Concurrent Reversible Sessions*

Ilaria Castellani[1], Mariangiola Dezani-Ciancaglini†[2], and Paola Giannini ‡ [3]

1 **Université Côte d'Azur, INRIA,**
2004 Route des Lucioles, 06902 Sophia Antipolis, France
ilaria.castellani@inria.fr

2 **Dipartimento di Informatica, Università di Torino,**
corso Svizzera 185, 10131 Torino, Italy
dezani@di.unito.it

3 **DiSIT, Università del Piemonte Orientale,**
Via Teresa Michel 11, 15121 Alessandria, Italy
paola.giannini@uniupo.it

Abstract

We present a calculus for concurrent reversible multiparty sessions, which improves on recent proposals in several respects: it allows for concurrent and sequential composition within processes and types, it gives a compact representation of the *past* of processes and types, which facilitates the definition of rollback, and it implements a fine-tuned strategy for backward computation. We propose a refined session type system for our calculus and show that it enforces the expected properties of session fidelity, forward and backward progress, as well as causal consistency. In conclusion, our calculus is a conservative extension of previous proposals, offering enhanced expressive power and refined analysis techniques.

1998 ACM Subject Classification F.1.2: Parallelism and concurrency, F. 3.3: Type structure.

Keywords and phrases Communication-centric Systems, Reversible Computation, Process Calculi, Multiparty Session Types.

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.26

1 Introduction

Building on recent proposals, we argue that session types and reversibility can be fruitfully combined to yield a model for correct and reliable communication-centric systems. Session types are a simple but expressive type formalism that specifies the structure of interactions. Traditionally, session types have been used to ensure safety properties of interactions, such as absence of communication errors, deadlock freedom and race freedom.

Reversibility, on its side, may be viewed as a means to improve system flexibility and reliability. Reversing a computation may be defined as the act of undoing some suffix of the computation, in order to return to a previously visited state. A condition which is usually required for undoing a computational step is that all its effects have been already undone [21, 22, 30, 33, 8, 16, 15]. This property is generally referred to as *causal consistency*.

* The authors acknowledge a partial support of COST Action IC1405 on Reversible Computation - extending horizons of computing.
† Partially supported by EU H2020-644235 Rephrase project, EU H2020-644298 HyVar project, IC1402 ARVI and Ateneo/CSP project RunVar.
‡ This original research has the financial support of the Università del Piemonte Orientale.

A stronger property, called *full reversibility* in [25], enables a system to restore exactly a past state, keeping no trace of the rollback [9, 20, 24]. Sometimes neither of these properties can be achieved, particularly in distributed systems, and one has to go for weaker properties. In some cases, it is even desirable that a restored state be not exactly the same as the original one: for instance, the restored state could keep some memory of the undone computational path, so as to avoid engaging in that path again in case it led to an unsuccessful state.

In the setting of structured communications, reversibility has been first studied for contracts [2, 3] and transactions [10, 11, 19]. Only recently has this issue started to be addressed for session calculi, both binary [33, 23] and multiparty [14, 34, 28, 24] (see Sect.6 for more discussion on related work).

When reversing a structured interaction, one has to face the problem of preserving consistency of the global state: if one of the partners triggers a rollback, then all its communicating partners should roll back accordingly. This is where session types come to the rescue, with their precise specification of the functionality of communications (sender, receiver and message), and of the order in which they should occur.

We present a calculus for concurrent reversible multiparty sessions, which extends previous proposals in several ways. First of all, our procotols are more general than those specified by standard multiparty session types [18]: concurrent communications are allowed both between disjoint groups of participants, and, in a controlled way that excludes auto-concurrency, also within participants themselves. More specifically, we relax the linearity constraint of standard multiparty session types, to allow protocol participants to perform communications in parallel, as long as they do not generate races. To enable the control flows to join again after a bunch of parallel interactions, besides parallel composition we also introduce full sequential composition in the syntax of both types and processes.

Moreover, our calculus gives a compact representation of the *past* of processes and types, which facilitates the definition of rollback, and it implements a fine-tuned strategy for backward computation, which is geared towards achieving compliance.

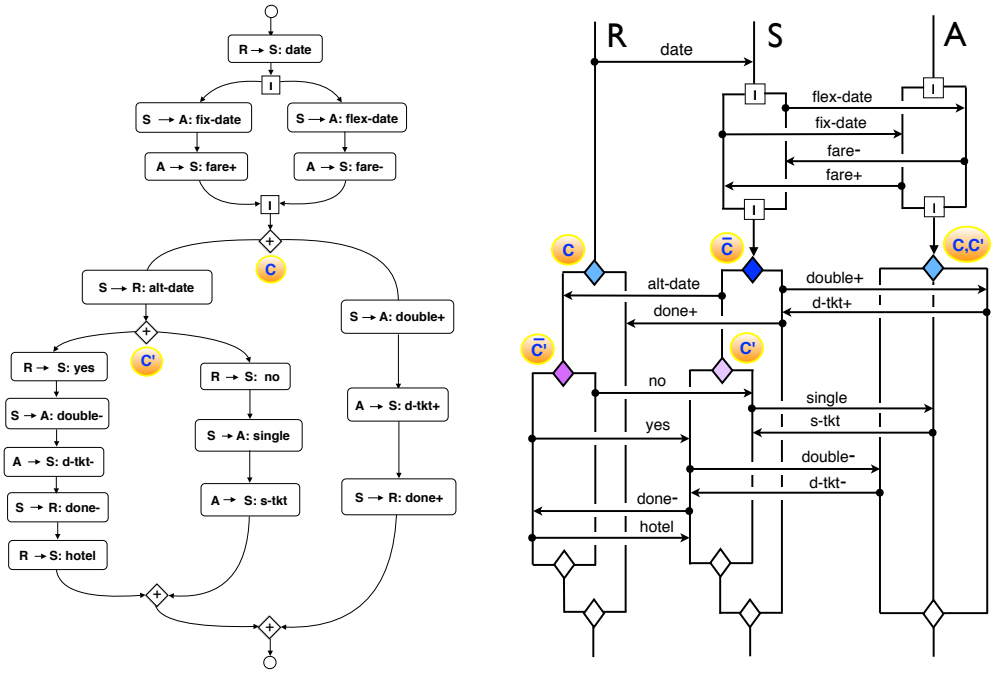
The main contributions of our paper may be summarised as follows:

- the introduction of *concurrent session types* in multiparty session frameworks;
- the proposal of a reversible session calculus which is more general than the existing ones, and which conservatively extends them, preserving the expected properties of session fidelity, forward and backward progress, and causal consistency;
- a fine-tuned strategy for rollback to checkpointed choices, which can only be triggered by choice leaders in predefined states of the computation, leading back to the choice state stripped off the unsuccessful path, so as redirect the computation towards alternative, potentially successful, paths.

