context of models of concurrency for *communication-centric* software systems, i.e., collections of distributed software components whose concurrent interactions are governed by reciprocal dialogues or *protocols*.

Building upon process calculi techniques, these models provide a rigorous footing for message-passing concurrency; on top of them, many (static) analysis techniques based on *(behavioral) types* and *contracts* have been put forward to enforce key safety and liveness properties [12]. Reversibility is an appealing notion in concurrency at large [16], but especially so in communication-centric scenarios: it may elegantly abstract fault-tolerant communicating systems that react to unforeseen circumstances (say, local failures) by "undoing" computation steps so as to reach a consistent previous state.

In communication-centric software systems, protocols specify the intended communication structures among interacting components. We focus on process calculi equipped with behavioral types, which use those protocols as types to enforce communication correctness. The interest is in different flavors of *protocol conformance*, i.e., properties that ensure that each component respects its ascribed protocol. The

*Revision of May 25, 2017. This work was partially supported by COST Actions IC1201 (Behavioral Types for Reliable Large-Scale Software Systems), IC1402 (Runtime Verification beyond Monitoring), and IC1405 (Reversible Computation - Extending Horizons of Computing). Pérez has been partially supported by CNRS PICS project 07313 (SuCCeSS); he is also affiliated to the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS - PEst/UID/CEC/04516/2013), Universidade Nova de Lisboa, Portugal.

integration of reversibility in models of communication-centric systems has been addressed from various angles (cf. [22, 23, 2, 17]). Focusing on *session types* [10, 11] (a well established class of behavioral types), Tiezzi and Yoshida [22] were the first to integrate reversibility into a session π-calculus, following the seminal approach of Danos and Krivine [6]; in [22], however, session types are not used in the definition of reversible communicating systems, nor play a role in establishing their properties.

Triggered by this observation, our prior work [17, 18] develops a *monitors-as-memories* approach. The idea is to use *monitors* (run-time entities that enact protocol actions) as the *memories* needed to record and undo communication steps. There is a monitor for each protocol participant; the monitor includes a session type that describes the intended protocol. We use a *cursor* to "mark" the current protocol state in the type; the cursor can move to the future (enacting protocol actions) but also to the past (reversing protocol actions). The result is a streamlined process framework in which the key properties of a reversible semantics can be established with simple proofs, because session types narrow down the spectrum of possible process behaviors, allowing only those forward and backward actions that correspond to the declared protocols. The most significant of such properties is *causal consistency* [6], considered as the "right" criterion for reversing concurrent processes [16]. Intuitively, causal consistency ensures that reversible steps lead to system states that could been have reached by performing forward steps only. That is, causally consistent reversibility does not lead to extraneous states, not reachable through ordinary computations.

The framework in [17, 18], however, accounts only for reversible π-calculus processes implementing *binary sessions*, i.e., protocols between exactly two partners. Also, it considers *synchronous communication* instead of the more general (and practical) *asynchronous (queue-based) communication*. Hence, our prior work rules out an important class of real-life protocols, namely the *choreographies* that describe interaction scenarios among multiple parties without a single point of control. In *multiparty session types* [11], these choreographies are represented by a *global type* that can be projected as *local types* to obtain each participant's contribution to the entire interaction. Moving from binary to multiparty sessions is a significant jump in expressiveness; in fact, global types offer a convenient declarative description of the entire communication scenario. However, the multiparty case also entails added challenges, as two levels of abstraction, global and local, should be considered for (reversible) protocols and their implementations. Hence, it is far from obvious that our monitors-as-memories approach to reversibility and causal consistency extend to the multiparty case.

This paper makes the following contributions:

1. We introduce a process model for reversible, multiparty sessions with asynchrony (as in [14]), process passing [20, 13] and decoupled rollbacks (§ 2). We define forward and backward semantics for multiparty processes by extending the monitors-as-memories approach to both global types and their implementations.

2. We prove that reversibility in our model is causally consistent (Theorem 4.3). The proof is challenging as we must appeal to an alternative reversible semantics with *atomic rollbacks*, which we show to coincide with the decoupled rollbacks (Theorem 4.2).

3. We formally connect reversibility at the (declarative) level of global types and that at the (operational) level of processes monitored by local types with cursors (Theorem 4.4).

We stress that asynchrony, process passing, and decoupled rollbacks are not considered in prior works [23, 8, 17, 18]. Asynchrony and decoupled rollbacks are delicate issues in a reversible multiparty setting—we do not know of other asynchronous calculi with reversible semantics, nor featuring the same combination of constructs. The formal connection between global and local levels of abstraction (Theorem 4.4) is also unique to our multiparty setting.

Organization This paper is organized as follows. In § 2, we introduce our process model of reversible choreographies. We illustrate the model by means of an example in § 3. In § 4 we establish causal consistency by relating decoupled and atomic semantics, and connect reversibility at global and local levels. § 5 contrasts with related works, while § 6 collects some concluding remarks. *The appendix contains additional material.*

2 Reversible Choreographies

Choreographies are defined in terms of *global types*, which declaratively describe a protocol among two or more participants. A global type can be *projected* onto each participant so as to obtain its corresponding *local type*, i.e., a session type that abstracts a participant's contribution to the global protocol. (Below we often use 'choreographies' and 'global types' as synonyms.) The semantics of global types is given in terms of forward and backward transition systems (Fig. 2). There is a *configuration* for each protocol participant: it includes a *located process* that specifies asynchronous communication behavior, subject to a *monitor* that enables forward/backward steps at run-time based on the local type. The semantics of configurations is given in terms of forward and backward reduction relations (Figs. 5 and 6).

Remark 2.1 (Colors)**.** Throughout the paper, we use colors to improve readability. In particular, elements in blue belong to a forward semantics; elements in red belong to a backward semantics.

2.1 Global and Local Types

2.1.1 Syntax

Let us write $\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{A}, \mathtt{B} \ldots$ to denote (protocol) *participants*. The syntax of global types ($G, G', \ldots$) and local types ($T, T', \ldots$) is standard [11] and defined as follows:

$$\begin{array}{rcl} G, G' & ::= & \mathtt{p} \to \mathtt{q} : \langle U \rangle.G \quad | \quad \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I} \\ & | & \mu X.G \quad | \quad X \quad | \quad \mathtt{end} \\ U, U' & ::= & \mathtt{bool} \quad | \quad \mathtt{nat} \quad | \quad \cdots \quad | \quad T \to \diamond \\ T, T' & ::= & \mathtt{p}!\langle U \rangle.T \mid \mathtt{p}?\langle U \rangle.T \\ & & \mid \mathtt{p}\oplus\{l_i : T_i\}_{i \in I} \mid \mathtt{p}\&\{l_i : T_i\}_{i \in I} \mid \mu X.T \mid X \mid \mathtt{end} \end{array}$$

Global type $\mathtt{p} \to \mathtt{q} : \langle U \rangle.G$ says that $\mathtt{p}$ may send a value of type U to $\mathtt{q}$, and then continue as G. Given a finite index set I and pairwise different *labels* l_i, global type $\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}$ specifies that $\mathtt{p}$ may choose label l_i, communicate this selection to $\mathtt{q}$, and then continue as G_i. In these two types we assume that $\mathtt{p} \neq \mathtt{q}$. Global recursive and terminated protocols are denoted $\mu X.G$ and $\mathtt{end}$, respectively. We write $\mathtt{pa}(G)$ to denote the set of participants in G. Value types U include basic first-order values (constants), but also *higher-order* values: abstractions from names to processes. (We write $\diamond$ to denote the type of processes.) Local types $\mathtt{p}!\langle U \rangle.T$ and $\mathtt{p}?\langle U \rangle.T$ denote, respectively, an output and input of value of type U by $\mathtt{p}$. We use α to denote type prefixes $\mathtt{p}?(U)$, $\mathtt{p}!\langle U \rangle$. Type $\mathtt{p}\&\{l_i : T_i\}_{i \in I}$ says that $\mathtt{p}$ offers different behaviors, available as labeled alternatives; conversely, type $\mathtt{p}\oplus\{l_i : T_i\}_{i \in I}$ says that $\mathtt{p}$ may select one of such alternatives. Terminated and recursive local types are denoted $\mathtt{end}$ and $\mu X.T$, respectively.

As usual, we consider only recursive types $\mu X.G$ (and $\mu X.T$) in which X occurs guarded in G (and T). We shall take an equi-recursive view of (global and local) types, and so we consider two types with the same regular tree as equal.

$$(\mathtt{p} \to \mathtt{q} : \langle U \rangle.G)\downarrow_{\mathtt{r}} = \begin{cases} \mathtt{q}!\langle U \rangle.(G\downarrow_{\mathtt{r}}) & \text{if } \mathtt{r} = \mathtt{p} \\ \mathtt{p}?\langle U \rangle.(G\downarrow_{\mathtt{r}}) & \text{if } \mathtt{r} = \mathtt{q} \\ (G\downarrow_{\mathtt{r}}) & \text{if } \mathtt{r} \neq \mathtt{q}, \mathtt{r} \neq \mathtt{p} \end{cases}$$

$$(\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I})\downarrow_{\mathtt{r}} = \begin{cases} \mathtt{q}\oplus\{l_i : (G_i\downarrow_{\mathtt{r}})\}_{i \in I} & \text{if } \mathtt{r} = \mathtt{p} \\ \mathtt{p}\&\{l_i : G_i\downarrow_{\mathtt{r}}\}_{i \in I} & \text{if } \mathtt{r} = \mathtt{q} \\ (G_1\downarrow_{\mathtt{r}}) & \text{if } \mathtt{r} \neq \mathtt{q}, \mathtt{r} \neq \mathtt{p} \text{ and} \\ & \forall i, j \in I. G_i\downarrow_{\mathtt{r}} = G_j\downarrow_{\mathtt{r}} \end{cases}$$

$$(\mu X.G)\downarrow_{\mathtt{r}} = \begin{cases} \mu X.G\downarrow_{\mathtt{r}} & \text{if } \mathtt{r} \text{ occurs in } G \\ \mathsf{end} & \text{otherwise} \end{cases}$$

$$X\downarrow_{\mathtt{r}} = X \qquad \mathsf{end}\downarrow_{\mathtt{r}} = \mathsf{end}$$

Figure 1: Projection of a global type G onto a participant $\mathtt{r}$.

Global and local types are connected by *projection*: following [11], the projection of G onto participant $\mathtt{p}$, written $G\downarrow_{\mathtt{p}}$, is defined in Fig. 1. Projection for $\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}$ is noteworthy: the projections of the participants not involved in the choice (different from $\mathtt{p}, \mathtt{q}$) should correspond to the same identical local type.

2.1.2 Semantics of Choreographies

The semantics of global types (Fig. 2) comprises forward and backward transition rules. To express backward steps, we require some auxiliary notions. We use *global contexts*, ranged over by $\mathbb{G}, \mathbb{G}', \ldots$ with holes $\bullet$, to record previous actions, including the choices discarded and committed:

$$\mathbb{G} \quad ::= \quad \bullet \mid \mathbb{G}[\mathtt{p} \to \mathtt{q} : \langle U \rangle.\mathbb{G}] \mid \mathbb{G}[\mathtt{p} \to \mathtt{q} : \{l_i : G_i \,;\, l_j : \mathbb{G}\}_{i \in I \setminus j}]$$

We also use *global types with history*, ranged over by $\mathsf{H}, \mathsf{H}', \ldots$, to record the current protocol state. This state is denoted by the *cursor* $\hat{}$, which we introduced in [17]:

$$\begin{aligned} \mathsf{H}, \mathsf{H}' \quad ::= \quad & \hat{}G \;\mid\; G\,\hat{} \;\mid\; \mathtt{p} \to \hat{}\mathtt{q} : \langle U \rangle.G \;\mid\; \mathtt{p} \to \mathtt{q} : \langle U \rangle.\hat{}G \\ \mid\; & \mathtt{p} \to \hat{}\mathtt{q} : \{l_i : G_i \,;\, l_j : G_j\}_{i \in I \setminus j} \\ \mid\; & \mathtt{p} \to \mathtt{q} : \{l_i : G_i \,;\, l_j : \hat{}G_j\}_{i \in I \setminus j} \end{aligned}$$

Intuitively, directed exchanges such as $\mathtt{p} \to \mathtt{q} : \langle U \rangle.G$ have three *intermediate states*, characterized by the decoupled involvement of $\mathtt{p}$ and $\mathtt{q}$ in the intended asynchronous model. The *first state*, denoted $\hat{}\mathtt{p} \to \mathtt{q} : \langle U \rangle.G$, describes the situation prior to the exchange. The *second state* represents the point in which $\mathtt{p}$ has sent a value of type U but this message has not yet reached $\mathtt{q}$; this is denoted $\mathtt{p} \to \hat{}\mathtt{q} : \langle U \rangle.G$. The *third state* represents the point in which $\mathtt{q}$ has received the message from $\mathtt{p}$ and the continuation G is ready to execute; this is denoted by $\mathtt{p} \to \mathtt{q} : \langle U \rangle.\hat{}G$. These intuitions extend to $\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}$, with the following caveat: the second state should distinguish the choice made by $\mathtt{p}$ from the discarded alternatives; we write $\mathtt{p} \to \hat{}\mathtt{q} : \{l_i : G_i \,;\, l_j : G_j\}_{i \in I \setminus j}$ to describe that $\mathtt{p}$ has selected l_j and that this choice is still to be received by $\mathtt{q}$. Once this occurs, a state $\mathtt{p} \to \mathtt{q} : \{l_i : G_i \,;\, l_j : \hat{}G_j\}_{i \in I \setminus j}$ is reached.

$$\text{(FVal1)}\ \mathbb{G}[\,\hat{}\,\mathtt{p} \to \mathtt{q} : \langle U\rangle.G] \hookrightarrow \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \langle U\rangle.G]$$

$$\text{(FVal2)}\ \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \langle U\rangle.G] \hookrightarrow \mathbb{G}[\mathtt{p} \to \mathtt{q} : \langle U\rangle.\,\hat{}\,G]$$

$$\text{(FCho1)}\quad \mathbb{G}[\,\hat{}\,\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i\in I}] \hookrightarrow \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \{l_i : G_i\,;\, l_j : G_j\}_{i\in I\setminus j}]$$

$$\text{(FCho2)}\quad \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \{l_i : G_i\,;\, l_j : G_j\}_{i\in I\setminus j}] \hookrightarrow \mathbb{G}[\mathtt{p} \to \mathtt{q} : \{l_i : G_i\,;\, l_j : \hat{}\,G_j\}_{i\in I\setminus j}]$$

$$\text{(BVal1)}\ \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \langle U\rangle.G] \rightharpoonup \mathbb{G}[\,\hat{}\,\mathtt{p} \to \mathtt{q} : \langle U\rangle.G]$$

$$\text{(BVal2)}\ \mathbb{G}[\mathtt{p} \to \mathtt{q} : \langle U\rangle.\,\hat{}\,G] \rightharpoonup \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \langle U\rangle.G]$$

$$\text{(BCho1)}\quad \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \{l_i : G_i\,;\, l_j : G_j\}_{i\in I\setminus j}] \rightharpoonup \mathbb{G}[\,\hat{}\,\mathtt{p} \to \mathtt{q} : \{l_i : G_i\,;\, l_j : G_j\}_{i\in I\setminus j}]$$

$$\text{(BCho2)}\quad \mathbb{G}[\mathtt{p} \to \mathtt{q} : \{l_i : G_i\,;\, l_j : \hat{}\,G_j\}_{i\in I\setminus j}] \rightharpoonup \mathbb{G}[\mathtt{p} \to \hat{}\,\mathtt{q} : \{l_i : G_i\,;\, l_j : G_j\}_{i\in I\setminus j}]$$

Figure 2: Semantics of Global Types (Forward & Backwards).

These intuitions come in handy to describe the forward and backward transition rules in Fig. 2. For a forward directed exchange of a value, Rule (FVal1) formalizes the transition from the first to the second state; Rule (FVal2) denotes the transition from the second to the third state. Rules (FCho1) and (FCho2) are their analogues for the forward directed communication of a label. Rules (BVal1) and (BVal2) undo the step performed by Rules (FVal1) and (FVal2), respectively. Also, Rules (BCho1) and (BCho2) undo the step performed by Rules (FCho1) and (FCho2), respectively.

2.2 Processes and Configurations

2.2.1 Syntax

The syntax of processes and configurations is given in Fig. 3. For processes $P, Q, \ldots$ we follow closely the syntax of HOπ, the core higher-order session π-calculus [13]. The syntax of configurations builds upon that of processes.