The rest of the paper is organised as follows. In Sect.2 we present our running example. In Sect.3 we define the syntax and operational semantics - both forward and backward - of our calculus. In Sect.4 we introduce our extended syntax for global types and session types and we establish well-formedness conditions for global types. Sect.5 presents our type system and proves its soundness, namely that it ensures the expected semantic properties. We conclude in Sect.6 with some discussion on related and future work.

2 Travel protocol example

To illustrate our approach, we present a simple travel protocol, involving three parties, a Researcher **R**, a Student **S** and an Airline company **A**. This protocol, henceforth called the *RSA protocol*, will serve as our running example throughout the paper.



(a) Global graph.

(b) Message Sequence Graph.

Figure 1 RSA protocol.

When graphically representing protocols, we shall use two kinds of graphs: the first one, called *global graph*, is borrowed from [13] and gives a global description of the protocol; the second one, which we call *message sequence graph*, is closer to message sequence charts and emphasises the control flows of the individual partners. In both cases, diamonds will be used as delimiters for the branching construct, and squares as delimiters for the parallel construct.

In the global graph, boxes represent communications and their whole functionality: sender, receiver and message. In the message sequence graph, the control flow of the protocol is split into the individual contributions of the partners: each partner is denoted by a vertical line; communications are represented by arrows from the sender to the receiver, labelled by messages; and branching and parallel constructs are replicated for each involved partner.

The RSA protocol is depicted in Fig.1a and Fig.1b according to these conventions. Let us now informally describe the protocol. Suppose that **R** and **S** wish to travel together to some conference, and that **R** has date constraints but is flexible about prices, while **S** has funding constraints but is flexible about dates. The protocol starts by **R** sending to **S** her chosen date. Then **S** sends two parallel flight requests to **A**: one for the date chosen by **R**, and one for more flexible dates. The airline **A** replies by sending in parallel the respective fares: a higher fare for the fixed date, and a lower fare for the flexible date. At this point, if **S** can afford the higher fare, she will book two tickets at this fare, then send a message done+ to **R**, and the whole interaction will end here.

Suppose now that **S** cannot afford the higher fare. In this case, she will propose to **R** the alternative date corresponding to the cheaper fare. If **R** cannot travel at this date she replies no. Then **S** will book a single ticket for herself at the cheaper fare, and the interaction terminates. If instead this date is suitable for **R**, she replies yes to **S**. Now **S** books two tickets at the cheaper fare and sends a message done- to **R**. Then **R** informs **S** that the hotel

booking has been changed to fit the new date (supposing **R** had previously booked the hotel for her chosen date), thereby closing the interaction.

So far we have described only the forward behaviour of the session. Suppose now that after proposing the alternative date to **R** and receiving **R**'s agreement, **S** discovers that this date is not suitable anymore. Then **S** will trigger a rollback to her internal choice with checkpoint label $\overline{C}$, and **R** will have to roll back to her corresponding external choice with checkpoint label C. At this point, since the first branch of the choice has been already explored with no success, **S** will have to choose the second branch and book the more expensive flight. On her side, **R** could decide to roll back to her internal choice with label $\overline{C'}$ if she discovers that the hotel is fully booked at the alternative date, and then both **S** and **A** will have to roll back to their corresponding external choices with label C'.

In our calculus, only "the leader" of a choice, i.e., the participant who solved the choice by sending the first message, will be authorised to trigger a rollback. In the RSA protocol, the leader of the first choice is **S**, while the leader of the second choice is **R**. In the message sequence graph, the leader is distinguished by the fact that her choice has an overlined checkpoint. In the global graph, which collapses the distributed structure of the interaction, the choice leader is the sender in the first communication after the choice. In fact, the graph of Fig.1b can always be derived from that of Fig.1a through a projection operation.

3 Calculus

In this section, we introduce the syntax and semantics of our calculus. As usual in session calculi, we distinguish between user processes, which have not started to be executed yet, and runtime processes. The executed part of runtime processes will be marked with hats: this extension is instrumental to reversing computations. Processes may be decorated by checkpoint labels, marking them as possible rollback points. Our syntax generalises that of standard multiparty session calculi [18, 12], featuring the additional operators of parallel and sequential composition (the latter replacing the prefixing operator).

We assume the following base sets: *messages*, ranged over by $\lambda, \lambda', \ldots$ and forming the set Msg; *checkpoint labels*, ranged over by C, C' and forming the set ChLa; and *session participants*, ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}$ and forming the set Part.

We use δ to range over general sets of checkpoint labels, and Δ to stand for either δ or δ extended by exactly one overlined checkpoint label:

$$\delta ::= \emptyset \quad | \quad \delta, C \qquad\qquad \Delta ::= \delta \quad | \quad \delta, \overline{C}$$

Sets of checkpoint labels are associated with choices. More precisely, an overlined checkpoint label can only be associated with an internal choice and is said to be *active*. A simple checkpoint label is *passive* and may be associated with both internal and external choices. Intuitively, an overlined checkpoint label is the handle of a backward move: a participant who crossed an internal choice (henceforth also called the *choice leader*) with an active checkpoint label, and then proceeded in the computation, may decide to return to that choice whenever she has the ability to send a message. Within a network, this backward move of the choice leader will have to be matched by backward moves of all the participants who did some action after that checkpoint label.

Let $\pi \in \{\mathsf{p}?\lambda, \mathsf{p}!\lambda \mid \mathsf{p} \in \mathsf{Part}, \lambda \in \mathsf{Msg}\}$ denote an *atomic action*, namely a directed input or output action. An atomic action can bear a hat, in which case it represents an already executed or *past* action. We use $\widetilde{\pi}$ to stand for either π or $\widehat{\pi}$. External and internal choices can also bear hats, indicating that one branch has been chosen. We use $\widetilde{\sum}$ to stand for either $\sum$ or $\widehat{\sum}$, and similarly for $\bigoplus$.

▶ **Definition 1.** *Processes* are defined by:

$$P \quad ::= \quad \widetilde{\sum}_{i\in I}\widetilde{\pi}_i; P_i \quad | \quad \widetilde{\bigoplus}_{i\in I}\widetilde{\pi}_i; P_i \quad | \quad {}_\Delta P \quad | \quad P \mid P \quad | \quad P; P \quad | \quad \mu X.P \quad | \quad X \quad | \quad \mathsf{skip}$$

Processes without and with hats are called respectively *user processes* and *runtime processes*.

We will omit empty sets of checkpoint labels, choice symbols in one-branch choices, and trailing skip processes. External and internal choices are assumed to be associative, commutative, idempotent, and non-empty (except when combined with binary choices in evaluation contexts or finished processes, see below). A choice with a single branch may be either an *input process* or an *output process*. Parallel composition is associative and commutative, with neutral element skip. Sequential composition is associative, with neutral element skip. The operators have the following precedence: ';', '+', '$\oplus$','|'.

Following [27], we require recursion to be:

- *guarded*, i.e. a recursion variable X can only appear free in the second argument of a sequential composition whose first argument is different from skip;
- *sequential*, i.e. a recursion variable X cannot occur free in any branch of a parallel composition.

Processes are treated equi-recursively, i.e. they are identified with their generated tree [31].

The typing rules of Sect.5 will ensure the following well-formedness conditions for processes:

1. External choices are between input processes;
2. Internal choices are between output processes;
3. A choice without a hat has branches without hats;
4. A choice with a hat has exactly one branch with hats;
5. Only choices are decorated with sets of checkpoint labels;
6. Only internal choices are decorated with an active checkpoint label.

Conditions 1 and 2 reflect the active role of outputs and the passive role of inputs. Conditions 3 and 4 express the fact that executing a choice amounts to executing one of its branches. Conditions 5 and 6 specify that choices are the only return points for backward reductions. In the following we will take advantage of these restrictions to simplify definitions.

In a full-fledged calculus, messages would carry values, namely they would be of the form $\lambda(v)$. Here, for simplicity we consider only pure messages.

Networks are parallel compositions of pairs $\mathsf{p}[\![\, P \,]\!]$, where participant p has behaviour P.

▶ **Definition 2.** *Networks* are defined by: $\quad \mathbb{N} \quad ::= \quad \mathsf{p}[\![\, P \,]\!] \quad | \quad \mathbb{N} \parallel \mathbb{N}$

The operator $\parallel$ is associative and commutative, with neutral element $\mathsf{p}[\![\, \mathsf{skip} \,]\!]$ for each p.

The operational semantics is given by two LTSs, one for processes and one for networks. In the LTS for processes, *forward transitions* have the form $P \xrightarrow{\pi} P'$ and *backward transitions* have the form $P \overset{\overline{C}}{\circlearrowleft} P'$ or $P \overset{C}{\circlearrowleft} P'$. We define $P \downarrow_{out}$ if $P \xrightarrow{\mathsf{p}!\lambda} P'$ for some p, λ, P'.

In the LTS for networks, forward and backward transitions have respectively the form $N \xrightarrow{\mathsf{p}\lambda\mathsf{q}} N'$ and $N \overset{C}{\circlearrowleft} N'$.

▶ **Definition 3.** *Evaluation contexts* $\mathcal{E}$ are defined by:

$$\mathcal{E} \quad ::= \quad {}_\delta(\widehat{\sum}_{i\in I}\pi_i; P_i \mathbin{\widehat{+}} \widehat{\pi}; \mathcal{E}) \quad | \quad {}_\Delta(\widehat{\bigoplus}_{i\in I}\pi_i; P_i \mathbin{\widehat{\oplus}} \widehat{\pi}; \mathcal{E}) \quad | \quad \mathcal{E} \mid P \quad | \quad \mathcal{E}; P \quad | \quad F; \mathcal{E} \quad | \quad [\,]$$

The definition of evaluation context ensures that in a sequential composition the evaluation of the second component can only start when the first is a *finished process* F, defined by:

$$F \quad ::= \quad {}_\delta(\widehat{\sum}_{i\in I}\pi_i; P_i \mathbin{\widehat{+}} \widehat{\pi}; F) \quad | \quad {}_\Delta(\widehat{\bigoplus}_{i\in I}\pi_i; P_i \mathbin{\widehat{\oplus}} \widehat{\pi}; F) \quad | \quad F \mid F \quad | \quad F; F \quad | \quad \mathsf{skip}$$

The LTSs for processes and networks are given in Fig.2. Rules [ExtCh] and [IntCh] allow an action to be extracted from one of the summands, as usual, but instead of discarding

$$ {}_\delta\textstyle\sum_{i\in I}\pi_i; P_i \xrightarrow{\pi_j} {}_\delta(\widehat{\sum}_{i\in I\setminus\{j\}}\pi_i; P_i \mathbin{\widehat{+}} \widehat{\pi}_j; P_j) \quad j\in I \qquad [\textsc{ExtCh}] $$

$$ {}_\Delta\textstyle\bigoplus_{i\in I}\pi_i; P_i \xrightarrow{\pi_j} {}_\Delta(\widehat{\bigoplus}_{i\in I\setminus\{j\}}\pi_i; P_i \mathbin{\widehat{\oplus}} \widehat{\pi}_j; P_j) \quad j\in I \qquad [\textsc{IntCh}] $$

$$ \frac{P\downarrow_{out} \quad \overline{C}\in\Delta \quad I\neq\emptyset}{{}_\Delta(\widehat{\bigoplus}_{i\in I}P_i \mathbin{\widehat{\oplus}} P) \overset{\overline{C}}{\circlearrowright} {}_\Delta\textstyle\bigoplus_{i\in I}P_i}\ [\textsc{BackCkt}] \qquad \frac{C\in\Delta}{{}_\Delta S \overset{C}{\circlearrowright} {}_\Delta\wr S\int}\ [\textsc{BackP}] $$

$$ \frac{P\xrightarrow{\pi}P'}{\mathcal{E}[P]\xrightarrow{\pi}\mathcal{E}[P']}\ [\textsc{CtFAt}] \qquad \frac{P\overset{\overline{C}}{\circlearrowright}P' \quad \mathcal{E}\ \mathsf{ok}\text{ for }\overline{C}}{\mathcal{E}[P]\overset{\overline{C}}{\circlearrowright}\mathcal{E}[P']}\ [\textsc{CtBA}] \qquad \frac{P\overset{C}{\circlearrowright}P' \quad \mathcal{E}\ \mathsf{ok}\text{ for }C}{\mathcal{E}[P]\overset{C}{\circlearrowright}\mathcal{E}[P']}\ [\textsc{CtBP}] $$

$$ \frac{P_1\xrightarrow{\mathsf{p}?\lambda}P_1' \quad P_2\xrightarrow{\mathsf{q}!\lambda}P_2'}{\mathsf{q}[\![\,P_1\,]\!]\parallel\mathsf{p}[\![\,P_2\,]\!]\parallel\mathbb{N}\xrightarrow{\mathsf{p}\lambda\mathsf{q}}\mathsf{q}[\![\,P_1'\,]\!]\parallel\mathsf{p}[\![\,P_2'\,]\!]\parallel\mathbb{N}}\ [\textsc{Com}] $$

$$ \frac{P\overset{\overline{C}}{\circlearrowright}P' \quad P_i\overset{C}{\circlearrowright}P_i' \ \ i\in I \quad P_j\overset{C}{\not\circlearrowright} \ \ j\in J}{\mathsf{p}[\![\,P\,]\!]\parallel\Pi_{i\in I}\mathsf{p}_i[\![\,P_i\,]\!]\parallel\Pi_{j\in J}\mathsf{p}_j[\![\,P_j\,]\!]\overset{C}{\circlearrowright}\mathsf{p}[\![\,P'\,]\!]\parallel\Pi_{i\in I}\mathsf{p}_i[\![\,P_i'\,]\!]\parallel\Pi_{j\in J}\mathsf{p}_j[\![\,P_j\,]\!]}\ [\textsc{Back}] $$