Names a, b, c (resp. s, s') range over shared (resp. session) names. We use session names indexed by participants, denoted $s[\mathtt{p}], s[\mathtt{q}]$. Names n, m are session or shared names. First-order values v, v' include base values and constants. Variables are denoted by x, y, and recursive variables are denoted by X, Y. The syntax of values V includes shared names, first-order values, but also name abstractions (higher-order values) $\lambda x.\, P$, where P is a process. As shown in [13], abstraction passing suffices to express name passing (*delegation*).

Process terms include prefixes for sending and receiving values V, written $u!\langle V\rangle.P$ and $u?(x).P$, respectively. Given a finite index set I, processes $u \triangleleft \{l_i.P_i\}_{i\in I}$ and $u \triangleright \{l_i : P_i\}_{i\in I}$ implement selection and branching (internal and external labeled choices, respectively). The selection $u \triangleleft \{l_i.P_i\}_{i\in I}$ is actually

$$
\begin{array}{rcl}
u, w & ::= & n \;\mid\; x, y, z \qquad n, m \;::=\; a, b \;\mid\; s[\mathtt{p}] \\
v, v' & ::= & \mathtt{tt} \;\mid\; \mathtt{ff} \;\mid\; \cdots \\
V, W & ::= & a, b \;\mid\; x, y, z \;\mid\; v, v' \;\mid\; \lambda x.\, P \\
P, Q & ::= & u!\langle V\rangle.P \mid u?(x).P \mid u \triangleleft \{l_i.P_i\}_{i\in I} \mid u \triangleright \{l_i : P_i\}_{i\in I} \\
 & & \mid P \mid Q \mid X \mid \mu X.P \mid V\, u \mid (\nu\, n)P \mid \mathbf{0} \\
M, N & ::= & \mathbf{0} \mid \ell\,\{a!\langle x\rangle.P\} \mid \ell\,\{a?(x).P\} \mid M \mid N \mid (\nu\, n)M \\
 & & \mid \boxed{\ell : \wr \mathtt{C} \,;\, P \int} \mid \boxed{s \lfloor H \cdot \widetilde{x} \cdot \sigma \rfloor^{\spadesuit}} \\
 & & \mid \boxed{s : (h_i \star h_o)} \mid \boxed{k \lfloor (V\, u)\,,\, \ell \rfloor} \\
\mathtt{C}, \mathtt{C}' & ::= & \mathbf{0} \mid u \triangleleft \{l_i.P_i\}_{i\in I} \mid u \triangleright \{l_i : P_i\}_{i\in I} \mid \mathtt{C}_1, \mathtt{C}_2 \\
\spadesuit & ::= & \blacklozenge \mid \lozenge \qquad h \;::=\; \epsilon \mid h \circ (\mathtt{p}\,,\, \mathtt{q}\,,\, m) \qquad m \;::=\; V \mid l \\
\\
\alpha & ::= & \mathtt{q}?(U) \mid \mathtt{q}!\langle U\rangle \\
T, S & ::= & \mathsf{end} \mid \alpha.S \mid \mathtt{q}\oplus\{l_i : S_i\}_{i\in I} \mid \mathtt{q}\&\{l_i : S_i\}_{i\in I} \\
H, K & ::= & \hat{}S \mid S\hat{} \mid \alpha_1.\cdots.\alpha_n.\hat{}S \\
 & & \mid \mathtt{q}\oplus\{l_i : S_i \;;\; l_j : H_j\}_{i\in I} \mid \mathtt{q}\&\{l_i : S_i \,,\, l_j : H_j\}_{i\in I}
\end{array}
$$

Figure 3: Syntax of processes P, Q, configurations M, N, stacks $\mathtt{C}, \mathtt{C}'$, local types T, S, local types with history H, K. Constructs given in $\boxed{\text{boxes}}$ appear only at run-time.

a non-deterministic choice over I. In an improvement with respect to [17, 18], here we consider parallel composition of processes $P \mid Q$ and recursion $\mu X.P$ (which binds the recursive variable X in process P). Process $V\, u$ is the application which substitutes name u on the abstraction V. Constructs for name restriction $(\nu\, n)P$ and inaction $\mathbf{0}$ are standard. Session restriction $(\nu\, s)P$ simultaneously binds all the participant endpoints in P. We write $\mathtt{fv}(P)$ and $\mathtt{fn}(P)$to denote the sets of free variables and names in P. We assume V in $u!\langle V\rangle.P$ does not include free recursive variables X. If $\mathtt{fv}(P) = \emptyset$, we call P *closed*.

The syntax of configurations $M, N, \ldots$, includes inaction $\mathbf{0}$, the parallel composition $M \mid N$, and name restriction $(\nu\, n)M$. Also, it includes constructs for *session initiation*: configuration $\ell\,\{a!\langle x\rangle.P\}$ denotes the *request* of a service identified with a implemented in P as x; conversely, configuration $\ell\,\{a?(x).P\}$ denotes service *acceptance*. In both constructs, identifiers $\ell, \ell', \ldots$ denote a process *location* or *site* (as in, e.g., the distributed π-calculus [9]).

Configurations also include the following *run-time elements*:

- *Running processes* are of the form $\ell : \wr \mathtt{C} \,;\, P \int$, where ℓ is a location that hosts a process P and a *(process) stack* $\mathtt{C}$. A process stack is simply a list of processes, useful to record/reinstate the discarded alternatives in a labeled choice.

- *Monitors* are of the form $s \lfloor H \cdot \widetilde{x} \cdot \sigma \rfloor^{\spadesuit}$ where s is the session being monitored, H is a history session type (i.e. a session type with "memory"), $\widetilde{x}$ is a set of free variables, and the *store* σ records the value of such variables (see Def. 2.1). These four elements allow us to track the current protocol and state of the monitored process. Each monitor has a *tag* $\spadesuit$, which can be either *empty* (denoted '$\lozenge$') or *full*

$$A \mid \mathbf{0} \equiv A \quad A \mid B \equiv B \mid A \quad A \mid (B \mid C) \equiv (A \mid B) \mid C$$
$$A \mid (\nu\, n)B \equiv (\nu\, n)(A \mid B)\ (n \notin \mathtt{fn}(P)) \quad (\nu\, n)\mathbf{0} \equiv \mathbf{0}$$
$$\mu X.P \equiv P\{{}^{\mu X.P}/X\} \quad A \equiv B \text{ if } A \equiv_\alpha B$$

Figure 4: Structural Congruence

(denoted '♦'). When first created all monitors have an empty tag; a full tag indicates that the running process associated to the monitor is currently involved in a decoupled reversible step. We often omit the empty tag (so we write $s\,\lfloor H \cdot \widetilde{x} \cdot \sigma \rfloor$ instead of $s\,\lfloor H \cdot \widetilde{x} \cdot \sigma \rfloor^{\diamond}$) and write $s\,\lfloor H \cdot \widetilde{x} \cdot \sigma \rfloor^{\blacklozenge}$ to emphasize the reversible (red) nature of a monitor with full tag.

- Following [14], we have *message queues* of the form $s : (h_i \star h_o)$, where s is a session, h_i is the input part of the queue, and h_o is the output part of the queue. Each queue contains messages of the form $(\mathtt{p}\,,\,\mathtt{q}\,,\,m)$ (read: "message m is sent from $\mathtt{p}$ to $\mathtt{q}$"). As we will see, the effect of an output prefix in a process is to place the message in its corresponding output queue; conversely, the effect of an input prefix is to obtain the first message from its input queue. Messages in the queue are *never consumed*: a process reads a message $(\mathtt{p}\,,\,\mathtt{q}\,,\,m)$ by moving it from the (tail of) queue h_o to the (top of) queue h_i. This way, the delimiter '$\star$' distinguishes the *past* of the queue from its *future*.

- We use *running functions* of the form $k\,\lfloor (V\,u)\,,\,\ell \rfloor$ to reverse applications $V\,u$. While k is a fresh identifier (key) for this term, ℓ is the location of the running process that contains the application.

We shall write $\mathcal{P}$ and $\mathcal{M}$ to indicate the set of processes and configurations, respectively. We call *agent* an element of the set $\mathcal{A} = \mathcal{M} \cup \mathcal{P}$. We let P, Q to range over $\mathcal{P}$; also, we use L, M, N to range over $\mathcal{M}$ and A, B, C to range over $\mathcal{A}$.

2.2.2 A Decoupled Semantics for Configurations

We define a reduction relation on configurations, coupled with a structural congruence on processes and configurations. Our reduction semantics defines a *decoupled* treatment for reversing communication actions within a protocol. Reduction is thus defined as $\longrightarrow \subset \mathcal{M} \times \mathcal{M}$, whereas structural congruence is defined as $\equiv \subset \mathcal{P}^2 \cup \mathcal{M}^2$. We require auxiliary definitions for *contexts*, *stores*, and *type contexts*.

Evaluation contexts are configurations with one hole '•', as defined by the following grammar:

$$\mathbb{E} \ ::= \ \bullet \ \mid \ M \mid \mathbb{E} \ \mid \ (\nu\, n)\,\mathbb{E}$$

General contexts $\mathbb{C}$ are processes or configurations with one hole •: they are obtained by replacing one occurrence of **0** (either as a process or as a configuration) with •. A congruence on processes and configurations is an equivalence $\Re$ that is closed under general contexts: $P \,\Re\, Q \Longrightarrow \mathbb{C}[P] \,\Re\, \mathbb{C}[Q]$ and $M \Re N \Longrightarrow \mathbb{C}[M] \,\Re\, \mathbb{C}[N]$. We define $\equiv$ as the smallest congruence on processes and configurations that satisfies rules in Fig. 4. A relation $\Re$ on configurations is *evaluation-closed* if it satisfies the following rules:

$$(\textsc{Ctx})\ \frac{M \,\Re\, N}{\mathbb{E}[M] \,\Re\, \mathbb{E}[N]} \qquad\qquad (\textsc{Eqv})\ \frac{M \equiv M' \qquad M' \,\Re\, N' \qquad N' \equiv N}{M \,\Re\, N}$$

The state of monitored processes is formalized as follows:

Definition 2.1. A store σ is a mapping from variables to values. Given a store σ, a variable x, and a value V, the *update* $\sigma[x \mapsto V]$ and the *reverse update* $\sigma \setminus x$ are defined as follows:

$$\sigma[x \mapsto V] = \begin{cases} \sigma \cup \{(x, V)\} & \text{if } x \notin \mathtt{dom}(\sigma) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\sigma \setminus x = \begin{cases} \sigma_1 & \text{if } \sigma = \sigma_1 \cup \{(x, V)\} \\ \sigma & \text{otherwise} \end{cases}$$

Together with local types with history, the following notion of type context allows us to record the current protocol state:

Definition 2.2. Let $k, k', \ldots$ denote fresh name identifiers. We define *type contexts* as (local) types with one hole, denoted "$\bullet$":

$$\begin{aligned} \mathbb{T}, \mathbb{S} \quad ::= \quad & \bullet \mid \mathtt{q} \oplus \{l_w : \mathbb{T}\ ;\ l_i : S_i\}_{i \in I \setminus w} \mid \mathtt{q}\&\{l_w : \mathbb{T}\ ,\ l_i : S_i\}_{i \in I \setminus w} \\ & \mid \alpha.\mathbb{T} \mid k.\mathbb{T} \mid (\ell, \ell_1, \ell_2).\mathbb{T} \end{aligned}$$

Type contexts $k.\mathbb{T}$ and $(\ell, \ell_1, \ell_2).\mathbb{T}$ will be instrumental in formalizing reversibility of name applications and thread spawning, respectively, which are not described by local types.

Abstraction passing can implement a form of *session delegation*, for received abstractions $\lambda x.\, P$ can contain free session names (indexed by participant identities). The following definition identifies those names:

Definition 2.3. Let h and $\mathtt{p}$ be a queue and a participant, respectively. Also, let $\{(\mathtt{q}_1\ , \mathtt{p}\ , \lambda x_1.\, P_1), \ldots, (\mathtt{q}_k\ , \mathtt{p}\ , \lambda x_k.\, P_k)\}$ denote the (possibly empty) set of messages in h containing abstractions sent to $\mathtt{p}$. We write $\mathbf{roles}(\mathtt{p}, h)$ to denote the set of participant identities occurring in $P_1, \ldots, P_k$.

The reduction relation $\longrightarrow$ is defined as the union of two relations: the forward and backward reduction relations, denoted $\twoheadrightarrow$ and $\rightsquigarrow$, respectively. That is, $\longrightarrow = \twoheadrightarrow \cup \rightsquigarrow$. Relations $\twoheadrightarrow$ and $\rightsquigarrow$ are the smallest evaluation-closed relations satisfying the rules in Figs. 5 and 6. We indicate with $\longrightarrow^*$, $\twoheadrightarrow^*$, and $\rightsquigarrow^*$ the reflexive and transitive closure of $\longrightarrow$, $\twoheadrightarrow$ and $\rightsquigarrow$, respectively. We now discuss the forward reduction rules (Fig. 5), omitting empty tags $\Diamond$:

- Rule (Init) initiates a choreography G with n participants. Given the composition of one service request and $n - 1$ service accepts (all along a, available in different locations ℓ_i), this rule sets up the run-time elements: running processes and monitors—one for each participant, with empty tag (omitted)—and the empty session queue. A unique session identifier (s in the rule) is also created. The processes are inserted in their respective running structures, and instantiated with an appropriate session name. Similarly, the local types for each participant are inserted in their respective monitor, with the cursor $\hat{}$ at the beginning.

- Rule (Out) starts the output of value V from $\mathtt{p}$ to $\mathtt{q}$. Given an output-prefixed process as running process, and a monitor with a local type supporting an output action, reduction adds the message $(\mathtt{p}\ , \mathtt{q}\ , \sigma(V))$ to the output part of the session queue (where σ is the current store). Also, the cursor within the local type is moved accordingly. In this rule (but also in several other rules), 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.

$$\text{(Init)}\quad \frac{\mathtt{pa}(G) = \{\mathtt{p}_1, \cdots, \mathtt{p}_n\} \qquad \forall \mathtt{p}_i \in \mathtt{pa}(G).\, G{\downarrow_{\mathtt{p}_i}} = T_i}{\begin{array}{c}\ell_1\,\{a!\langle x_1 : T_1\rangle.P_1\} \mid \prod_{i\in\{2,\cdots,n\}} \ell_i\,\{a?(x_i : T_i).P_i\} \\ \twoheadrightarrow \\ (\nu\, s)\Big(\prod_{i\in\{1,\cdots,n\}} \ell_i[\mathtt{p}_i] : \wr\mathbf{0}\,;\, P_i\{{}^{s[\mathtt{p}_i]}/_{x_i}\}\int \mid s_{\mathtt{p}_i}\lfloor\, \hat{}\,T_i \cdot x_i \cdot [x_i \mapsto a]\rfloor \mid s : (\epsilon \star \epsilon)\Big)\end{array}}$$

$$\text{(Out)}\quad \frac{\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)}{\begin{array}{c}\ell[\mathtt{r}] : \wr\mathtt{C}\,;\, s[\mathtt{p}]!\langle V\rangle.P \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\,\hat{}\,\mathtt{q}!\langle U\rangle.S] \cdot \widetilde{x} \cdot \sigma\rfloor \mid s : (h_i \star h_o) \\ \twoheadrightarrow \\ \ell[\mathtt{r}] : \wr\mathtt{C}\,;\, P \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\mathtt{q}!\langle U\rangle.\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma\rfloor \mid s : (h_i \star h_o \circ (\mathtt{p}, \mathtt{q}, \sigma(V)))\end{array}}$$

$$\text{(In)}\quad \frac{\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)}{\begin{array}{c}\ell[\mathtt{r}] : \wr\mathtt{C}\,;\, s[\mathtt{p}]?(y).P \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\,\hat{}\,\mathtt{q}?\langle U\rangle.S] \cdot \widetilde{x} \cdot \sigma\rfloor \mid s : (h_i \star (\mathtt{q}, \mathtt{p}, V) \circ h_o) \\ \twoheadrightarrow \\ \ell[\mathtt{r}] : \wr\mathtt{C}\,;\, P \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\mathtt{q}?\langle U\rangle.\,\hat{}\,S] \cdot \widetilde{x}, y \cdot \sigma[y \mapsto V]\rfloor \mid s : (h_i \circ (\mathtt{q}, \mathtt{p}, V) \star h_o)\end{array}}$$