Figure 2 LTS for processes and networks.

the other summands they record the fact that the choice has been crossed by marking the choice operator with a hat. With this technique, inspired by [6] and already used for reversible computations in [24], all the dynamic operators are turned into static operators, and nothing is lost of the original user process. Notice that when $I=\{j\}$ these rules become $\pi_j; P_j \xrightarrow{\pi_j} \widehat{\pi}_j; P_j$. Rule [BackCkt] is the main backward rule: it applies to a past internal choice, where one branch has been partially executed, and it allows the process to roll back to the original choice where the executed branch is removed. For this to be possible, the choice must have at least one non executed branch P_i and a set of checkpoint labels containing an overlined label $\overline{C}$, which will label the back transition. This is essential to ensure (by means of typing) that the choice leader will be the only one who can decide to roll back to this choice. The condition $P\downarrow_{out}$ means that in order to trigger a rollback, P should be "in lead" again, namely able to do an output.

Rule [BackP], where S denotes either $\widehat{\sum}_{i\in I}P_i$, or $\widehat{\bigoplus}_{i\in I}P_i$, is needed to allow the remaining participants to roll back. The mapping $\wr\,\int$ erases hats from processes, yielding user processes, i.e. $\wr\widehat{\pi}; P\int = \pi$; $\wr\,P\int$ and $\wr\,\int$ acts homomorphically otherwise. Note that the rollback rules can only be applied to processes that are not user processes: in particular, one branch can be erased only if at least one of its actions has been executed. An evaluation context $\mathcal{E}$ is ok for $\overline{C}$ (C) if $\overline{C}\notin\Delta$ ($C\notin\Delta$) whenever $\mathcal{E}$ is a sub-context of a context of the shape ${}_\Delta(\widehat{\sum}_{i\in I}P_i \mathbin{\widehat{+}} \widehat{\pi}; \mathcal{E}')$ or ${}_\Delta(\widehat{\bigoplus}_{i\in I}P_i \mathbin{\widehat{\oplus}} \widehat{\pi}; \mathcal{E}')$. We use this condition in rules [CtBA] and [CtBP] to assure that all participants involved in a recursion go back to the same checkpoint, namely to the first one, as in [28]. This is needed to assure subject reduction, see Example 8.

Rule [Com] is standard. We write $P\overset{C}{\not\circlearrowright}$ if neither Rule [BackP] nor Rule [CtBP] (with label C) can be applied to P. This means that C can only occur in user processes within P. In a well-typed network, Rule [Back] will make participant p roll back to an internal choice and moreover, all participants that can roll back to this choice will do so in the same step. This will be the basis for our soundness result in Sect.5. A direct implementation of this

rule is clearly unrealistic. To that purpose, asynchronous communications including rollback messages should be used, as in [24]. As expected, sequences of network transitions generate *traces*, which are words over the infinite alphabet $\{\mathsf{p}\lambda\mathsf{q} \mid \mathsf{p},\mathsf{q} \in \mathsf{Part}, \lambda \in \mathsf{Msg}\} \cup \mathsf{ChLa}$.

When the labels of transitions are not relevant, we write them simply as $\longrightarrow$ and $\circlearrowright$. In this case, we use $\longrightarrow^*$ to denote the reflexive and transitive closure of $\longrightarrow$ and $\overset{\circlearrowright}{\longrightarrow}{}^*$ to denote the reflexive and transitive closure of $\longrightarrow \cup \circlearrowright$.

Note that our semantics does not enforce any scheduling policy. Hence, a network may generate an infinite trace that does not involve all participants. For instance, the network:

$$\mathsf{p}[\![\, \mu X_1.\mathsf{q}!\lambda_1;X_1 \mid \mu X_2.\mathsf{r}!\lambda_2;X_2 \,]\!] \parallel \mathsf{q}[\![\, \mu X_3.\mathsf{p}?\lambda_1;X_3 \,]\!] \parallel \mathsf{r}[\![\, \mu X_4.\mathsf{p}?\lambda_2;X_4 \,]\!]$$

may generate the infinite trace $\mathsf{p}\lambda_1\mathsf{q}\,\mathsf{p}\lambda_1\mathsf{q}\cdots$. In a more elaborate calculus, we could impose a *fair scheduling policy*, forcing p to communicate alternately with q and r, and in general, no communication to be iterated until each participant has communicated at least once.

A network for our RSA travel protocol in Sect.2 is $\mathbb{N}_{in} = \mathbf{R}[\![\, P^{\mathbf{R}} \,]\!] \parallel \mathbf{S}[\![\, P^{\mathbf{S}} \,]\!] \parallel \mathbf{A}[\![\, P^{\mathbf{A}} \,]\!]$, defined as follows (where messages are abbreviated in a programming language style):

- The process $P^{\mathbf{S}}$ is $\mathbf{R}?\mathtt{dt};(P_1^{\mathbf{S}} \mid P_2^{\mathbf{S}})\,;\,(P_3^{\mathbf{S}} \oplus_{\{\overline{C}\}} P_4^{\mathbf{S}})$ where

$$P_1^{\mathbf{S}} = \mathbf{A}!\mathtt{fxDt};\mathbf{A}?\mathtt{frPl} \qquad P_2^{\mathbf{S}} = \mathbf{A}!\mathtt{flDt};\mathbf{A}?\mathtt{frMn}$$
$$P_3^{\mathbf{S}} = \mathbf{R}!\mathtt{alDt};(P_5^{\mathbf{S}} +_{\{C'\}} P_6^{\mathbf{S}}) \qquad P_4^{\mathbf{S}} = \mathbf{A}!\mathtt{dbPl};\mathbf{A}?\mathtt{dTkPl};\mathbf{R}!\mathtt{dnPl}$$
$$P_5^{\mathbf{S}} = \mathbf{R}?\mathtt{yes};\mathbf{A}!\mathtt{dbMn};P_7^{\mathbf{S}} \quad P_7^{\mathbf{S}} = \mathbf{A}?\mathtt{dTkMn};\mathbf{R}!\mathtt{dnMn};\mathbf{R}?\mathtt{htl} \qquad P_6^{\mathbf{S}} = \mathbf{R}?\mathtt{no};\mathbf{A}!\mathtt{sn};\mathbf{A}?\mathtt{snTk}$$

- The process $P^{\mathbf{R}}$ is $\mathbf{S}!\mathtt{dt};(\, P_1^{\mathbf{R}} +_{\{C\}} \mathbf{S}?\mathtt{dnPl} \,)$ where

$$P_1^{\mathbf{R}} = \mathbf{S}?\mathtt{alDt};(\, P_2^{\mathbf{R}} \oplus_{\{\overline{C'}\}} \mathbf{S}!\mathtt{no} \,) \qquad P_2^{\mathbf{R}} = \mathbf{S}!\mathtt{yes};\mathbf{S}?\mathtt{dnMn};\mathbf{S}!\mathtt{htl}$$

- The process $P^{\mathbf{A}}$ is $(P_1^{\mathbf{A}} \mid P_2^{\mathbf{A}});_{\{C,C'\}} \sum_{i\in\{3,4,5\}} P_i^{\mathbf{A}}$ where

$$P_1^{\mathbf{A}} = \mathbf{S}?\mathtt{fxDt};\mathbf{S}!\mathtt{frPl} \qquad P_2^{\mathbf{A}} = \mathbf{S}?\mathtt{flDt};\mathbf{S}!\mathtt{frMn}$$
$$P_3^{\mathbf{A}} = \mathbf{S}?\mathtt{dbMn};\mathbf{S}!\mathtt{dTkMn} \qquad P_4^{\mathbf{A}} = \mathbf{S}?\mathtt{sn};\mathbf{S}!\mathtt{snTk} \qquad P_5^{\mathbf{A}} = \mathbf{S}?\mathtt{dbPl};\mathbf{S}!\mathtt{dTkPl}$$

In the computation below, we denote with $\widehat{P}$ the process P entirely marked with hats.

$$\begin{array}{rl}
\mathbb{N}_{in} \xrightarrow{\mathbf{R}\,\mathtt{dt}\,\mathbf{S}} & \mathbf{R}[\![\, \widehat{\mathbf{S}!\mathtt{dt}};(P_1^{\mathbf{R}} +_{\{C\}} \mathbf{S}?\mathtt{dnPl}) \,]\!] \parallel \mathbf{S}[\![\, \widehat{\mathbf{R}?\mathtt{dt}};(P_1^{\mathbf{S}} \mid P_2^{\mathbf{S}})\,;\,(P_3^{\mathbf{S}} \oplus_{\{\overline{C}\}} P_4^{\mathbf{S}}) \,]\!] \parallel \mathbf{A}[\![\, P^{\mathbf{A}} \,]\!] \\
\longrightarrow^* & \mathbf{R}[\![\, \widehat{\mathbf{S}!\mathtt{dt}};(P_1^{\mathbf{R}} +_{\{C\}} \mathbf{S}?\mathtt{dnPl}) \,]\!] \parallel \mathbf{S}[\![\, F^{\mathbf{S}}\,;\,(P_3^{\mathbf{S}} \oplus_{\{\overline{C}\}} P_4^{\mathbf{S}}) \,]\!] \parallel \mathbf{A}[\![\, Q^{\mathbf{A}} \,]\!] \\
& \text{where } F^{\mathbf{S}} = \widehat{\mathbf{R}?\mathtt{dt}};(\widehat{P_1^{\mathbf{S}}} \mid \widehat{P_2^{\mathbf{S}}}),\ Q^{\mathbf{A}} = (\widehat{P_1^{\mathbf{A}}} \mid \widehat{P_2^{\mathbf{A}}});_{\{C,C'\}} \sum_{i\in\{3,4,5\}} P_i^{\mathbf{A}} \\
\xrightarrow{\mathbf{S}\,\mathtt{alDt}\,\mathbf{R}} & \mathbf{R}[\![\, \widehat{\mathbf{S}!\mathtt{dt}};(\, \widehat{\mathbf{S}?\mathtt{alDt}};(\, P_2^{\mathbf{R}} \oplus_{\{\overline{C'}\}} \mathbf{S}!\mathtt{no} \,) +_{\{C\}} \mathbf{S}?\mathtt{dnPl} \,) \,]\!] \parallel \\
& \mathbf{S}[\![\, F^{\mathbf{S}}\,;\,(\widehat{\mathbf{R}!\mathtt{alDt}};(P_5^{\mathbf{S}} +_{\{C'\}} P_6^{\mathbf{S}}) \oplus_{\{\overline{C}\}} P_4^{\mathbf{S}}) \,]\!] \parallel \mathbf{A}[\![\, Q^{\mathbf{A}} \,]\!] \\
\longrightarrow^* & \mathbf{R}[\![\, \widehat{\mathbf{S}!\mathtt{dt}};(\, \widehat{\mathbf{S}?\mathtt{alDt}};(\, \widehat{\mathbf{S}!\mathtt{yes}};\mathbf{S}?\mathtt{dnMn};\mathbf{S}!\mathtt{htl} \oplus_{\{\overline{C'}\}} \mathbf{S}!\mathtt{no} \,) +_{\{C\}} \mathbf{S}?\mathtt{dnPl} \,) \,]\!] \parallel \\
& \mathbf{S}[\![\, F^{\mathbf{S}}\,;\,(\widehat{\mathbf{R}!\mathtt{alDt}};(\widehat{\mathbf{R}?\mathtt{yes}};\mathbf{A}!\mathtt{dbMn};P_7^{\mathbf{S}} +_{\{C'\}} P_6^{\mathbf{S}}) \oplus_{\{\overline{C}\}} P_4^{\mathbf{S}}) \,]\!] \parallel \mathbf{A}[\![\, Q^{\mathbf{A}} \,]\!] \\
\overset{C}{\circlearrowright} & \mathbf{R}[\![\, \widehat{\mathbf{S}!\mathtt{dt}};(P_1^{\mathbf{R}} +_{\{C\}} \mathbf{S}?\mathtt{dnPl}\,) \,]\!] \parallel \mathbf{S}[\![\, F^{\mathbf{S}}\,;\,P_4^{\mathbf{S}} \,]\!] \parallel \mathbf{A}[\![\, Q^{\mathbf{A}} \,]\!]
\end{array}$$

Note that the last network differs from that in Line 2 only by the absence of process $P_3^{\mathbf{S}}$.