(Sel)

$$\frac{\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i) \qquad w \in J \qquad J \subseteq I}{\begin{array}{c}\ell[\mathtt{r}] : \wr\mathtt{C}\,;\, s[\mathtt{p}] \triangleleft \{l_i.P_i\}_{i\in I} \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\,\hat{}\,\mathtt{q}\oplus\{l_j : S_j\}_{j\in J}] \cdot \widetilde{x} \cdot \sigma\rfloor \mid s : (h_i \star h_o) \\ \twoheadrightarrow \\ \ell[\mathtt{r}] : \wr\mathtt{C}, s[\mathtt{p}] \triangleleft \{l_i.P_i\}_{i\in I\setminus w}\,;\, P_w \int \mid s_{\mathtt{p}}\left\lfloor \mathbb{T}\left[\mathtt{q}\oplus\{l_j : S_j,\, l_w :\, \hat{}\,S_w\}_{j\in J\setminus w}\right] \cdot \widetilde{x} \cdot \sigma\right\rfloor \mid s : (h_i \star h_o \circ (\mathtt{p}, \mathtt{q}, l_w))\end{array}}$$

(Bra)

$$\frac{\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i) \qquad w \in I \qquad I \subseteq J}{\begin{array}{c}\ell[\mathtt{r}] : \wr\mathtt{C}\,;\, s[\mathtt{p}] \triangleright \{l_i : P_i\}_{i\in I} \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\,\hat{}\,\mathtt{q}\&\{l_j : S_j\}_{j\in J}] \cdot \widetilde{x} \cdot \sigma\rfloor \mid s : (h_i \star (\mathtt{q}, \mathtt{p}, l_w) \circ h_o) \\ \twoheadrightarrow \\ \ell[\mathtt{r}] : \wr\mathtt{C}, s[\mathtt{p}] \triangleright \{l_i : P_i\}_{i\in I\setminus w}\,;\, P_w \int \mid s_{\mathtt{p}}\left\lfloor \mathbb{T}\left[\mathtt{q}\&\{l_j : S_j,\, l_w :\, \hat{}\,S_w\}_{j\in J\setminus w}\right] \cdot \widetilde{x} \cdot \sigma\right\rfloor \mid s : (h_i \circ (\mathtt{q}, \mathtt{p}, l_w) \star h_o)\end{array}}$$

(Beta)

$$\frac{\sigma(V) = \lambda x.\, P}{\ell[\mathtt{p}] : \wr\mathtt{C}\,;\, (V\, w) \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma\rfloor \twoheadrightarrow (\nu\, k)\ \big(\ell[\mathtt{p}] : \wr\mathtt{C}\,;\, P\{{}^{\sigma(w)}/_{x}\}\int \mid k\,\lfloor (V\, w),\, \ell\rfloor \mid s_{\mathtt{p}}\lfloor \mathbb{T}[k.\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma\rfloor\big)}$$

(Spawn)

$$\frac{}{\begin{array}{c}\ell[\mathtt{p}] : \wr\mathtt{C}\,;\, P \mid Q \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma\rfloor \\ \twoheadrightarrow \\ (\nu\, \ell_1, \ell_2)\ \big(\ell[\mathtt{p}] : \wr\mathtt{C}\,;\, \mathbf{0} \int \mid \ell_1[\mathtt{p}] : \wr\mathbf{0}\,;\, P \int \mid \ell_2[\mathtt{p}] : \wr\mathbf{0}\,;\, Q \int \mid s_{\mathtt{p}}\lfloor \mathbb{T}[(\ell, \ell_1, \ell_2).\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma\rfloor\big)\end{array}}$$

Figure 5: Decoupled semantics for configurations: Forward reduction ($\twoheadrightarrow$).

- ▶ Rule (In) allows a participant p to receive a value V from q: it simply takes the first element of the output part of the queue and places it in the input part. The cursor of the local type and state in the monitor for p are updated accordingly.

- ▶ Rule (Sel) is the forward rule for labeled selection, which in our case entails a non-deterministic choice between pairwise different labels indexed by I. We require that I is contained in J, i.e., the set that indexes the choice according to the choreography. After reduction, the selected label (l_w in the rule) is added to the output part of the queue, and the continuation P_w is kept in the running process; to support reversibility, alternatives different from l_w are stored in the stack C with their continuations. The cursor is also appropriately updated in the monitor.

- ▶ Rule (Bra) is similar to Rule (Sel): it takes a message containing a label l_w as the first element in the output part of the queue, and places it into the input part. This entails a selection between the options indexed by I; the continuation P_w is kept in the running process, and all those options different from l_w are kept in the stack. Also, the local type in the monitor is updated accordingly.

- ▶ Rule (Beta) handles name applications. Reduction creates a fresh identifier (k in the rule) for the running function, which keeps (i) the structure of the process prior to application, and (ii) the identifier of the running process that "invokes" the application. Notice that k is recorded also in the monitor: this is necessary to undo applications in the proper order. To determine the actual abstraction and the name applied, we use σ.

- ▶ Rule (Spawn) handles parallel composition. Location ℓ is "split" into running processes with fresh identifiers (ℓ_1, ℓ_2 in the rule). This split is recorded in the monitor.

Now we comment on the backward rules (Fig. 6) which, in most cases, change the monitor tags from ♢ into ♦:

- ◀ Rule (RInit) undoes session establishment. It requires that local types for every participant are at the beginning of the protocol, and empty session queue and process stacks. Run-time elements are discarded; located service accept/requests are reinstated.

- ◀ Rule (RollS) starts to undo an input-output synchronization between p and q. Enabled when there are complementary session types in the two monitors, this rule changes the monitor tags from ♢ to ♦. This way, the undoing of input and output actions occurs in a decoupled way. Rule (RollC) is the analog of (RollS) but for synchronizations originated in labeled choices.

- ◀ Rule (ROut) undoes an output. This is only possible for a monitor tagged with ♦, exploiting the first message in the input queue. After reduction, the process prefix is reinstated, the cursor is adjusted, the message is removed from the queue, the monitor is tagged again with ♢. Rule (RIn) is the analog of Rule (ROut). In this case, we also need to update the state of store σ.

- ◀ Rule (RBra) undoes the input part of a labeled choice: the choice context is reinstated; the cursor is moved; the last message in the input part of the queue is moved to the output part. Rule (RSel) is the analog of (RBra), but for the output part of the labeled choice. The non-deterministic selection is reinstated.

- ◀ Rule (RBeta) undoes β-reduction, reinstating the application. The running function disappears, using the information in the monitor (k in the rule). Rule (RSpawn) undoes the spawn of a parallel thread, using the identifiers in the monitor.

We now illustrate our reversible process model with an example.

3 Example: the Three-Buyer protocol

We illustrate our framework by presenting a reversible variant of the *Three-Buyer protocol* (see, e.g., [5]) with abstraction passing (code mobility), one of the distinctive traits of our framework. (In App. A.1 we discuss an additional example involving labeled choices.)

The protocol involves three buyers—Alice (A), Bob (B), and Carol (C)—who interact with a Seller (S) as follows:

1. Alice sends a book title to Seller, which replies back to Alice and Bob with a quote. Alice tells Bob how much she can contribute.
2. Bob notifies Seller and Alice that he agrees with the price, and asks Carol to assist him in completing the protocol. To delegate his remaining interactions with Alice and Seller to Carol, Bob sends her the code she must execute.
3. Carol continues the rest of the protocol with Seller and Alice as if she were Bob. She sends Bob's address (contained in the mobile code she received) to Seller.
4. Seller answers to Alice and Carol (who represents Bob) with the delivery date.

We formalize this protocol is as the global type G below. We write $\mathtt{p} \to \{\mathtt{q}_1, \mathtt{q}_2\} : \langle U \rangle.G$ as a shorthand notation for $\mathtt{p} \to \mathtt{q}_1 : \langle U \rangle.\mathtt{p} \to \mathtt{q}_2 : \langle U \rangle.G$ (and similarly for local types). We write $\{\{\diamond\}\}$ to denote the type $\mathsf{end} \to \diamond$, associated to a *thunk process* $\lambda x.\, P$ with $x \notin \mathtt{fn}(P)$, written $\{\{P\}\}$. A thunk is an inactive process; it can be activated by applying to it a dummy name of type end, denoted $*$. This way, we have $(\{\{P\}\}\, *) \twoheadrightarrow P$.

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

where price and share are base types treated as integers, and title, OK, address, and date are base types treated as strings.

Then we have the following projections of G onto local types:

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

We now give processes for each participant:

$$\begin{aligned}
\text{Seller} &= d!\langle x : G{\downarrow_{\mathtt{S}}}\rangle.x?(t).x!\langle price(t)\rangle.x!\langle price(t)\rangle.\\
&\qquad x?(ok).x?(a).x!\langle date\rangle.\mathbf{0}\\
\text{Alice} &= d?(y : G{\downarrow_{\mathtt{A}}}).y!\langle\text{‘Logicomix’}\rangle.y?(p).y?(s).y?(ok).\mathbf{0}\\
\text{Bob} &= d?(z : G{\downarrow_{\mathtt{B}}}).z?(p).z?(s).z!\langle ok\rangle.z!\langle ok\rangle.z!\langle s\rangle.\\
&\qquad\qquad z!\big\langle\{\{z!\langle\text{‘Lucca, 55100’}\rangle.z?(d).\mathbf{0}\}\}\big\rangle.\mathbf{0}\\
\text{Carol} &= d?(w : G{\downarrow_{\mathtt{C}}}).w?(s).w?(code).(code *)
\end{aligned}$$

where $price(\cdot)$ returns a value of type price given a title. Observe how Bob's implementation sends part of its protocol to Carol in the form of a thunk containing his session name and address. This is how abstraction passing may serve to implement session delegation.

The whole system, given by configuration M below, is obtained by placing these process implementations in appropriate locations:

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

The session starts with an application of Rule (Init):

$$\begin{aligned}
M \twoheadrightarrow (\nu\, s)\big(&\,\ell_1[\mathtt{S}] : \wr\mathbf{0}\ ;\ S_1\{{}^{s[\mathtt{S}]}/x\}\ \int\ \mid s_{\mathtt{S}}\,\lfloor\,\hat{}\,G{\downarrow_{\mathtt{S}}}\cdot x\cdot[x\mapsto d]\rfloor^{\diamond}\\
&\mid \ell_2[\mathtt{A}] : \wr\mathbf{0}\ ;\ A_1\{{}^{s[\mathtt{A}]}/y\}\ \int\ \mid s_{\mathtt{A}}\,\lfloor\,\hat{}\,G{\downarrow_{\mathtt{A}}}\cdot y\cdot[y\mapsto d]\rfloor^{\diamond}\\
&\mid \ell_3[\mathtt{B}] : \wr\mathbf{0}\ ;\ B_1\{{}^{s[\mathtt{B}]}/z\}\ \int\ \mid s_{\mathtt{B}}\,\lfloor\,\hat{}\,G{\downarrow_{\mathtt{B}}}\cdot z\cdot[z\mapsto d]\rfloor^{\diamond}\\
&\mid \ell_4[\mathtt{C}] : \wr\mathbf{0}\ ;\ C_1\{{}^{s[\mathtt{C}]}/w\}\ \int\ \mid s_{\mathtt{C}}\,\lfloor\,\hat{}\,G{\downarrow_{\mathtt{C}}}\cdot w\cdot[w\mapsto d]\rfloor^{\diamond}\\
&\mid s : (\epsilon \star \epsilon)\big) = M_1
\end{aligned}$$

where S_1, A_1, B_1, and C_1 stand for the continuation of processes Seller, Alice, Bob, and Carol after the service request/declaration. So, e.g., $A_1 = y!\langle\text{‘Logicomix’}\rangle.y?(p).y?(s).y?(ok).\mathbf{0}$. We use configuration M_1 to illustrate some forward and backward reductions.

From M_1 we could either undo the reduction (using Rule (RInit)) or execute the communication from Alice to Seller (using two rules: (Out) and (In)). This latter option would be as follows:

$$\begin{aligned}
M_1 \twoheadrightarrow (\nu\, s)(&\,\ell_2[\mathtt{A}] : \wr\mathbf{0}\ ;\ s[\mathtt{A}]?(p).s[\mathtt{A}]?(s).s[\mathtt{A}]?(ok).\mathbf{0}\int\\
&\mid s_{\mathtt{A}}\,\lfloor \mathtt{S}!\langle\mathsf{title}\rangle.\,\hat{}\,\mathtt{S}?\langle\mathsf{price}\rangle.\mathtt{B}!\langle\mathsf{share}\rangle.\mathtt{B}?\langle\mathsf{OK}\rangle.\mathsf{end}\cdot y\cdot[y\mapsto d]\rfloor^{\diamond}\\
&\mid N_2 \mid s : (\epsilon \star (\mathtt{A}\,,\,\mathtt{S}\,,\,\text{‘Logicomix’}))) = M_2
\end{aligned}$$

where N_2 stands for the running processes and monitors for Seller, Bob, and Carol, not involved in the reduction. We now have:

$$\begin{aligned}
M_2 \twoheadrightarrow (\nu\, s)(&\,\ell_1[\mathtt{S}] : \wr\mathbf{0}\ ;\ s[\mathtt{S}]!\langle price(t)\rangle.s[\mathtt{S}]!\langle price(t)\rangle.s[\mathtt{S}]?(ok).\\
&\qquad s[\mathtt{S}]?(a).s[\mathtt{S}]!\langle date\rangle.\mathbf{0}\int\\
&\mid s_{\mathtt{S}}\,\lfloor \mathtt{A}?\langle\mathsf{title}\rangle.\,\hat{}\,\{\mathtt{A},\mathtt{B}\}!\langle\mathsf{price}\rangle.T_{\mathtt{S}}\cdot x,t\cdot\sigma_3\rfloor^{\diamond} \mid N_3\\
&\mid s : ((\mathtt{A}\,,\,\mathtt{S}\,,\,\text{‘Logicomix’}) \star \epsilon)) = M_3
\end{aligned}$$

where $\sigma_3 = [x \mapsto d], [t \mapsto \text{‘Logicomix’}]$ is the resulting store, $T_{\mathtt{S}} = \mathtt{B}?\langle\mathsf{OK}\rangle.\mathtt{B}?\langle\mathsf{address}\rangle.\mathtt{B}!\langle\mathsf{date}\rangle.\mathsf{end}$, and N_3 stands for the participants not involved in the reduction. Observe that the cursors in monitors $s_{\mathtt{S}}$ and $s_{\mathtt{A}}$ have evolved, and that message from $\mathtt{A}$ to $\mathtt{S}$ has now been moved to the input queue.