4 Global Types and Session Types

A *multiparty session* is a series of communications among a fixed number of participants [18], which follows a predefined protocol specified by a *global type*.

We use γ to denote either the empty set or a singleton made of a checkpoint label:

$$\gamma ::= \emptyset \quad | \quad \{C\}$$

Sets γ will be associated with choices in global types.

Let $\alpha^{\mathsf{p}} \in \{\mathsf{p} \xrightarrow{\lambda} \mathsf{q} \mid \mathsf{q} \in \mathsf{Part}, \lambda \in \mathsf{Msg}\}$ denote an *atomic communication* with sender p. An atomic communication can bear a hat, in notation $\widehat{\alpha^{\mathsf{p}}}$, in which case it represents an executed or *past* communication. The symbol $\widehat{\alpha^{\mathsf{p}}}$ stands for either α^{p} or $\widehat{\alpha^{\mathsf{p}}}$.

Global types G are built from choices among atomic communications with the same sender, with the constructs of parallel composition, sequential composition, and recursion. A

global type K specifies an interaction that is still to start. On the opposite, a global type H specifies a completely executed interaction, in which there is one path entirely marked with hats. A general global type G specifies a partially executed interaction, whose executed part (history) is specified by subterms H, while the parts that have been discarded in choices or remain to be executed are specified by subterms K.

▶ **Definition 4.** *Global types* G are defined by:

$$\begin{array}{lcl} \mathsf{K} & ::= & {}_{\gamma}\boxplus_{i\in I}\ \alpha_i^{\mathsf{p}};\mathsf{K}_i \mid \mathsf{K}\parallel\mathsf{K} \mid \mathsf{K};\mathsf{K} \mid \mu\mathbf{t}.\mathsf{K} \mid \mathbf{t} \mid \mathsf{Skip} \\ \mathsf{H} & ::= & {}_{\gamma}(\widehat{\boxplus}_{i\in I}\,\alpha_i^{\mathsf{p}};\mathsf{K}_i\ \widehat{\boxplus}\ \widehat{\alpha^{\mathsf{p}}}\ ;\mathsf{H}) \mid \mathsf{H}\parallel\mathsf{H} \mid \mathsf{H};\mathsf{H} \mid \mathsf{Skip} \\ \mathsf{G} & ::= & \mathsf{K} \mid {}_{\gamma}(\widehat{\boxplus}_{i\in I}\,\alpha_i^{\mathsf{p}};\mathsf{K}_i\ \widehat{\boxplus}\ \widehat{\alpha^{\mathsf{p}}}\ ;\mathsf{G}) \mid \mathsf{G}\parallel\mathsf{G} \mid \mathsf{G};\mathsf{K} \mid \mathsf{H};\mathsf{G} \end{array}$$

The choice operator $\boxplus$ is n-ary, commutative and idempotent. We use $\widetilde{\boxplus}$ for either $\boxplus$ or $\widehat{\boxplus}$. In the binary case we write $\widetilde{\alpha_1^{\mathsf{p}}};\mathsf{G}\ {}_{\gamma}\widetilde{\boxplus}\ \widetilde{\alpha_2^{\mathsf{p}}};\mathsf{G}'$. The notation ${}_{\gamma}(\widehat{\boxplus}_{i\in I}\,\alpha_i^{\mathsf{p}};\mathsf{K}_i\ \widehat{\boxplus}\ \widehat{\alpha^{\mathsf{p}}}\ ;\mathsf{G})$ stands for a choice where the branch initiating with $\widehat{\alpha^{\mathsf{p}}}$ has started to be executed. By abuse of notation, we will write ${}_{\gamma}\widetilde{\boxplus}_{i\in I}\,\widetilde{\alpha_i^{\mathsf{p}}};\mathsf{G}_i$ for either ${}_{\gamma}\boxplus_{i\in I}\ \alpha_i^{\mathsf{p}};\mathsf{G}_i$ or ${}_{\gamma}(\widehat{\boxplus}_{i\in I\setminus\{j\}}\,\alpha_i^{\mathsf{p}};\mathsf{G}_i\ \widehat{\boxplus}\ \widehat{\alpha_j^{\mathsf{p}}}\ ;\mathsf{G}_j)$. When the checkpoint set γ associated with a choice is empty, we omit it. We call *rooted interaction* a subterm $\widetilde{\alpha^{\mathsf{p}}};\mathsf{G}$. Hence a type ${}_{\gamma}\widetilde{\boxplus}_{i\in I}\,\widetilde{\alpha_i^{\mathsf{p}}};\mathsf{G}_i$ is a choice among a number of rooted interactions with the same sender, at most one of which bears a hat. If I is a singleton we write $\widetilde{\alpha^{\mathsf{p}}};\mathsf{G}$. So $\widetilde{\alpha^{\mathsf{p}}};\mathsf{G}$ can denote either a rooted interaction or a choice with a single branch (the context will disambiguate if needed). We use $\mathsf{pa}(\mathsf{G})$ to denote *the set of participants of* G, namely all p,q such that $\mathsf{p}\xrightarrow{\lambda}\mathsf{q}$ occurs in G.

As for processes, parallel and sequential composition are associative, with neutral element Skip, parallel composition is commutative, and recursion is guarded, sequential, and treated equi-recursively. The operators have the following precedence: ';', '$\boxplus$', '$\parallel$'.

Session types are projections of global types onto participants. They represent the contributions of individual participants to the session. The projection of a choice yields a union for the choice leader and intersections for the receivers. Checkpoint labels of global types are preserved by the projection onto session types, and the checkpoint label of the choice leader is distinguished by overlining it.

We now define *session pre-types*, which are a superset of session types. In this paper, session types will only be defined as projections of well-formed global types, see Definition 7. (There is no circularity since global type well-formedness does not depend on session types.) Session pre-types are obtained from processes by replacing external and internal choices with intersections and unions, X with $\mathbf{t}$ and skip with Skip, with similar conventions.

▶ **Definition 5.** *Session pre-types* are defined by:

$$\mathsf{T} \quad ::= \quad \widetilde{\bigwedge}_{i\in I}\widetilde{\pi_i};\mathsf{T}_i \mid \widetilde{\bigvee}_{i\in I}\widetilde{\pi_i};\mathsf{T}_i \mid {}_{\Delta}\mathsf{T} \mid \mathsf{T}\mid\mathsf{T} \mid \mathsf{T};\mathsf{T} \mid \mu\mathbf{t}.\mathsf{T} \mid \mathbf{t} \mid \mathsf{Skip}$$

We call an intersection $\widetilde{\bigwedge}_{i\in I}\widetilde{\mathsf{p}_i?\lambda_i};\mathsf{T}_i$ *non-ambiguous* if its initial inputs are all distinct, namely if $i\neq j$ implies either $\mathsf{p}_i\neq\mathsf{p}_j$ or $\lambda_i\neq\lambda_j$. Projectability will require non-ambiguity of intersections. Our definition of projection (see Fig.3) is more liberal than in other session calculi, since we have an extended syntax for global types and we want to maximise the set of global types that are projectable. In particular, we allow equal messages between the same pair of participants in choices (this is usually forbidden [12, 28]). In any case, we want projection to ensure that in a global choice, the choice leader makes the decision and all the other participants act accordingly. We do so by requiring that, for any participant except the choice leader, the set of projections of the choice branches on that participant, say $\{\mathsf{T}_i \mid i\in I\}$, be consistent, i.e., the *join of the types in the set,* $\bigsqcup_{i\in I}\mathsf{T}_i$, be defined.

The join $\bigsqcup_{i\in I}\mathsf{T}_i$ is a partial operator, which checks that the T_i's are compatible behaviours and then combines them into a single session type. (In the definition of join we may assume

that all the T_i's are projections of global types.) Intuitively, $\bigsqcup_{i\in I}\mathsf{T}_i$ is defined if the concerned participant either has the same behaviour in all T_i, or, if this is not the case, if she receives a message that "notifies" her about the chosen T_i before she starts differentiating her behaviour. If one of the T_i's is an intersection of input behaviours, then so must be all the other T_i's.

To join intersection types, we define an auxiliary operator $\biguplus$, which takes a non-ambiguous intersection $\widetilde{\bigwedge}_{i\in I}\widetilde{\mathsf{p}_i?\lambda_i};\mathsf{T}_i$ and a session type $\widetilde{\mathsf{p}?\lambda};\mathsf{T}$, and combines them, if possible:

$$(\widetilde{\bigwedge}_{i\in I}\widetilde{\mathsf{p}_i?\lambda_i};\mathsf{T}_i)\biguplus\widetilde{\mathsf{p}?\lambda};\mathsf{T}=\begin{cases}\widetilde{\bigwedge}_{i\in I}\widetilde{\mathsf{p}_i?\lambda_i};\mathsf{T}_i\ \widetilde{\wedge}\ \widetilde{\mathsf{p}?\lambda};\mathsf{T} & \text{if } \mathsf{p}_i\neq\mathsf{p} \text{ or } \lambda_i\neq\lambda \text{ for all } i\in I\\ \widetilde{\bigwedge}_{i\in I\setminus\{j\}}\widetilde{\mathsf{p}_i?\lambda_i};\mathsf{T}_i\ \widetilde{\wedge}\ (\widetilde{\mathsf{p}_j?\lambda_j}\ \widetilde{\cup}\ \widetilde{\mathsf{p}?\lambda});\mathsf{T}\bigsqcup\mathsf{T}_j & \text{if } \mathsf{p}_j=\mathsf{p}\\ & \text{and } \lambda_j=\lambda\ (j\in I)\end{cases}$$

where $\widehat{\mathsf{p}?\lambda}\ \widetilde{\cup}\ \mathsf{p}?\lambda=\mathsf{p}?\lambda\ \widetilde{\cup}\ \widehat{\mathsf{p}?\lambda}=\widehat{\mathsf{p}?\lambda}$ and $\mathsf{p}?\lambda\ \widetilde{\cup}\ \mathsf{p}?\lambda=\mathsf{p}?\lambda$ and the obtained intersections have hats if one of the input behaviours has a hat. If the prefix $\mathsf{p}?\lambda$ is different from all the $\mathsf{p}_i?\lambda_i$, then $\biguplus$ produces the intersection of the two types. In this case, receiving message λ from p amounts to being notified of the choice of branch T. If instead $\mathsf{p}?\lambda=\mathsf{p}_j?\lambda_j$ for some $j\in I$, then we try to combine the types $\widetilde{\mathsf{p}?\lambda};\mathsf{T}$ and $\widetilde{\mathsf{p}_j?\lambda_j};\mathsf{T}_j$ by factoring out their common prefix and producing $\widetilde{\mathsf{p}?\lambda};\mathsf{T}\bigsqcup\mathsf{T}_j$ (where $\mathsf{p}?\lambda$ has a hat if one of the two building prefixes has a hat), so that the resulting intersection is again non-ambiguous. If $\mathsf{T}\bigsqcup\mathsf{T}_j$ is defined, then T and T_j will be discriminated later if they are different, see G_1 in Example 6.

To define $\biguplus$ on two intersection types, we just iterate the above definition on the members of one of the intersections. In a similar way, we can extend $\biguplus$ to a set of intersection types.

We may now formally define the join of a set of session types.

$$\bigsqcup_{i\in I}\mathsf{T}_i=\begin{cases}{}_\Delta\biguplus_{i\in I,j\in J_i}\widetilde{\mathsf{p}_{i,j}?\lambda_{i,j}};\mathsf{T}_{i,j} & \text{if } \mathsf{T}_i={}_{\Delta_i}\widetilde{\bigwedge}_{j\in J_i}\widetilde{\mathsf{p}_{i,j}?\lambda_{i,j}};\mathsf{T}_{i,j} \text{ for all } i\in I\\ & \text{and } \Delta=\bigcup_{i\in I}\Delta_i\\ \mathsf{T};\bigsqcup_{i\in I}\mathsf{T}'_i & \text{if } \mathsf{T}_i=\mathsf{T};\mathsf{T}'_i \text{ for all } i\in I\\ \mathsf{Skip} & \text{if } \mathsf{T}_i=\mathsf{Skip} \text{ for all } i\in I\end{cases}$$

The definitions of $\bigsqcup_{i\in I}\mathsf{T}_i$ and $\biguplus$ are mutually recursive. Let us examine the three clauses. If all T_i's are intersections of session types, then we combine them with $\biguplus$ as explained above. The checkpoint set of the resulting intersection is the union of the checkpoint sets of the T_i's. If all T_i's start with the same interaction T, and their continuations T'_i's are consistent, then we factor out T and we produce the session type T followed by the join of the T'_i's.
If some T_i is Skip, it means that the participant terminates in the branch T_i. Then it must terminate in all branches, so all T_i's must be Skip.
The join is not defined between intersections and unions, nor between unions with different prefixes. Our join extends that of [12], to deal with parallel and sequential composition.