We illustrate reversibility by showing how to return to M_1 starting from M_3. We need to apply three rules: (RollS), (RIn), and (ROut). Reversibility is decoupled in the sense that there is no fixed order in which the latter two rules should be applied; below we give just a possible sequence. First, Rule (RollS) modifies the tags of monitors $s_{\mathtt{S}}$ and $s_{\mathtt{A}}$, leaving the rest unchanged:

$$
\begin{aligned}
M_3 \rightsquigarrow (\nu\, s)(\, & \ell_1[\mathtt{S}] : \wr \mathbf{0} \,;\, s[\mathtt{S}]!\langle price(t)\rangle.s[\mathtt{S}]!\langle price(t)\rangle.s[\mathtt{S}]?(ok). \\
& \quad s[\mathtt{S}]?(a).s[\mathtt{S}]!\langle date\rangle.\mathbf{0} \wr \\
& \mid s_{\mathtt{S}} \lfloor \mathtt{A}?\langle \mathsf{title}\rangle.\,\hat{}\{\mathtt{A},\mathtt{B}\}!\langle \mathsf{price}\rangle.T_{\mathtt{B}} \cdot x, t \cdot \sigma_3 \rfloor^{\blacklozenge} \\
& \mid \ell_2[\mathtt{A}] : \wr \mathbf{0} \,;\, s[\mathtt{A}]?(p).s[\mathtt{A}]?(s).s[\mathtt{A}]?(ok).\mathbf{0} \wr \\
& \mid s_{\mathtt{A}} \lfloor \mathtt{S}!\langle \mathsf{title}\rangle.\,\hat{}\mathtt{S}?\langle \mathsf{price}\rangle.\mathtt{B}!\langle \mathsf{share}\rangle.\mathtt{B}?\langle \mathsf{OK}\rangle.\mathsf{end} \cdot y \cdot [y \mapsto d] \rfloor^{\blacklozenge} \\
& \mid N_4 \mid s : ((\mathtt{A}\,,\,\mathtt{S}\,,\,\text{‘Logicomix’}) \star \epsilon)) = M_4
\end{aligned}
$$

where, as before, N_4 represents participants not involved in the reduction. M_4 has several possible forward and backward reductions. One particular reduction uses Rule (RIn) to undo the input at $\mathtt{S}$:

$$
\begin{aligned}
M_4 \rightsquigarrow (\nu\, s)(\, & \ell_1[\mathtt{S}] : \wr \mathbf{0} \,;\, s[\mathtt{S}]?(t).s[\mathtt{S}]!\langle price(t)\rangle.s[\mathtt{S}]!\langle price(t)\rangle. \\
& \quad s[\mathtt{S}]?(ok).s[\mathtt{S}]?(a).s[\mathtt{S}]!\langle date\rangle.\mathbf{0} \wr \\
& \mid s_{\mathtt{S}} \lfloor \hat{}\mathtt{A}?\langle \mathsf{title}\rangle.\{\mathtt{A},\mathtt{B}\}!\langle \mathsf{price}\rangle.T_{\mathtt{B}} \cdot x \cdot [x \mapsto d] \rfloor^{\lozenge} \\
& \mid \ell_2[\mathtt{A}] : \wr \mathbf{0} \,;\, s[\mathtt{A}]?(p).s[\mathtt{A}]?(s).s[\mathtt{A}]?(ok).\mathbf{0} \wr \\
& \mid s_{\mathtt{A}} \lfloor \mathtt{S}!\langle \mathsf{title}\rangle.\,\hat{}\mathtt{S}?\langle \mathsf{price}\rangle.\mathtt{B}!\langle \mathsf{share}\rangle.\mathtt{B}?\langle \mathsf{OK}\rangle.\mathsf{end} \cdot y \cdot [y \mapsto d] \rfloor^{\blacklozenge} \\
& \mid N_4 \mid s : (\epsilon \star (\mathtt{A}\,,\,\mathtt{S}\,,\,\text{‘Logicomix’}))) = M_5
\end{aligned}
$$

Just as an application of Rule (RollS) need not be immediately followed by an application of Rule (RIn), an application of Rule (RIn) need not be immediately followed by an application of Rule (ROut). A particular reduction from M_5 undoes the output at $\mathtt{A}$:

$$
\begin{aligned}
M_5 \rightsquigarrow (\nu\, s)(\, & \ell_1[\mathtt{S}] : \wr \mathbf{0} \,;\, s[\mathtt{S}]?(t).s[\mathtt{S}]!\langle price(t)\rangle.s[\mathtt{S}]!\langle price(t)\rangle. \\
& \quad s[\mathtt{S}]?(ok).s[\mathtt{S}]?(a).s[\mathtt{S}]!\langle date\rangle.\mathbf{0} \wr \\
& \mid s_{\mathtt{S}} \lfloor \hat{}\mathtt{A}?\langle \mathsf{title}\rangle.\{\mathtt{A},\mathtt{B}\}!\langle \mathsf{price}\rangle.T_{\mathtt{B}} \cdot x \cdot [x \mapsto d] \rfloor^{\lozenge} \\
& \mid \ell_2[\mathtt{A}] : \wr \mathbf{0} \,;\, s[\mathtt{A}]!\langle \text{‘Logicomix’}\rangle.s[\mathtt{A}]?(p).s[\mathtt{A}]?(s).s[\mathtt{A}]?(ok).\mathbf{0} \wr \\
& \mid s_{\mathtt{A}} \lfloor \hat{}\mathtt{S}!\langle \mathsf{title}\rangle.\mathtt{S}?\langle \mathsf{price}\rangle.\mathtt{B}!\langle \mathsf{share}\rangle.\mathtt{B}?\langle \mathsf{OK}\rangle.\mathsf{end} \cdot y \cdot [y \mapsto d] \rfloor^{\lozenge} \\
& \mid N_4 \mid s : (\epsilon \star \epsilon)) = M_6
\end{aligned}
$$

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

To illustrate abstraction passing, let us assume that M_3 above follows a sequence of forward reductions until the configuration:

$$
\begin{aligned}
M_7 = \; (\nu\, s)(\, & \ell_3[\mathtt{B}] : \wr \mathbf{0} \,;\, s[\mathtt{B}]!\big\langle \{\{ s[\mathtt{B}]!\langle \text{‘Lucca, 55100’}\rangle.s[\mathtt{B}]?(d).\mathbf{0} \}\} \big\rangle.\mathbf{0} \wr \\
& \mid s_{\mathtt{B}} \lfloor \mathbb{T}_7 [\, \hat{}\mathtt{C}!\langle \{\{\diamond\}\}\rangle.\mathtt{S}!\langle \mathsf{address}\rangle.\mathtt{S}?\langle \mathsf{date}\rangle.\mathsf{end}] \cdot z, p, s \cdot \sigma_7 \rfloor^{\lozenge} \\
& \mid \ell_4[\mathtt{C}] : \wr \mathbf{0} \,;\, s[\mathtt{C}]?(code).(code\, *) \wr \\
& \mid s_{\mathtt{C}} \lfloor \mathbb{T}_8 [\, \hat{}\mathtt{B}?\langle \{\{\diamond\}\}\rangle.\mathsf{end}] \cdot w, s \cdot \sigma_8 \rfloor^{\lozenge} \mid N_5 \mid s : (h_7 \star \epsilon))
\end{aligned}
$$

where $120 < price(\text{'Logicomix'})$ is the amount B may contribute and $\mathbb{T}_7[\bullet]$, σ_7, $\mathbb{T}_8[\bullet]$, σ_8, and h_7 capture past interactions as follows:

$$\begin{aligned}\mathbb{T}_7[\bullet] &= \mathtt{S}?\langle\mathsf{price}\rangle.\mathtt{A}?\langle\mathsf{share}\rangle.\{\mathtt{A},\mathtt{S}\}!\langle\mathsf{OK}\rangle.\mathtt{C}!\langle\mathsf{share}\rangle.\bullet\\ \sigma_7 &= [z\mapsto d],[p\mapsto price(\text{'Logicomix'})],[s\mapsto 120]\\ \mathbb{T}_8[\bullet] &= \mathtt{B}?\langle\mathsf{share}\rangle.\bullet \qquad \sigma_8=[w\mapsto d],[s\mapsto 120]\\ h_7 &= (\mathtt{A},\mathtt{S},\text{'Logicomix'})\\ &\circ(\mathtt{S},\mathtt{A},price(\text{'Logicomix'}))\circ(\mathtt{S},\mathtt{B},price(\text{'Logicomix'}))\\ &\circ(\mathtt{A},\mathtt{B},120)\circ(\mathtt{B},\mathtt{A},\text{'ok'})\circ(\mathtt{B},\mathtt{S},\text{'ok'})\circ(\mathtt{B},\mathtt{C},120)\end{aligned}$$

If $M_7 \twoheadrightarrow\ \twoheadrightarrow M_8$ by using Rules (Out) and (In) we would have:

$$\begin{aligned}M_8 = \ &(\nu\, s)(\,\ell_3[\mathtt{B}] : \wr\mathbf{0}\ ;\ \mathbf{0}\wr\\ &\mid s_{\mathtt{B}}\lfloor \mathbb{T}_7[\mathtt{C}!\langle\{\{\diamond\}\}\rangle.\,\hat{}\,\mathtt{S}!\langle\mathsf{address}\rangle.\mathtt{S}?\langle\mathsf{date}\rangle.\mathsf{end}]\cdot z,p,s\cdot\sigma_7\rfloor^{\diamond}\\ &\mid \ell_4[\mathtt{C}] : \wr\mathbf{0}\ ;\ (code\,*)\wr\\ &\mid s_{\mathtt{C}}\lfloor \mathbb{T}_8[\mathtt{B}?\langle\{\{\diamond\}\}\rangle.\,\hat{}\,\mathsf{end}]\cdot w,s,code\cdot\sigma_9\rfloor^{\diamond}\\ &\mid N_5 \mid s : (h_7\circ(\mathtt{B},\mathtt{C},\{\{s[\mathtt{B}]!\langle\text{'Lucca, 55100'}\rangle.s[\mathtt{B}]?(d).\mathbf{0}\}\})\star\epsilon))\end{aligned}$$

where $\sigma_9 = \sigma_8[code \mapsto \{\{s[\mathtt{B}]!\langle\text{'Lucca, 55100'}\rangle.s[\mathtt{B}]?(d).\mathbf{0}\}\}]$. We now may apply Rule (Beta) so as to obtain:

$$\begin{aligned}M_8 \twoheadrightarrow\ &(\nu\, s)(\nu\, k)(\,\ell_4[\mathtt{C}] : \wr\mathbf{0}\ ;\ s[\mathtt{B}]!\langle\text{'Lucca, 55100'}\rangle.s[\mathtt{B}]?(d).\mathbf{0}\wr \mid N_6\\ &\mid k\lfloor(code\,*)\,,\,\ell_4\rfloor \mid s_{\mathtt{C}}\lfloor\mathbb{T}_8[\mathtt{B}?\langle\{\{\diamond\}\}\rangle.k.\,\hat{}\,\mathsf{end}]\cdot w,s,code\cdot\sigma_9\rfloor^{\diamond}\\ &\mid s : (h_7\circ(\mathtt{B},\mathtt{C},\{\{s[\mathtt{B}]!\langle\text{'Lucca, 55100'}\rangle.s[\mathtt{B}]?(d).\mathbf{0}\}\})\star\epsilon)) = M_9\end{aligned}$$

where N_6 is for the rest of the system. Notice that this reduction has added a running function on a fresh k. This fresh k is also used within the type stored in the monitor $s_{\mathtt{C}}$.

The reduction $M_8 \twoheadrightarrow M_9$ completes the code mobility from B to C: the now active thunk will execute B's implementation from C's location. This justifies the premise $\mathtt{p} = \mathtt{r} \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}, h_i)$ present in Rules (Out), (In), (Sel) and (Bra) (and in their backward counterparts): when executing previously received mobile code, the participant mentioned in the location (i.e., C) and that mentioned in the located process (i.e., B) may differ. Further forward reductions from M_9 will modify the cursor in the type stored in monitor $s_{\mathtt{B}}$ based on the process behavior located at $\ell_4[\mathtt{C}]$.

Having introduced our process model and its reversible semantics, we now move on to establish its key properties.

4 Main Results

We now establish our main result: we prove that reversibility in our model of choreographic, asynchronous communication is *causally consistent*. We proceed in three steps:

a) First, we introduce an alternative *atomic* semantics and show that it corresponds, in a tight technical sense, to the decoupled semantics in § 2.2.2 (Theorems 4.1 and 4.2).

b) Second, in the light of this correspondence, we establish causal consistency for the atomic semantics, following the approach of Danos and Krivine [7] (Theorem 4.3).

c) Finally, we state a fine-grained, bidirectional connection between the semantics of (high-level) global types with the decoupled semantics of (low-level) configurations (Theorem 4.4).

As a result of these steps, we may transfer causal consistency to choreographies expressed as global types.

4.1 Atomic Semantics vs. Decoupled Semantics

Our main insight is that causal consistency for asynchronous communication can be established by considering a *coarser* synchronous reduction relation. We define *atomic* versions of the forward and backward reduction relations, relying on the rules in Fig. 7. The *forward atomic reduction*, denoted $\Rrightarrow$, is the smallest evaluation-closed relation that satisfies Rules (AC) and (AS) (Fig. 7), together with Rules (Init), (Beta), and (Spawn) (Fig. 5). Similarly, the *backward atomic reduction*, denoted $\Lleftarrow$, is the smallest evaluation-closed relation that satisfies Rules (RAC) and (RAS) (Fig. 7), together with Rules (RInit), (RBeta), and (RSpawn) (Fig. 6). We then define the atomic reduction relation $\rightarrowtail$ as $\Rrightarrow \cup \Lleftarrow$.

We start by introducing *reachable* configurations:

Definition 4.1. A configuration M is *initial* if $M \equiv (\nu\, \tilde{n}) \prod_i \ell_i \{P_i\}$. A configuration is *reachable*, if it is derived from an initial configuration by using $\longrightarrow$. A configuration is *atomically reachable*, if it is derived from an initial configuration by using $\rightarrowtail$.

To relate the decoupled semantics $\longrightarrow$ (cf. § 2.2.2) with the atomic reduction $\rightarrowtail$ (just defined), we introduce the concept of *stable* configuration. Roughly speaking, in a stable configuration there are no "ongoing" reduction steps. In the forward case, an ongoing step is witnessed by non-empty output queues (which should eventually become empty to complete a synchronization); in the backward case, an ongoing step is witnessed by a marked monitor (which should be eventually unmarked when a synchronization is undone). This way, e.g., in the example of § 3 configurations M_3 and M_7 are stable, whereas M_2 and M_4 are not stable. Reduction $\rightarrowtail$ will move between stable configurations only. We therefore have:

Definition 4.2. A configuration M is *stable*, written $\mathtt{sb}(M)$, if

$$M \equiv \prod_i \ell_i \{P_i\} \mid (\nu\, s\tilde{a})\Big(\prod_j \ell_j[\mathtt{p}_j] : \wr \mathtt{C}_j \,;\, P_j \int \mid s_{\mathtt{p}_i} \lfloor T_i \cdot \widetilde{x}_i \cdot \sigma_i \rfloor^{\Diamond} \mid s : (h_1 \star \epsilon)\Big)$$

Reduction $\longrightarrow$ does not preserve stability, but it can be recovered:

Lemma 4.1. Given M a stable configuration then

- if $M \twoheadrightarrow N$ with $\neg\mathtt{sb}(N)$ then there exists an N' such that $N \twoheadrightarrow N'$ and $\mathtt{sb}(N')$;
- if $M \rightsquigarrow N$ with $\neg\mathtt{sb}(N)$ then there exists an N' such that $N \rightsquigarrow\, \rightsquigarrow N'$ and $\mathtt{sb}(N')$.

We may then have:

Corollary 4.1. If $\mathtt{sb}(M)$ and $M \longrightarrow^* N$ with $\neg\mathtt{sb}(N)$, then there exists an N' such that $N \longrightarrow^* N'$ with $\mathtt{sb}(N')$.

Proof. By induction on the reduction sequence $M \longrightarrow^* N$. □

We now show the Loop lemma [7], which ensures that every reduction step can be reverted. This lemma will be crucial both in proving a correspondence between atomic and decoupled semantics, and in showing causal consistency of the atomic semantics.

Lemma 4.2 (Loop)**.** Let M, N be stable and atomic reachable configurations. Then $M \Rrightarrow N$ if and only if $N \Lleftarrow M$.

Proof. By induction on the derivation of $M \Rrightarrow N$ for the if direction, and on the derivation of $N \Lleftarrow M$ for the converse. □

The following lemma allow us to "rearrange" atomic reduction steps; it will be useful to connect atomic and decoupled reductions.

Lemma 4.3 (Swap)**.** Let M be a reachable configuration, then:

- If $M \twoheadrightarrow^* N_1$ using Rules (Out) or (Sel), and $N_1 \twoheadrightarrow N_2$ by using Rules (In) or (Sel) then $M \twoheadrightarrow \twoheadrightarrow N \twoheadrightarrow^* N_2$, for some N;
- If $M \rightsquigarrow^* N_1$ using Rules (ROut) or (RSel), and $N_1 \rightsquigarrow N_2$ by using Rules (RIn) or (RSel) then $M \rightsquigarrow \rightsquigarrow N \rightsquigarrow^* N_2$, for some N.

The following theorem is a first connection between decoupled and atomic reductions; its proof is immediate from their definitions:

Theorem 4.1 (Relating $\longrightarrow$ and $\rightarrowtail$)**.** Let M and N be stable configurations. We have:

- $M \Rrightarrow N$ if and only if either $M \twoheadrightarrow N$ or $M \twoheadrightarrow \twoheadrightarrow N$;
- $M \Lleftarrow N$ if and only if either $M \rightsquigarrow N$ or $M \rightsquigarrow \rightsquigarrow \rightsquigarrow N$.

We now embark ourselves in providing a tighter formal connection between $\longrightarrow$ and $\rightarrowtail$, using *back-and-forth bisimulations* [15]. We shall work with binary relations on configurations, written $\Re \subseteq \mathcal{M} \times \mathcal{M}$. We now adapt the classical notion of barbs [21] to our setting: rather than communication subjects (which are hidden/unobservable names in intra-session communications), it suffices to use participant identities as observables:

Definition 4.3 (Barbs)**.** A reachable configuration M has a barb $\mathtt{p}$, written $M \downarrow_{\mathtt{p}}$, if

- $M \equiv (\nu\, \tilde{n})(N \mid \ell[\mathtt{r}] : \wr \mathtt{C} \,;\, P \int \mid s_{\mathtt{p}} \lfloor \mathbb{S}[\hat{\ } T] \cdot \widetilde{x} \cdot \sigma \rfloor)$ where either:
  (i) $P \equiv s[\mathtt{p}]!\langle V \rangle.Q \mid R$ and $T = \mathtt{q}!\langle U \rangle.T_1$ or
  (ii) $P \equiv s[\mathtt{p}] \triangleleft \{l_i.P_i\}_{i \in I} \mid R$ and $T = \mathtt{q} \oplus \{l_j : T_j\}_{j \in J}$.

Notice that our definition of barbs is connected to the notion of stability: since in $M \downarrow_{\mathtt{p}}$ we require a monitor with empty tag, this ensures that $\mathtt{p}$ is not involved in an ongoing backward step. In a way, this allows us to consider just *forward barbs* (as in [1]).

We now adapt the definition of weak barbed back-and-forth (bf) bisimulation and congruence [15] in order to work with decoupled and atomic reduction semantics:

Definition 4.4. A relation $\Re$ is a *(weak) barbed bf simulation* if whenever $M \Re N$

1. $M \downarrow_{\mathtt{p}}$ implies $N \longrightarrow^* \downarrow_{\mathtt{p}}$;
2. $M \Rrightarrow M_1$ implies $N \twoheadrightarrow^* N_1$, with $M_1 \Re N_1$;

3. $M \Lleftarrow M_1$ implies $N \rightsquigarrow^* N_1$, with $M_1 \Re N_1$.

A relation $\Re$ is a *(weak) barbed bisimulation* if $\Re$ and $\Re^{-1}$ are weak bf barbed simulations. The largest weak barbed bisimulation is *(weak) barbed bisimilarity*, noted $\approx$. M and N are *(weakly) barbed congruent*, written $\dot{\approx}$, if for each context $\mathbb{C}$ such that $\mathbb{C}[M]$ and $\mathbb{C}[N]$ are atomic reachable configurations, then $\mathbb{C}[M] \approx \mathbb{C}[N]$.

We now may state our second connection between decoupled and atomic reductions:

Theorem 4.2. For any atomic reachable configuration M, we have that $M \dot{\approx} M$.

Proof (Sketch). It suffices to show that the following relation is a bf weak bisimulation:

$$\Re = \{(M, N) \mid M \twoheadrightarrow^* N \text{ via Rules (Out) or (Sel)} \wedge$$
$$M \rightsquigarrow^* N \text{ via Rules (ROut) or (RSel)}\}$$

The analysis uses Corollary 4.1, Theorem 4.1, the Loop and Swap Lemmas (Lemmas 4.2 and 4.3). See App. B.1 for details. □

By observing that the set of atomic configurations is a subset of reachable configurations, this result can also be formulated as full abstraction. Let f be the (injective, identity) mapping from atomic reachable configurations to reachable configurations. We then have:

Corollary 4.2 (Full Abstraction)**.** Let f be the injection from atomic reachable configurations to reachable configurations, and let M, N be two atomic reachable configurations. Then we have $f(M) \dot{\approx} f(N)$ if and only if $M \dot{\approx} N$.

Proof. From Theorem 4.2 we have $M \dot{\approx} f(M)$ and $N \dot{\approx} f(N)$. The thesis follows then by transitivity of $\dot{\approx}$. □

The results above ensure that the loss of atomicity preserves the reachability of configurations yet does not make undesired configurations reachable.

4.2 Causal Consistency

Theorems 4.1 and 4.2 allow us to focus on the atomic reduction $\rightarrowtail$ for the purposes of establishing causal consistency. We adapt the approach of [7] (developed for a reversible CCS) to our higher-order session π-calculus with asynchronous communication. Causal consistency concerns traces of *transitions*:

Definition 4.5. A *transition* t is a triplet of the form $t : M \stackrel{\eta}{\Rightarrow} N$ where $M \rightarrowtail N$ and the *transition stamp* η is defined as follows:

- $\eta = \{\ell_1, \cdots, \ell_n\}$, if Rule (Init) or (RInit) is used;
- $\eta = \{\mathsf{p}, \mathsf{q}\}$, if one of Rules (AC), (AS), (RAC) and (RAS) is used;
- $\eta = \{\ell, \mathsf{p}\}$, if one of Rules (Beta), (Spawn), (RBeta) or (RSpawn) is used.

Given $t : M \overset{\eta}{\Rightarrow} N$, we say M and N are its source and target (written $\mathtt{src}(t)$ and $\mathtt{trg}(t)$), respectively. A transition $t : M \overset{\eta}{\Rightarrow} N$ is *forward* if $M \Rrightarrow N$ and *backward* if $M \Lleftarrow N$. Given $t : M \overset{\eta}{\Rightarrow} N$, its *inverse*, denoted $t_\bullet$, is the transition $t_\bullet : N \overset{\eta}{\Rightarrow} M$. Two transitions are *coinitial* if they have the same source; *cofinal* if they have the same target; *composable* if the target of the first one is the source of the other. Given coinitial transitions $t_1 : M \overset{\eta_1}{\Longrightarrow} N_1$ and $t_2 : M \overset{\eta_2}{\Longrightarrow} N_2$, we define t_2/t_1 (read "t_2 after t_1") as $N_1 \overset{\eta_2}{\Longrightarrow} N_2$, i.e., the transition with stamp η_2 that starts from the target of t_1. A *trace* is a sequence of pairwise composable transitions. We let t and ρ range over transitions and traces, respectively. Notions of target, source, composability and inverse extend naturally to traces. We write ε_M to denote the empty trace with source M, and $\rho_1; \rho_2$ to denote the composition of two composable traces ρ_1 and ρ_2. Two important classes of transitions are *conflicting* and *concurrent* ones:

Definition 4.6. Two coinitial transitions $t_1 : M \overset{\eta_1}{\Longrightarrow} M_1$ and $t_2 : M \overset{\eta_2}{\Longrightarrow} M_2$ are said to be in *conflict* if $\eta_1 \cap \eta_2 \neq 0$. Two transitions are *concurrent* if they are not in conflict.

A property that a reversible semantics should enjoy is the so-called Square Lemma [7], which may be informally described as follows. Assume a configuration from which two transitions are possible: if these transitions are concurrent then the order in which they are executed does not matter, and the same configuration is reached.

Lemma 4.4 (Square)**.** If $t_1 : M \overset{\eta_1}{\Longrightarrow} M_1$ and $t_2 : M \overset{\eta_2}{\Longrightarrow} M_2$ are coinitial and concurrent transitions, then there exist cofinal transitions $t_2/t_1 = M_1 \overset{\eta_2}{\Longrightarrow} N$ and $t_1/t_2 = M_2 \overset{\eta_1}{\Longrightarrow} N$.

Definition 4.7. We define $\asymp$ as the least equivalence between traces that is closed under composition and that obeys: i) $t_1; t_2/t_1 \asymp t_2; t_1/t_2$; ii) $t; t_\bullet \asymp \varepsilon_{\mathtt{src}(t)}$; iii) $t_\bullet; t \asymp \varepsilon_{\mathtt{trg}(t)}$.

Intuitively, $\asymp$ says that: (a) given two concurrent transitions, the traces obtained by swapping their execution order are equivalent; (b) a trace consisting of opposing transitions is equivalent to the empty trace. The proof of causal consistency follows that in [7], but with simpler arguments because of our simpler transition stamps. The following lemma says that, up to causal equivalence, traces can be rearranged so as to reach the maximum freedom of choice, first going only backwards, and then going only forward.

Lemma 4.5 (Rearranging)**.** Let ρ be a trace. There are forward traces ρ', ρ'' such that $\rho \asymp {\rho'}_\bullet; \rho''$.

Proof. By lexicographic induction on the length of ρ and on the distance between the beginning of ρ and the earliest pair of opposing transitions in ρ. The analysis uses both the Loop Lemma (Lemma 4.2) and the Square Lemma (Lemma 4.4). □

If trace ρ_1 and forward trace ρ_2 start from the same configuration and end up in the same configuration, then ρ_1 may contain some "local steps", not present in ρ_2, which must be eventually reversed—otherwise there would be a difference with respect to ρ_2. Hence, ρ_1 could be *shortened* by removing such local steps and their corresponding reverse steps.

Lemma 4.6 (Shortening)**.** Let ρ_1, ρ_2 be coinitial and cofinal traces, with ρ_2 forward. Then, there exists a forward trace ρ_1' of length at most that of ρ_1 such that $\rho_1' \asymp \rho_1$.

Proof. By induction on the length of ρ_1, using Square and Rearranging Lemmas (Lemmas 4.4, 4.5). The proof uses the forward trace ρ_2 as guideline for shortening ρ_1 into a forward trace, relying on the fact that ρ_1, ρ_2 share the same source and target. □

We may now state our main result:

Theorem 4.3 (Causal consistency)**.** Let ρ_1 and ρ_2 be coinitial traces, then $\rho_1 \asymp \rho_2$ if and only if ρ_1 and ρ_2 are cofinal.

Proof. The ‘if’ direction follows by definition of $\asymp$ and trace composition. The ‘only if’ direction uses Square, Rearranging and Shortening Lemmas (Lemmas 4.4, 4.5, 4.6). □

4.3 Connecting (Reversible) Choreographies and (Reversible) Configurations

We now relate choreographies and configurations to connect the two levels of abstraction for reversible global protocols. For convenience, we focus on first-order global types (i.e., without abstraction passing), relying on a simple characterization of the *well-formed processes* that implement a given local type. We write $P \bowtie_x T$ to denote that P implements the local type T along variable x—see App. B.2 for a definition. We may then define the configurations that implement a global type with history. First, an auxiliary definition:

Definition 4.8. We say the global type with history H is *reachable* if it can be obtained from a global type G via a sequence of $\hookrightarrow$ and $\rightharpoonup$ transitions (cf. Fig. 2).

Definition 4.9. Let G be a global type, with $\mathtt{pa}(G) = \{\mathtt{p}_1, \cdots, \mathtt{p}_n\}$. We say that configuration M *initially implements* G if we have

$$M \equiv (\nu\, s)\Big(\prod_{i\in\{1,\cdots,n\}} \ell_i[\mathtt{p}_i] : \wr \mathbf{0} \,;\, P_i\{s[\mathtt{p}_i]/x_i\} \wr \;\mid\; s_{\mathtt{p}_i} \lfloor \hat{}\, G{\downarrow_{\mathtt{p}_i}} \cdot x_i \cdot \sigma_i \rfloor \mid s : (\epsilon \star \epsilon)\Big)$$

with $P_i \bowtie_{x_i} G{\downarrow_{\mathtt{p}_i}}$, for all $i \in \{1,\cdots,n\}$, for some stores $\sigma_1, \ldots, \sigma_n$. A configuration N *implements* the global type with history H, written $N \bowtie \mathsf{H}$, if there exist M, G such that (i) H is reachable from G, (ii) M initially implements G, and (iii) N is reachable from M.

The last ingredient required is a *swapping relation* over global types, denoted $\approx_{\mathtt{sw}}$, which enables behavior-preserving transformations among causally independent communications.

Definition 4.10 (Swapping)**.** We define $\approx_{\mathtt{sw}}$ as the smallest congruence on G that satisfies the rules in Fig. 8 (where we omit the symmetric of (Sw1), (Sw2), and (Sw3)). We extend $\approx_{\mathtt{sw}}$ to global types with history H as follows: $\mathbb{G}[\hat{}\, G_1] \approx_{\mathtt{sw}} \mathbb{G}'[\hat{}\, G_2]$ if $\mathbb{G}[\mathtt{end}] \approx_{\mathtt{sw}} \mathbb{G}'[\mathtt{end}]$ and $G_1 \approx_{\mathtt{sw}} G_2$.

We may now relate (i) transitions in the semantics of (high-level) global types (with history) with (ii) reductions in the semantics of their (low-level) process implementations. We write $M \rightsquigarrow^j M'$ to denote a sequence of $j \geq 0$ reduction steps (if $j = 0$ then $M = M'$).

Theorem 4.4. Let H be a reachable, first-order global type with history (cf. § 2.1.2).

a) If $M \bowtie \mathsf{H}$ and $\mathsf{H} \hookrightarrow \mathsf{H}'$ then $M \twoheadrightarrow M'$ and $M' \bowtie \mathsf{H}'$, for some M'. If $M \bowtie \mathsf{H}$ and $\mathsf{H} \rightharpoonup \mathsf{H}'$ then $M \rightsquigarrow^j M'$ (with $j = 1$ or $j = 2$) and $M' \bowtie \mathsf{H}'$, for some M'.

b) Suppose $M \bowtie \mathsf{H}$. For all configurations N_i such that $M \twoheadrightarrow N_i$ there exist $\mathsf{H}_i, \mathsf{H}'_i, \mathsf{H}''$, and M', such that $\mathsf{H} \approx_{\mathtt{sw}} \mathsf{H}_i \hookrightarrow \mathsf{H}'_i$, $N_i \bowtie \mathsf{H}'_i$, $N_i \twoheadrightarrow^* M'$, $\mathsf{H}'_i \hookrightarrow^* \mathsf{H}''$, and $M' \bowtie \mathsf{H}''$ (and similarly for $\rightsquigarrow$, $\rightharpoonup$).

Proof. By induction on the transitions/reductions. See Appendix B.2 for details. □

Theorem 4.4 captures an asymmetry between global types and configurations. While Part (a) shows that a configuration closely mimics the behavior of its associated global type, Part (b) shows that a configuration may have more immediate behaviors than those described its associated global type: this is because a configuration may include several independent (and immediate) reductions (N_i above), which are matched by the global type only up to swapping.

Summing up, we have that Theorem 4.3 ensures that reversibility in the atomic semantics is causally consistent. Theorem 4.2 transfers this result to decoupled semantics; since by Theorem 4.4 decoupled semantics defines a sound local implementation, we conclude that reversibility for global types is also causally consistent.

5 Related Work

Reversibility in concurrency has received much attention recently. A detailed overview of the literature on the intersection between reversibility and behavioral contracts/types appears in [18, § 7]. Within this research line, the works most related to ours are [23, 8]. Tiezzi and Yoshida [23] study the cost of implementing different ways of reversing binary and multiparty sessions; since they work in a *synchronous* setting, these alternatives are simpler or incomparable to our asynchronous, decoupled rollback. Dezani-Ciancaglini and Giannini [8] develop typed multiparty sessions with *checkpoints*, points in the global protocol to which computation may return. While our reversible actions are embedded in/guaranteed by the semantics, rollbacks in [8] should specify the name of the checkpoint to which computation should revert. Defining reversibility in [8] requires modifying both processes and types. In contrast, we consider standard untyped processes and local types (with cursors) as monitors. While we show causal consistency with a direct proof, in [8] causal consistency follows indirectly, as a consequence of typing. Reversibility in our model is fine-grained in that we allow reversible actions concerning exactly two of the protocol participants; in [8] when a checkpoint is taken, also parties not related with that choice are forced to return to a checkpoint.

6 Concluding Remarks

We presented a process framework of reversible, multiparty asynchronous communication, built upon session-based concurrency. As illustrated in § 3, the distinguishing features of our framework (decoupled rollbacks and abstraction passing, including delegation) endow it with substantial expressiveness, improving on prior works.

Our processes/configurations are untyped, but their (reversible) behavior is governed by monitors derived from local (session) types. In our view, our monitored approach to reversibility is particularly appropriate for specifying and reasoning about systems with components whose behavior may not be statically analyzed (e.g., legacy components or services available as black-boxes). A monitored approach is general enough to support also the analysis of reversible systems that combine typed and untyped components.

We proved that our reversible semantics is *causally consistent*, which ensures that reversing a computation leads to a state that could have been reached by performing only forward steps. The proof is challenging (and, in our view, also interesting), as we must resort to an alternative atomic semantics for rollbacks (Fig. 7). We then connected reversibility at the level of process/configurations with reversibility at the level of global types, therefore linking the operational and declarative levels of abstraction typical of choreographic approaches to correctness for communication-centric software systems.

Extensions and Future Work As already mentioned, our framework does not include name passing, which is known to be representable, in a fully abstract way, using name abstractions [13]. Primitive support for name passing is not difficult, but would entail notational burden. We do not foresee difficulties to strengthen Theorem 4.4 to cover global types with higher-order values. Such an extension would entail replacing $P \bowtie_x T$ with a type system for multiparty, higher-order sessions, which could be obtained by adapting known type systems for binary, higher-order sessions [19, 13]. These extensions (name passing, typability) would allow us to relate our framework with known typed frameworks for monitored networks (without reversibility) based on multiparty sessions [3].

In future work, we plan to extend our framework with so-called *reversibility modes* [18], which implement *controlled reversibility* by specifying how many times a particular protocol step can be reversed—zero, one, or infinite times. (Currently all actions can be reversed infinite times.) In a related vein, we plan to explore variants of our model in which certain protocol branches are "forgotten" after they have been reversed; this modification is delicate, because it would weaken the notion of causal consistency.

References

[1] C. Aubert and I. Cristescu. Contextual equivalences in configuration structures and reversibility. *J. Log. Algebr. Meth. Program.*, 86(1):77–106, 2017.

[2] F. Barbanera, M. Dezani-Ciancaglini, and U. de'Liguoro. Reversible client/server interactions. *Formal Asp. Comput.*, 28(4):697–722, 2016.

[3] L. Bocchi, T. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. *Theor. Comput. Sci.*, 669:33–58, 2017.

[4] I. Castellani, M. Dezani-Ciancaglini, and J. A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Asp. Comput.*, 28(4):669–696, 2016.

[5] M. Coppo, M. Dezani-Ciancaglini, L. Padovani, and N. Yoshida. A gentle introduction to multiparty asynchronous session types. In M. Bernardo and E. B. Johnsen, editors, *Formal Methods for Multicore Programming*, volume 9104 of *LNCS*, pages 146–178. Springer, 2015.

[6] V. Danos and J. Krivine. Reversible communicating systems. In P. Gardner and N. Yoshida, editors, *CONCUR 2004*, LNCS, pages 292–307. Springer, 2004.

[7] V. Danos and J. Krivine. Transactions in RCCS. In M. Abadi and L. de Alfaro, editors, *CONCUR 2005*, pages 398–412, 2005.

[8] M. Dezani-Ciancaglini and P. Giannini. Reversible multiparty sessions with checkpoints. In D. Gebler and K. Peters, editors, *EXPRESS/SOS 2016*, volume 222 of *EPTCS*, pages 60–74, 2016.

[9] M. Hennessy. *A Distributed Pi-Calculus.* CUP, 2007.

[10] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

[11] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.

[12] H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

[13] D. Kouzapas, J. A. Pérez, and N. Yoshida. On the relative expressiveness of higher-order session processes. In P. Thiemann, editor, *ESOP 2016*, pages 446–475, 2016.

[14] D. Kouzapas, N. Yoshida, R. Hu, and K. Honda. On asynchronous eventful session semantics. *Mathematical Structures in Computer Science*, 26(2):303–364, 2016.

[15] I. Lanese, C. A. Mezzina, and J. Stefani. Reversibility in the higher-order π-calculus. *Theor. Comput. Sci.*, 625:25–84, 2016.

[16] I. Lanese, C. A. Mezzina, and F. Tiezzi. Causal-consistent reversibility. *Bulletin of the EATCS*, 114, 2014.

[17] C. A. Mezzina and J. A. Pérez. Reversible sessions using monitors. In D. A. Orchard and N. Yoshida, editors, *PLACES 2016*, volume 211 of *EPTCS*, pages 56–64, 2016.

[18] C. A. Mezzina and J. A. Pérez. Reversibility in session-based concurrency: A fresh look. *J. Log. Algebr. Meth. Program.*, 2017. To appear. Available in `http://jperez.nl`.

[19] D. Mostrous and N. Yoshida. Two session typing systems for higher-order mobile processes. In *TLCA*, volume 4583 of *LNCS*, pages 321–335. Springer, 2007.

[20] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[21] D. Sangiorgi and D. Walker. On barbed equivalences in pi-calculus. In K. G. Larsen and M. Nielsen, editors, *CONCUR 2001*, volume 2154 of *LNCS*, pages 292–304. Springer, 2001.

[22] F. Tiezzi and N. Yoshida. Reversible session-based pi-calculus. *J. Log. Algebr. Meth. Program.*, 84(5):684–707, 2015.

[23] F. Tiezzi and N. Yoshida. Reversing single sessions. In S. J. Devitt and I. Lanese, editors, *Reversible Computation - 8th International Conference, RC 2016*, volume 9720 of *LNCS*, pages 52–69. Springer, 2016.

A Additional Examples

A.1 A Reversible Protocol with Choices

We use a simple binary (two-party) protocol between a Buyer (B) and a Seller (S) to further illustrate our process framework, in particular to showcase reversibility of labeled choices. Consider the following global type:

$$G = \ \mathtt{B} \to \mathtt{S} : \langle \mathsf{title} \rangle . \mathtt{S} \to \mathtt{B} : \langle \mathsf{price} \rangle .$$
$$\mathtt{S} \to \mathtt{B}\{\mathbf{ok} : \mathtt{B} \to \mathtt{S} : \langle \mathsf{addr} \rangle . \mathtt{S} \to \mathtt{B} : \langle \mathsf{date} \rangle . \mathtt{end} \,; \mathbf{quit} : \mathtt{end}\}$$

This way, after receiving a title from Buyer, Seller replies with the price of the requested item; subsequently, a choice indicated by labels **ok** and **quit** takes place: Buyer can select whether to continue with the transaction or to conclude it. The projection of G onto local types are:

$$\begin{aligned} G\!\downarrow_{\mathtt{S}} \ &= \ \mathtt{B}?\langle \mathsf{title} \rangle . \mathtt{B}!\langle \mathsf{price} \rangle . \\ &\quad\ \mathtt{B}\&\{\mathbf{ok} : \mathtt{B}?\langle \mathsf{addr} \rangle . \mathtt{B}!\langle \mathsf{date} \rangle . \mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\} \\ G\!\downarrow_{\mathtt{B}} \ &= \ \mathtt{S}!\langle \mathsf{title} \rangle . \mathtt{S}?\langle \mathsf{price} \rangle . \\ &\quad\ \mathtt{S} \oplus \{\mathbf{ok} : \mathtt{S}!\langle \mathsf{addr} \rangle . \mathtt{S}?\langle \mathsf{date} \rangle . \mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\} \end{aligned}$$

Possible implementations for the participants are as follows:

$$\begin{aligned} \text{Seller} &= a!\langle x : G\!\downarrow_{\mathtt{S}} \rangle . x?(\mathit{title}) . x!\langle \mathit{quote} \rangle . \\ &\qquad x \triangleright \{\mathbf{ok} : x?(\mathit{addr}) . x!\langle \mathit{date} \rangle . \,;\, \mathbf{quit} : \mathbf{0}\} \\ \text{Buyer} &= a?(y : G\!\downarrow_{\mathtt{B}}) . y!\langle \mathit{title} \rangle . y?(\mathit{quote}) . \\ &\qquad y \triangleleft \{\mathbf{ok} : y!\langle \mathit{addr} \rangle . y?(\mathit{date}) . \,;\, \mathbf{quit} : \mathbf{0}\} \end{aligned}$$

The whole system, given by configuration M below, is obtained by placing these process implementations in appropriate locations:

$$M = \ell_1 \{\text{Seller}\} \mid \ell_2 \{\text{Buyer}\}$$

We then may have:

$$\begin{aligned} &M \twoheadrightarrow^* \\ &(\nu\, s)(\, \ell_1[\mathtt{S}] : \wr \mathbf{0} \,;\, s[\mathtt{S}] \triangleright \{\mathbf{ok} : s[\mathtt{S}]?(\mathit{addr}) . s[\mathtt{S}]!\langle \mathit{date} \rangle . \mathbf{0} \,;\, \mathbf{quit} : \mathbf{0}\} \int \ \mid \\ &s_{\mathtt{S}} \lfloor \mathbb{T}_1[\, \hat{}\mathtt{B}\&\{\mathbf{ok} : \mathtt{B}?\langle \mathsf{addr} \rangle . \mathtt{B}!\langle \mathsf{date} \rangle . \mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\}] \cdot x_1 \cdot \sigma_1 \rfloor^{\diamondsuit} \mid \\ &\ell_2[\mathtt{B}] : \wr \mathbf{0} \,;\, s[\mathtt{B}] \triangleleft \{\mathbf{ok} : s[\mathtt{B}]?(\mathit{addr}) . s[\mathtt{B}]!\langle \mathit{date} \rangle . \mathbf{0} \,;\, \mathbf{quit} : \mathbf{0}\} \int \ \mid \\ &s_{\mathtt{B}} \lfloor \mathbb{S}_1[\, \hat{}\mathtt{S} \oplus \{\mathbf{ok} : \mathtt{S}!\langle \mathsf{addr} \rangle . \mathtt{S}?\langle \mathsf{date} \rangle . \mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\}] \cdot x_2 \cdot \sigma_2 \rfloor^{\diamondsuit} \mid \\ &s : (h_1 \star \epsilon)\,) = M_1 \end{aligned}$$

where M_1 is the configuration obtained from M once the two participants have initiated the session and exchanged the title and the corresponding price. Above, x_1 and x_2 are the free variables of S and B after the first three interactions; also, σ_1 and σ_2 represent their respective stores. Queue h_1 contains the two messages related to *title* and *price*. The context types are:

$$\mathbb{T}_1\, [\bullet] = \mathtt{B}?\langle \mathsf{title} \rangle . \mathtt{B}!\langle \mathsf{price} \rangle . \bullet \qquad \mathbb{S}_1\, [\bullet] = \mathtt{S}!\langle \mathsf{title} \rangle . \mathtt{S}?\langle \mathsf{price} \rangle . \bullet$$

In M_1, Buyer can decide either (a) to accept the suggested price and continue with the prescribed protocol or (b) to refuse it and exit. The first possibility may proceed using Rule (SEL) as follows:

$$
\begin{aligned}
&M_1 \twoheadrightarrow \\
&(\nu\, s)(\, \ell_2[\mathtt{B}] : \wr \mathbf{0}, s[\mathtt{B}] \triangleleft \{\mathbf{quit} : \mathbf{0}\} \,;\, s[\mathtt{B}]?(addr).s[\mathtt{B}]!\langle date\rangle.\mathbf{0} \int \;| \\
&s_{\mathtt{B}} \lfloor \mathbb{S}_1[\mathtt{S} \oplus \{\mathbf{ok} : \hat{\ }\mathtt{S}!\langle \mathsf{addr}\rangle.\mathtt{S}?\langle \mathsf{date}\rangle.\mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\}] \cdot x_2 \cdot \sigma_2 \rfloor^{\Diamond} \;| \\
&s : (h_1 \star (\mathtt{B}\,,\, \mathtt{S}\,,\, \mathbf{ok})) \,|\, N_1\,) = M_2
\end{aligned}
$$

where N_1 contains the rest of the Seller process and monitor of M_1. As we can see, in M_2 the cursor $\hat{\ }$ of the Buyer monitor has been moved into the choice. Moreover. the process stack of Buyer is updated in order to register the discarded branch of the choice (i.e., the branch involving label **quit**). From M_2, Seller can consume the message on top of the queue (which details the choice by B), or the Buyer can revert its choice. In the first case we have the following, using Rule (BRA):

$$
\begin{aligned}
&M_2 \twoheadrightarrow \\
&(\nu\, s)(\, \ell_1[\mathtt{S}] : \wr \mathbf{0}, s[\mathtt{S}] \triangleright \{\mathbf{quit} : \mathbf{0}\} \,;\, s[\mathtt{S}]?(addr).s[\mathtt{S}]!\langle date\rangle.\mathbf{0} \int \;| \\
&s_{\mathtt{S}} \lfloor \mathbb{T}_1[\mathtt{B}\&\{\mathbf{ok} : \hat{\ }\mathtt{B}?\langle \mathsf{addr}\rangle.\mathtt{B}!\langle \mathsf{date}\rangle.\mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\}] \cdot x_1 \cdot \sigma_1 \rfloor^{\Diamond} \;| \\
&\ell_2[\mathtt{B}] : \wr \mathbf{0}, s[\mathtt{B}] \triangleleft \{\mathbf{quit} : \mathbf{0}\} \,;\, s[\mathtt{B}]?(addr).s[\mathtt{B}]!\langle date\rangle.\mathbf{0} \int \;| \\
&s_{\mathtt{B}} \lfloor \mathbb{S}_1[\mathtt{S} \oplus \{\mathbf{ok} : \hat{\ }\mathtt{S}!\langle \mathsf{addr}\rangle.\mathtt{S}?\langle \mathsf{date}\rangle.\mathsf{end} \,;\, \mathbf{quit} : \mathsf{end}\}] \cdot x_2 \cdot \sigma_2 \rfloor^{\Diamond} \;| \\
&s : (h_1 \circ (\mathtt{B}\,,\, \mathtt{S}\,,\, \mathbf{ok}) \star \epsilon)\,) = M_3
\end{aligned}
$$

In the second case, we can revert the labeled choice by using Rule (ROLLC) from M_3 first, and then using Rules (RBRA) and (RSEL) in a decoupled fashion.

B Omitted Proofs

B.1 Proof of Theorem 4.2

We repeat the statement in Page 17:

Theorem 4.2. *For any atomic reachable configuration M, we have that $M \dot{\approx} M$.*

Proof. First, notice that showing $\mathbb{C}[M] \approx \mathbb{C}[M]$ is similar to show $M_1 \approx M_1$ with $M_1 = \mathbb{C}[M]$. This allows us to just focus on the "hole" of the context. It is then sufficient to show that the following relation is a bf weak bisimulation.

$$
\begin{aligned}
\Re = \{(M, N) \mid M \twoheadrightarrow^* N \text{ via Rules OUT or SEL} \wedge \\
M \rightsquigarrow^* N \text{ via Rules ROUT or RSEL}\}
\end{aligned}
$$

Clearly, $(M, M) \in \Re$. We consider the requirements in Def. 4.4.

Let us first consider barbs. Suppose that M challenges N with a barb, we distinguish two cases: N is stable or not. If $\mathtt{sb}(N)$ then N has the same barb. Otherwise, if $\neg\mathtt{sb}(N)$, by Corollary 4.1 there exists an N_1 such that $N \longrightarrow^* N_1$ and $\mathtt{sb}(N_1)$. Since $M \longrightarrow^* N$ we may derive $M \longrightarrow^* N_1$ with both stable configurations. By applying Theorem 4.1 on $M \longrightarrow^* N_1$ we infer $M \rightarrowtail^* N_1$; then, by applying the Loop Lemma (Lemma 4.2) we further infer $N_1 \rightarrowtail^* M$. Using again Theorem 4.1 we infer that $N_1 \longrightarrow^* M$; since we have deduced that $N \longrightarrow^* N' \longrightarrow^* M$, we know that N weakly matches all the barbs of M, as desired. Suppose now that N challenges M with a barb. We proceed similarly as above: if $\mathtt{sb}(M)$ then M has the same barb; otherwise, if $\neg\mathtt{sb}(M)$, since $M \longrightarrow^* N$, by Corollary 4.1 we have that $M \longrightarrow^* N \longrightarrow^* N_1$, with $\mathtt{sb}(N_1)$. Let us note that the reductions in $N \longrightarrow^* N_1$ do not add barbs to N_1: they only finalize ongoing synchronizations; by definition of barbs (Def. 4.3) parties involved in ongoing rollbacks do not contribute to barbs. We can conclude by applying Theorem 4.1 and deriving $M \rightarrowtail^* N_1$, which has the same barbs of N, as desired.

Let us now consider reductions. We will just focus on synchronizations due to input/output and branching/selection reduction steps, since these are the cases in which $\longrightarrow$ and $\rightarrowtail$ differ; indeed, reductions due to Rules Spawn and Beta can be trivially matched. There are two cases: $M \Rrightarrow M_1$ and $M \Lleftarrow M_1$. In the first case, as we distinguish two sub-cases: either N has already started the synchronization or not. In the first case, N can conclude the step: $N \twoheadrightarrow N'$. Now we have that $M \twoheadrightarrow^* N \twoheadrightarrow N'$. Thanks to Lemma 4.3 we can rearrange such a reduction sequence as follows: $M \twoheadrightarrow \twoheadrightarrow M_1 \twoheadrightarrow^* N'$. We then have that the pair $(M_1, N') \in \Re$, as desired. In the second case, N can match the step with 2 reductions: $N \twoheadrightarrow \twoheadrightarrow N'$. Also in this case we can rearrange the reduction sequence so as to obtain $M \twoheadrightarrow \twoheadrightarrow M_1 \twoheadrightarrow {}^* N'$, with $(M_1, N') \in \Re$, as desired. The second case is when $M \Lleftarrow M_1$ (i.e., the challenge is a backward move) and is handled similarly.

We now consider challenges from N, focusing only on synchronizations, just as before. If $N \twoheadrightarrow N'$, we distinguish two cases: whether the reduction finalizes ends an ongoing input/output and branching/selection, or it opens a new one. In the second case M matches the move with an idle move, i.e., $(M, N') \in \Re$. In the other case we can rearrange the reduction $M \twoheadrightarrow^* N \twoheadrightarrow N'$ into a similar reduction sequence $M \longrightarrow^* N_1 \twoheadrightarrow^* N'$ with $\mathtt{sb}(N_1)$, and all reductions in $N_1 \twoheadrightarrow^* N'$ just start new synchronizations. Thanks to Theorem 4.1, M can mimick the same reduction to N_1, i.e., $M \Rrightarrow^* N_1$, and we have that $(N_1, N') \in \Re$, as desired. The case in which $N \rightsquigarrow N'$ (i.e., the challenge is a backward move) is similar. This concludes the proof. $\square$

B.2 Appendix to § 4.3

Well-formed Process Implementations of a Local Type The results in § 4.3 rely on *well-formedness* of a process P with respect to a local type T. Figure 9 reports a very simple system for decreeing well-formedness. It is inspired by the type system for processes defined in [4]; clearly, more sophisticated systems, such as variants of those in [19, 13], can be considered. We rely on two judgments:

- $\vdash V :: U$ says that V is a well-formed (i.e. valid) value of type V.
- $\Gamma \vdash P :: x : T$ says that P implements a single session with local type T along variable x, relying on type assignments for name variables and recursive variables declared in Γ.

We will write $P \bowtie_x T$ if and only iff $\emptyset \vdash P :: x : T$. This notion of well-formedness can be extended to relate configurations and local types with history; see Figure 10.

B.3 Proof of Theorem 4.4

We write $\mathsf{pa}(\mathsf{H})$ to denote the set of participants in a global type with history H. The following proposition details the shape of configurations that are reachable from a configuration that initially implements a global type with history. In particular, monitor tags can be full $\blacklozenge$ or empty $\lozenge$, and so if $M \bowtie \mathsf{H}$ then M may not be stable:

Proposition B.1. Let $M \bowtie \mathsf{H}$ with $\mathsf{pa}(\mathsf{H}) = \{\mathtt{p}_1, \cdots, \mathtt{p}_n\}$. Then we have

$$M \equiv (\nu\, s, \widetilde{n})\Big(\prod_{i \in \{1,\cdots,n\}} \ell_i[\mathtt{p}_i] : \wr \mathtt{C}_i \,;\, Q_i \int \mid s_{\mathtt{p}_i} \lfloor \mathbb{T}_i \,[\,\hat{}S_i] \cdot \widetilde{x}_i \cdot \sigma_i \rfloor^{\spadesuit} \mid s : (h_1^i \star h_2^i)\Big)$$

with $\wr \mathtt{C}_i \,;\, Q_i \int \bowtie_{s[\mathtt{p}_i]} \mathbb{T}_i\,[\,\hat{}S_i]$ (cf. Figure 10), for all $i \in \{1,\cdots,n\}$, for some stores $\sigma_1, \ldots, \sigma_n$, process stacks $\mathtt{C}_1, \ldots, \mathtt{C}_n$. type contexts $\mathbb{T}_1, \ldots, \mathbb{T}_n$, local types $S_1, \ldots, S_n$, variables $\widetilde{x_1}, \ldots, \widetilde{x_n}$, and queues $h_1^1, h_2^1, \ldots, h_1^n, h_2^n$.

Proof. Immediate from Definition 4.1 (reachable configuration), Definition 4.9 ("initially implements"), and the reduction semantics $\longrightarrow$. □

The following congruence allows us to (silently) modify the order of the messages in the queue in a consistent way:

Definition B.1 (Equivalence on message queues)**.** We define the structural equivalence on queues, denoted $\equiv_{\mathtt{q}}$, as follows:

$$h \circ (\mathtt{p}_1\,,\, \mathtt{q}_1\,,\, m_1) \circ (\mathtt{p}_2\,,\, \mathtt{q}_2\,,\, m_2) \circ h' \equiv_{\mathtt{q}} h \circ (\mathtt{p}_2\,,\, \mathtt{q}_2\,,\, m_2) \circ (\mathtt{p}_1\,,\, \mathtt{q}_1\,,\, m_1) \circ h'$$

whenever $\mathtt{p}_1 \neq \mathtt{p}_2 \wedge \mathtt{q}_1 \neq \mathtt{q}_2$. Equivalence $\equiv_{\mathtt{q}}$ extends to configurations as expected.

We may now have:

Proof of Theorem 4.4 (Sketch). The proof of Part (a) proceeds by induction on the transitions $\mathsf{H} \hookrightarrow \mathsf{H}'$ and $\mathsf{H} \rightharpoonup \mathsf{H}'$, with a case analysis on the last applied rule. For the forward case we have a one-to-one correspondence; there are four possible transitions at the level of global types: a transition derived using Rule (FVal1) is matched by M using Rule (Out); a transition derived using Rule (FVal2) is matched by M using Rule (In); a transition derived using Rule (FCho1) is matched by M using Rule (Sel); a transition derived using Rule (FCho2) is matched by M using Rule (Bra). The analysis for the backward case is similar, but before matching the transition $\mathsf{H} \rightharpoonup \mathsf{H}'$, we may require an additional reduction step from M, depending on the tag of the corresponding monitor. We use Prop. B.1 to determine this tag. If the tag is $\lozenge$, then a reduction (using Rule (RollS) or (RollC)) is required in order to have a full tag $\blacklozenge$, as needed by all relevant backward reduction rules ($j = 2$). Otherwise, if the tag is already $\blacklozenge$ from a previous reduction then no additional step is needed ($j = 1$). Once the tag is full, the transition is matched as follows: a transition derived using Rule (BVal1) is matched by M using Rule (ROut); a transition derived using Rule (BVal2) is matched by M using Rule (RIn); a transition derived using Rule (BCho1) is matched by M using Rule (RSel); a transition derived using Rule (BCho2) is matched by M using Rule (RBra).

The proof of Part (b) proceeds by induction on transitions $M \twoheadrightarrow N$ and $M \rightsquigarrow N$, with a case analysis on the last applied rule, following similar lines. There are two main cases:

(i) There is exactly one reduction from M involving the participants that appear at the top-level in H and can evolve.

(ii) There are one or more reductions from M to N_i whose involved participants cannot be found at the top-level in H.

While in case (i) the proof follows the analysis for Part (a), in case (ii) we use $\approx_{\mathtt{sw}}$ on H to obtain behavior-preserving transformations H_i of H in which the participants involved in the reductions ($M \twoheadrightarrow N_i$ or $M \rightsquigarrow N_i$) appear at the top-level. Such transformations exist, because of assumption $M \bowtie \mathsf{H}$. This way, reductions from M can be matched by H up to swapping; after all the independent communication actions have been performed and matched (there are finitely many of them), one obtains M', H such that $M' \bowtie \mathsf{H}''$.

In both (i) and (ii), the analysis of a backwards reduction from M relies on Prop. B.1 to determine the relevant tag involved. If the tag in M is $\blacklozenge$ then a reduction $M \rightsquigarrow N'$ was derived using Rules (ROut), (RIn), (RSel), or (RBra) and it is easy to show that it corresponds directly to one transition from H ($j = 0$). When the reduction is derived using Rules (RollS) or (RollC) (because the corresponding tag in M is $\lozenge$), there is no corresponding transition from H and an extra reduction (using Rules (ROut), (RIn), (RSel), or (RBra), which become enabled thanks to Rules (RollS) and (RollC)) is required to actually match the transition ($j = 1$). It is worth noticing that due to the simplicity of our well-formed processes/configurations (and our focus on first-order global types), reduction $M \twoheadrightarrow N$ does not involve Rule (Spawn) (for well-formed processes are single-threaded) nor Rule (Beta) (for $(V\ w)$ is a well-formed value but not a well-formed process). Similarly, $M \rightsquigarrow N$ does not involve Rules (RSpawn) and (RBeta). □

$$\text{(RInit)}\ \frac{\mathtt{pa}(G)=\{\mathtt{p}_1,\cdots,\mathtt{p}_n\} \qquad \forall \mathtt{p}_i\in \mathtt{pa}(G).\, G\downarrow_{\mathtt{p}_i}=T_i \qquad P_i = Q_i\{s[\mathtt{p}_i]/x\}}{\begin{array}{c}(\nu\, s)\left(\prod_{i\in\{1,\cdots,n\}} \ell_i[\mathtt{p}_i] : \wr\, \mathbf{0}\, ;\, Q_i \int \ \mid s_{\mathtt{p}_i}\lfloor\, \hat{}\,T_i\cdot x_i\cdot[x_i\mapsto a]\rfloor^{\diamondsuit} \mid s:(\epsilon\star\epsilon)\right)\\ \rightsquigarrow \\ \ell_1\,\{a!\langle x_1:T_1\rangle.P_1\} \mid \prod_{i\in\{2,\cdots,n\}}\ell_i\,\{a?(x_i:T_i).P_i\}\end{array}}$$

$$\text{(RollS)}\ \frac{}{\begin{array}{c} s_{\mathtt{p}}\lfloor \mathbb{T}\,[\mathtt{q}?\langle U\rangle.\,\hat{}\,T]\cdot\widetilde{x}\cdot\sigma_1\rfloor^{\diamondsuit} \mid s_{\mathtt{q}}\lfloor \mathbb{S}\,[\mathtt{p}!\langle U\rangle.\,\hat{}\,S]\cdot\widetilde{y}\cdot\sigma_2\rfloor^{\diamondsuit} \mid s:(h_i\star h_o)\\ \rightsquigarrow\\ s_{\mathtt{p}}\lfloor \mathbb{T}\,[\mathtt{q}?\langle U\rangle.\,\hat{}\,T]\cdot\widetilde{x}\cdot\sigma_1\rfloor^{\blacklozenge} \mid s_{\mathtt{q}}\lfloor \mathbb{S}\,[\mathtt{p}!\langle U\rangle.\,\hat{}\,S]\cdot\widetilde{y}\cdot\sigma_2\rfloor^{\blacklozenge} \mid s:(h_i\star h_o)\end{array}}$$

$$\text{(RollC)}\ \frac{}{\begin{array}{c} s_{\mathtt{p}}\left\lfloor \mathbb{T}\left[\mathtt{q}\&\{l_z:\hat{}\,S_z\,,\,l_w:S_w\}_{z\in J\setminus w}\right]\cdot\widetilde{x}\cdot\sigma_1\right\rfloor^{\diamondsuit} \mid s_{\mathtt{q}}\left\lfloor \mathbb{S}\left[\mathtt{p}\oplus\{l_z:\hat{}\,S_z\,,\,l_w:S_w\}_{z\in J\setminus w}\right]\cdot\widetilde{y}\cdot\sigma_2\right\rfloor^{\diamondsuit} \mid s:(h_i\star h_o)\\ \rightsquigarrow\\ s_{\mathtt{p}}\left\lfloor \mathbb{T}\left[\mathtt{q}\&\{l_z:\hat{}\,S_z\,,\,l_w:S_w\}_{z\in J\setminus w}\right]\cdot\widetilde{x}\cdot\sigma_1\right\rfloor^{\blacklozenge} \mid s_{\mathtt{q}}\left\lfloor \mathbb{S}\left[\mathtt{p}\oplus\{l_z:\hat{}\,S_z\,,\,l_w:S_w\}_{z\in J\setminus w}\right]\cdot\widetilde{y}\cdot\sigma_2\right\rfloor^{\blacklozenge} \mid s:(h_i\star h_o)\end{array}}$$

$$\text{(ROut)}\ \frac{\mathtt{p}=\mathtt{r}\ \vee\ \mathtt{p}\in\mathbf{roles}(\mathtt{r},h_i)}{\begin{array}{c}\ell[\mathtt{r}]:\wr\, \mathtt{C}\,;\,P\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\mathtt{q}!\langle U\rangle.\,\hat{}\,S]\cdot\widetilde{x}\cdot\sigma\rfloor^{\blacklozenge} \mid s:(h_i\star(\mathtt{p},\mathtt{q},V)\circ h_o)\\ \rightsquigarrow\\ \ell[\mathtt{r}]:\wr\, \mathtt{C}\,;\,s[\mathtt{p}]!\langle V\rangle.P\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\,\hat{}\,\mathtt{q}!\langle U\rangle.S]\cdot\widetilde{x}\cdot\sigma\rfloor^{\diamondsuit} \mid s:(h_i\star h_o)\end{array}}$$

$$\text{(RIn)}\ \frac{\mathtt{p}=\mathtt{r}\ \vee\ \mathtt{p}\in\mathbf{roles}(\mathtt{r},h_i)}{\begin{array}{c}\ell[\mathtt{r}]:\wr\, \mathtt{C}\,;\,P\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\mathtt{q}?\langle U\rangle.\,\hat{}\,S]\cdot\widetilde{x},y\cdot\sigma\rfloor^{\blacklozenge} \mid s:(h_i\circ(\mathtt{q},\mathtt{p},V)\star h_o)\\ \rightsquigarrow\\ \ell[\mathtt{r}]:\wr\, \mathtt{C}\,;\,s[\mathtt{p}]?(y).P\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\,\hat{}\,\mathtt{q}?\langle U\rangle.S]\cdot\widetilde{x}\cdot\sigma\setminus y\rfloor^{\diamondsuit} \mid s:(h_i\star(\mathtt{q},\mathtt{p},V)\circ h_o)\end{array}}$$

$$\text{(RBra)}\ \frac{\mathtt{p}=\mathtt{r}\ \vee\ \mathtt{p}\in\mathbf{roles}(\mathtt{r},h_i) \qquad w\in I \qquad I\subseteq J}{\begin{array}{c}\ell[\mathtt{r}]:\wr\, \mathtt{C}, s[\mathtt{p}]\triangleright\{l_i:P_i\}_{i\in I\setminus\{w\}}\,;\,P\int \ \mid s_{\mathtt{p}}\left\lfloor\mathbb{T}\left[\mathtt{q}\&\{l_j:S_j\,,\,l_w:\hat{}\,S_w\}_{j\in J\setminus w}\right]\cdot\widetilde{x}\cdot\sigma\right\rfloor^{\blacklozenge} \mid s:(h_i\circ(\mathtt{q}\,,\,\mathtt{p}\,,\,l_w)\star h_o)\\ \rightsquigarrow\\ \ell[\mathtt{r}]:\wr\, \mathtt{C}\,;\,s[\mathtt{p}]\triangleright\{l_i:P_i,l_w:P\}_{i\in I\setminus\{w\}}\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\,\hat{}\,\mathtt{q}\&\{l_j:S_j\}_{j\in J}]\cdot\widetilde{x}\cdot\sigma\rfloor^{\diamondsuit} \mid s:(h_i\star(\mathtt{q}\,,\,\mathtt{p}\,,\,l_w)\circ h_o)\end{array}}$$

$$\text{(RSel)}\ \frac{\mathtt{p}=\mathtt{r}\ \vee\ \mathtt{p}\in\mathbf{roles}(\mathtt{r},h_i) \qquad w\in I \qquad I\subseteq J}{\begin{array}{c}\ell[\mathtt{r}]:\wr\, \mathtt{C}, s[\mathtt{p}]\triangleleft\{l_i.P_i\}_{i\in I}\,;\,P\int \ \mid s_{\mathtt{p}}\left\lfloor\mathbb{T}\left[\mathtt{q}\oplus\{l_j:S_j\,,\,l_w:\hat{}\,S_w\}_{j\in J\setminus w}\right]\cdot\widetilde{x}\cdot\sigma\right\rfloor^{\blacklozenge} \mid s:(h_i\star(\mathtt{p}\,,\,\mathtt{q}\,,\,l_w)\circ h_o)\\ \rightsquigarrow\\ \ell[\mathtt{r}]:\wr\, \mathtt{C}\,;\,s[\mathtt{p}]\triangleleft\{l_w.P\}+s[\mathtt{p}]\triangleleft\{l_i.P_i\}_{i\in I}\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\,\hat{}\,\mathtt{q}\oplus\{l_j:S_j\}_{j\in J}]\cdot\widetilde{x}\cdot\sigma\rfloor^{\diamondsuit} \mid s:(h_i\star h_o)\end{array}}$$

$$\text{(RBeta)}\ \frac{}{(\nu\, k)\ \left(\ell[\mathtt{p}]:\wr\, \mathtt{C}\,;\,Q\int \mid k\,\lfloor(V\,w)\,,\,\ell\rfloor \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[k.\,\hat{}\,S]\cdot\widetilde{x}\cdot\sigma\rfloor\right) \rightsquigarrow \ell[\mathtt{p}]:\wr\, \mathtt{C}\,;\,(V\,w)\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\,\hat{}\,S]\cdot\widetilde{x}\cdot\sigma\rfloor}$$

$$\text{(RSpawn)}\ \frac{}{\begin{array}{c}(\nu\,\ell_1,\ell_2)\ \left(\ell[\mathtt{p}]:\wr\, \mathtt{C}\,;\,\mathbf{0}\int \mid \ell_1[\mathtt{p}]:\wr\,\mathbf{0}\,;\,P\int \ \mid \ell_2[\mathtt{p}]:\wr\,\mathbf{0}\,;\,Q\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[(\ell,\ell_1,\ell_2).\,\hat{}\,S]\cdot\widetilde{x}\cdot\sigma\rfloor\right)\\ \rightsquigarrow\\ \ell[\mathtt{p}]:\wr\, \mathtt{C}\,;\,P\mid Q\int \ \mid s_{\mathtt{p}}\lfloor\mathbb{T}\,[\,\hat{}\,S]\cdot\widetilde{x}\cdot\sigma\rfloor\end{array}}$$

Figure 6: Decoupled semantics for configurations: Backwards reduction ($\rightsquigarrow$).

(AC)

$$\frac{\mathtt{p} = \mathtt{r}_1 \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}_1, h_i) \qquad \mathtt{q} = \mathtt{r}_2 \vee \mathtt{q} \in \mathbf{roles}(\mathtt{r}_2, h_i)}{\begin{array}{c} \ell_1[\mathtt{r}_1] : \wr \mathtt{C} \,;\, s[\mathtt{p}]!\langle V\rangle.P \int \mid s_{\mathtt{p}} \lfloor \mathbb{T}\,[\,\hat{}\,\mathtt{q}!\langle U\rangle.S] \cdot \widetilde{x} \cdot \sigma \rfloor \mid \ell_2[\mathtt{r}_2] : \wr \mathtt{C} \,;\, s[\mathtt{q}]?(y).Q \int \mid s_{\mathtt{q}} \lfloor \mathbb{S}\,[\,\hat{}\,\mathtt{p}?\langle U\rangle.T] \cdot \widetilde{x} \cdot \sigma \rfloor \mid s : (h_i \star h_o) \\ \Rrightarrow \\ \ell_1[\mathtt{r}_1] : \wr \mathtt{C} \,;\, P \int \mid s_{\mathtt{p}} \lfloor \mathbb{T}\,[\mathtt{q}!\langle U\rangle.\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma \rfloor \mid \ell_2[\mathtt{r}_2] : \wr \mathtt{C} \,;\, Q \int \mid s_{\mathtt{q}} \lfloor \mathbb{S}\,[\mathtt{p}?\langle U\rangle.\,\hat{}\,T] \cdot \widetilde{x}, y \cdot \sigma[y \mapsto V] \rfloor \mid s : (h_i \circ (\mathtt{q}\,,\,\mathtt{p}\,,\,V) \star h_o) \end{array}}$$

$$\text{(AS)}\ \frac{\mathtt{p} = \mathtt{r}_1 \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}_1, h_i) \qquad \mathtt{q} = \mathtt{r}_2 \vee \mathtt{q} \in \mathbf{roles}(\mathtt{r}_2, h_i)}{\begin{array}{c} \ell_1[\mathtt{r}_1] : \wr \mathtt{C} \,;\, s[\mathtt{p}] \triangleleft \{l_i.P_i\}_{i \in I} \int \mid s_{\mathtt{p}} \lfloor \mathbb{S}\,[\,\hat{}\,\mathtt{q} \oplus \{l_j : S_j\}_{j \in J}] \cdot \widetilde{x} \cdot \sigma \rfloor \\ \ell_2[\mathtt{r}_2] : \wr \mathtt{C} \,;\, s[\mathtt{q}] \triangleright \{l_i : Q_i\}_{i \in I} \int \mid s_{\mathtt{p}} \lfloor \mathbb{T}\,[\,\hat{}\,\mathtt{p}\&\{l_j : T_j\}_{j \in J}] \cdot \widetilde{x} \cdot \sigma \rfloor \mid s : (h_i \star h_o) \\ \Rrightarrow \\ \ell_1[\mathtt{r}_1] : \wr \mathtt{C}, s[\mathtt{p}] \triangleright \{l_l : P_l\}_{l \in I \setminus w} \,;\, P_w \int \mid s_{\mathtt{p}} \left\lfloor \mathbb{S}\left[\mathtt{q}\&\{l_j : S_j\,,\, l_w :\, \hat{}\,S_w\}_{j \in J \setminus w}\right] \cdot \widetilde{x} \cdot \sigma \right\rfloor \\ \ell_2[\mathtt{r}_2] : \wr \mathtt{C}, s[\mathtt{q}] \triangleleft \{l_l.Q_l\}_{l \in I \setminus w} \,;\, Q_w \int \mid s_{\mathtt{q}} \left\lfloor \mathbb{T}\left[\mathtt{p} \oplus \{l_j : T_j\,,\, l_w :\, \hat{}\,T_w\}_{j \in J \setminus w}\right] \cdot \widetilde{x} \cdot \sigma \right\rfloor \mid s : (h_i \circ (\mathtt{p}\,,\,\mathtt{q}\,,\,l_w) \star h_o) \end{array}}$$

(RAC)

$$\frac{\mathtt{p} = \mathtt{r}_1 \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}_1, h_i) \qquad \mathtt{q} = \mathtt{r}_2 \vee \mathtt{q} \in \mathbf{roles}(\mathtt{r}_2, h_i)}{\begin{array}{c} \ell_1[\mathtt{r}_1] : \wr \mathtt{C} \,;\, P \int \mid s_{\mathtt{p}} \lfloor \mathbb{T}\,[\mathtt{q}!\langle U\rangle.\,\hat{}\,S] \cdot \widetilde{x} \cdot \sigma \rfloor \mid \ell_2[\mathtt{r}_2] : \wr \mathtt{C} \,;\, Q \int \mid s_{\mathtt{q}} \lfloor \mathbb{S}\,[\mathtt{p}?\langle U\rangle.\,\hat{}\,T] \cdot \widetilde{x}, y \cdot \sigma[y \mapsto V] \rfloor \mid s : (h_i \circ (\mathtt{q}\,,\,\mathtt{p}\,,\,V) \star h_o) \\ \Lleftarrow \\ \ell_1[\mathtt{r}_1] : \wr \mathtt{C} \,;\, s[\mathtt{p}]!\langle V\rangle.P \int \mid s_{\mathtt{p}} \lfloor \mathbb{T}\,[\,\hat{}\,\mathtt{q}!\langle U\rangle.S] \cdot \widetilde{x} \cdot \sigma \rfloor \mid \ell_2[\mathtt{r}_2] : \wr \mathtt{C} \,;\, s[\mathtt{p}]?(y).Q \int \mid s_{\mathtt{q}} \lfloor \mathbb{S}\,[\,\hat{}\,\mathtt{p}?\langle U\rangle.T] \cdot \widetilde{x} \cdot \sigma \rfloor \mid s : (h_i \star h_o) \end{array}}$$

$$\text{(RAS)}\ \frac{\mathtt{p} = \mathtt{r}_1 \vee \mathtt{p} \in \mathbf{roles}(\mathtt{r}_1, h_i) \qquad \mathtt{q} = \mathtt{r}_2 \vee \mathtt{q} \in \mathbf{roles}(\mathtt{r}_2, h_i)}{\begin{array}{c} \ell_1[\mathtt{r}_1] : \wr \mathtt{C}, s[\mathtt{p}] \triangleright \{l_l : P_l\}_{l \in I \setminus w} \,;\, P_w \int \mid s_{\mathtt{p}} \left\lfloor \mathbb{S}\left[\mathtt{q}\&\{l_j : S_j\,,\, l_w :\, \hat{}\,S_w\}_{j \in J \setminus w}\right] \cdot \widetilde{x} \cdot \sigma \right\rfloor \mid \\ \ell_2[\mathtt{r}_2] : \wr \mathtt{C}, s[\mathtt{q}] \triangleleft \{l_l.Q_l\}_{l \in I \setminus w} \,;\, Q_w \int \mid s_{\mathtt{q}} \left\lfloor \mathbb{T}\left[\mathtt{p} \oplus \{l_j : T_j\,,\, l_w :\, \hat{}\,T_w\}_{j \in J \setminus w}\right] \cdot \widetilde{x} \cdot \sigma \right\rfloor \mid s : (h_i \circ (\mathtt{p}\,,\,\mathtt{q}\,,\,l_w) \star h_o) \\ \Lleftarrow \\ \ell_1[\mathtt{r}_1] : \wr \mathtt{C} \,;\, s[\mathtt{p}] \triangleleft \{l_i.P_i\}_{i \in I} \int \mid s_{\mathtt{p}} \lfloor \mathbb{S}\,[\,\hat{}\,\mathtt{q} \oplus \{l_j : S_j\}_{j \in J}] \cdot \widetilde{x} \cdot \sigma \rfloor \mid \\ \ell_2[\mathtt{r}_2] : \wr \mathtt{C} \,;\, s[\mathtt{q}] \triangleright \{l_i : Q_i\}_{i \in I} \int \mid s_{\mathtt{p}} \lfloor \mathbb{T}\,[\,\hat{}\,\mathtt{p}\&\{l_j : T_j\}_{j \in J}] \cdot \widetilde{x} \cdot \sigma \rfloor \mid s : (h_i \star h_o) \end{array}}$$

Figure 7: Atomic semantics for configurations: Forward and backward reduction ($\Rrightarrow$ and $\Lleftarrow$).

$$\text{(Sw1)}\ \frac{\{\mathtt{p}_1, \mathtt{q}_1\} \# \{\mathtt{p}_2, \mathtt{q}_2\}}{\begin{array}{l} \mathtt{p}_1 \to \mathtt{q}_1 : \langle U_1\rangle.(\mathtt{p}_2 \to \mathtt{q}_2 : \langle U_2\rangle.G) \approx_{\mathtt{sw}} \\ \qquad \mathtt{p}_2 \to \mathtt{q}_2 : \langle U_2\rangle.(\mathtt{p}_1 \to \mathtt{q}_1 : \langle U_1\rangle.G) \end{array}}$$

$$\text{(Sw2)}\ \frac{\{\mathtt{p}_1, \mathtt{q}_1\} \# \{\mathtt{p}_2, \mathtt{q}_2\}}{\begin{array}{l} \mathtt{p}_1 \to \mathtt{q}_1 : \langle U_1\rangle.(\mathtt{p}_2 \to \mathtt{q}_2 : \{l_i : G_i\}_{i \in I}) \approx_{\mathtt{sw}} \\ \qquad \mathtt{p}_2 \to \mathtt{q}_2 : \{l_i : (\mathtt{p}_1 \to \mathtt{q}_1 : \langle U_1\rangle.G_i)\}_{i \in I} \end{array}}$$

$$\text{(Sw3)}\ \frac{\{\mathtt{p}_1, \mathtt{q}_1\} \# \{\mathtt{p}_2, \mathtt{q}_2\}}{\begin{array}{l} \mathtt{p}_1 \to \mathtt{q}_1 : \{l_i{:}(\mathtt{p}_2 \to \mathtt{q}_2 : \{l_j{:}G_j\}_{j \in J})\}_{i \in I} \approx_{\mathtt{sw}} \\ \qquad \mathtt{p}_2 \to \mathtt{q}_2 : \{l_j{:}(\mathtt{p}_1 \to \mathtt{q}_1 : \{l_i{:}G_i\}_{i \in I})\}_{j \in J} \end{array}}$$

Figure 8: Swapping on global types G. We write $A\#B$ if A and B are disjoint sets.

$$\frac{}{\Gamma \vdash \mathbf{0} :: x : \mathsf{end}} \qquad \frac{}{\Gamma, X : T \vdash X :: x : T} \qquad \frac{\Gamma, X : T \vdash P :: x : T}{\Gamma \vdash \mu X.P :: x : T}$$

$$\frac{\Gamma \vdash P :: x : T \quad \vdash V :: U}{\Gamma \vdash x!\langle V \rangle.P :: x : \mathtt{p}!\langle U \rangle.T} \qquad \frac{\Gamma, y : U \vdash P :: x : T}{\Gamma \vdash x?(y).P :: x : \mathtt{p}?(U).T}$$

$$\frac{\forall i \in \{1, \ldots, n\}.(\Gamma \vdash P_i :: x : T_i)}{\Gamma \vdash x \triangleleft \{l_i.P_i\}_{i \in \{1,\ldots,n\}} :: x : \mathtt{q}\oplus\{l_i : T_i\}_{i \in \{1,\ldots,n\}}}$$

$$\frac{\forall i \in \{1, \ldots, n\}.(\Gamma \vdash P_i :: x : T_i)}{\Gamma \vdash x \triangleright \{l_i : P_i\}_{i \in \{1,\ldots,n\}} :: x : \mathtt{q}\&\{l_i : T_i\}_{i \in \{1,\ldots,n\}}}$$

Figure 9: Well-formed processes with respect to a local type.

$$\frac{P \bowtie_x T}{\wr \mathbf{0} \,;\, P \int \bowtie_x {}^\wedge T}$$

$$\frac{\wr \mathtt{C} \,;\, x!\langle V \rangle.P \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}!\langle U \rangle.S]}{\wr \mathtt{C} \,;\, P \int \bowtie_x \mathbb{T}[\mathtt{q}!\langle U \rangle.{}^\wedge S]} \qquad \frac{\wr \mathtt{C} \,;\, x?(y).P \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}?\langle U \rangle.S]}{\wr \mathtt{C} \,;\, P \int \bowtie_x \mathbb{T}[\mathtt{q}?\langle U \rangle.{}^\wedge S]}$$

$$\frac{\wr \mathtt{C} \,;\, x \triangleleft \{l_i.P_i\}_{i \in I} \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}\oplus\{l_j : S_j\}_{j \in J}] \quad w \in I, J}{\wr \mathtt{C}, x \triangleleft \{l_l.P_l\}_{l \in I \setminus w} \,;\, P_w \int \bowtie_x \mathbb{T}\left[\mathtt{q}\oplus\{l_j : S_j \,,\, l_w : {}^\wedge S_w\}_{j \in J \setminus w}\right]}$$

$$\frac{\wr \mathtt{C} \,;\, x \triangleright \{l_i.P_i\}_{i \in I} \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}\&\{l_j : S_j\}_{j \in J}] \quad w \in I, J}{\wr \mathtt{C}, x \triangleright \{l_l.P_l\}_{l \in I \setminus w} \,;\, P_w \int \bowtie_x \mathbb{T}\left[\mathtt{q}\&\{l_j : S_j \,,\, l_w : {}^\wedge S_w\}_{j \in J \setminus w}\right]}$$

$$\frac{\wr \mathtt{C} \,;\, P \int \bowtie_x \mathbb{T}[\mathtt{q}!\langle U \rangle.{}^\wedge S] \quad \vdash V :: U}{\wr \mathtt{C} \,;\, x!\langle V \rangle.P \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}!\langle U \rangle.S]}$$

$$\frac{\wr \mathtt{C} \,;\, P \int \bowtie_x \mathbb{T}[\mathtt{q}?\langle U \rangle.{}^\wedge S] \quad y : U \vdash P :: S}{\wr \mathtt{C} \,;\, x?(y).P \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}?\langle U \rangle.S]}$$

$$\frac{\wr \mathtt{C}, x \triangleleft \{l_l.P_l\}_{l \in I} \,;\, P_w \int \bowtie_x \mathbb{T}[\mathtt{q}\oplus\{l_j : S_j \,,\, l_w : {}^\wedge S_w\}_{j \in J}]}{\wr \mathtt{C} \,;\, x \triangleleft \{l_i.P_i\}_{i \in I \cup w} \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}\oplus\{l_j : S_j\}_{j \in J \cup w}]}$$

$$\frac{\wr \mathtt{C}, x \triangleright \{l_l.P_l\}_{l \in I} \,;\, P_w \int \bowtie_x \mathbb{T}[\mathtt{q}\&\{l_j : S_j \,,\, l_w : {}^\wedge S_w\}_{j \in J}]}{\wr \mathtt{C} \,;\, x \triangleright \{l_i.P_i\}_{i \in I \cup w} \int \bowtie_x \mathbb{T}[{}^\wedge \mathtt{q}\&\{l_j : S_j\}_{j \in J \cup w}]}$$

Figure 10: Well-formed configurations with respect to a local type with history.