To define projection we need one last auxiliary operator, the prefixing of a session pre-type T by a set γ of at most one checkpoint label (notation $\gamma\lfloor\mathsf{T}\rfloor$), defined by:

$$\gamma\lfloor\widetilde{\bigwedge}_{i\in I}\widetilde{\pi_i};\mathsf{T}_i\rfloor=\gamma\widetilde{\bigwedge}_{i\in I}\widetilde{\pi_i};\mathsf{T}_i\qquad\gamma\lfloor\widetilde{\bigvee}_{i\in I}\widetilde{\pi_i};\mathsf{T}_i\rfloor=\gamma\widetilde{\bigvee}_{i\in I}\widetilde{\pi_i};\mathsf{T}_i\qquad\gamma\lfloor{}_\Delta\mathsf{T}\rfloor={}_{\gamma\cup\Delta}\mathsf{T}\text{ if }\gamma\cap\Delta=\emptyset$$
$$\gamma\lfloor\mathsf{T}_0\mid\mathsf{T}_1\rfloor=\gamma\lfloor\mathsf{T}_0\rfloor\mid\gamma\lfloor\mathsf{T}_1\rfloor\quad\gamma\lfloor\mathsf{T}_0;\mathsf{T}_1\rfloor=\gamma\lfloor\mathsf{T}_0\rfloor;\mathsf{T}_1\quad\gamma\lfloor\mu\mathbf{t}.\mathsf{T}'\rfloor=\mu\mathbf{t}.\gamma\lfloor\mathsf{T}'\rfloor\quad\gamma\lfloor\mathsf{Skip}\rfloor=\mathsf{Skip}$$

Prefixing adds γ to the first (possibly empty) set Δ found in T. Intuitively, the checkpoint labels that are spread along successive choices in the global type may get grouped together on a single local choice when projected on participants. We do not need to define $\gamma\lfloor\mathbf{t}\rfloor$ since recursion is guarded. The condition in the clause for $\gamma\lfloor{}_\Delta\mathsf{T}\rfloor$ is needed to avoid checkpoint labels which can never be used. This condition rules out for instance the global type:

$$(\mathsf{p}\xrightarrow{\lambda_1}\mathsf{q};\mathsf{p}\xrightarrow{\lambda'_1}\mathsf{r};(\mathsf{p}\xrightarrow{\lambda_2}\mathsf{q};\mathsf{p}\xrightarrow{\lambda'_2}\mathsf{r}\ {}_{\{C\}}\boxplus\ \mathsf{p}\xrightarrow{\lambda_3}\mathsf{q};\mathsf{p}\xrightarrow{\lambda'_3}\mathsf{r}))\ {}_{\{C\}}\boxplus\ \mathsf{p}\xrightarrow{\lambda_4}\mathsf{q};\mathsf{p}\xrightarrow{\lambda'_4}\mathsf{r}$$

in which no backward reduction could return to the innermost occurrence of C. This is due to the condition on evaluation contexts in rules [CTBA] and [CTBP].

The projection of global types uses the projections of rooted interactions, see the first

$$(\widetilde{\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{G}}) \upharpoonright \mathsf{p} = \widetilde{\mathsf{q}!\lambda}; \mathsf{G} \upharpoonright \mathsf{p} \quad (\widetilde{\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{G}}) \upharpoonright \mathsf{q} = \widetilde{\mathsf{p}?\lambda}; \mathsf{G} \upharpoonright \mathsf{q} \quad (\widetilde{\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{G}}) \upharpoonright \mathsf{r} = \mathsf{G} \upharpoonright \mathsf{r} \text{ , if } \mathsf{r} \neq \mathsf{p}, \mathsf{q}$$

$$(\ {}_{\gamma}\widetilde{\boxplus}_{i \in I}\, \widetilde{\alpha_i^{\mathsf{p}}}; \mathsf{G}_i\ ) \upharpoonright \mathsf{r} = \begin{cases} \overline{\gamma}\widetilde{\bigvee}_{i \in I}(\ \widetilde{\alpha_i^{\mathsf{p}}}; \mathsf{G}_i\ ) \upharpoonright \mathsf{r} & \text{if } \mathsf{r} = \mathsf{p} \\ \gamma \lfloor \bigsqcup_{i \in I}(\ \widetilde{\alpha_i^{\mathsf{p}}}; \mathsf{G}_i\ ) \upharpoonright \mathsf{r} \rfloor & \text{otherwise} \end{cases}$$

Figure 3 Projection of global types onto participants.

line of Fig.3. The projection of a choice is a union for the sending participant, and it is otherwise computed as the join of the projections on the branches. With $\overline{\gamma}$ we denote $\{\overline{C}\}$ if $\gamma = \{C\}$ and $\emptyset$ if $\gamma = \emptyset$. The join operation can be undefined, and therefore also the projection of global types can be undefined. Since $\mathsf{Skip} \bigsqcup \mathsf{T}$ is defined only if $\mathsf{T} = \mathsf{Skip}$, the definition of projection ensures that all the branches of a choice ${}_{\gamma}\widetilde{\boxplus}_{i \in I}\, \widetilde{\alpha_i^{\mathsf{p}}}; \mathsf{G}_i$ have the same participants, i.e., $\mathsf{pa}(\widetilde{\alpha_i^{\mathsf{p}}}; \mathsf{G}_i) = \mathsf{pa}(\widetilde{\alpha_j^{\mathsf{p}}}; \mathsf{G}_j)$ for all $i, j \in I$. The omitted cases in Fig.3 are either standard (recursion and Skip) or homomorphic projections (parallel and sequential composition). Notice that projection respects hats and checkpoint labels.

▶ **Example 6.** Let $\mathsf{G}_1 = (\ \widehat{\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{p} \xrightarrow{\lambda_1} \mathsf{q}}; \mathsf{q} \xrightarrow{\lambda_2} \mathsf{r}\ ) \mathbin{\widehat{\boxplus}} (\ \mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{p} \xrightarrow{\lambda_3} \mathsf{q}; \mathsf{q} \xrightarrow{\lambda_4} \mathsf{r}\ )$. The projections of G_1 onto its participants are: $\mathsf{G}_1 \upharpoonright \mathsf{p} = (\ \widehat{\mathsf{q}!\lambda}; \widehat{\mathsf{q}!\lambda_1}\ ) \mathbin{\widehat{\vee}} (\ \mathsf{q}!\lambda; \mathsf{q}!\lambda_3\ )$, $\mathsf{G}_1 \upharpoonright \mathsf{q} = \widehat{\mathsf{p}?\lambda}; (\ (\ \widehat{\mathsf{p}?\lambda_1}; \mathsf{r}!\lambda_2\ ) \mathbin{\widehat{\wedge}} (\ \mathsf{p}?\lambda_3; \mathsf{r}!\lambda_4\ )\ )$ and $\mathsf{G}_1 \upharpoonright \mathsf{r} = \mathsf{q}?\lambda_2 \wedge \mathsf{q}?\lambda_4$. Even though q receives the same message λ from p in the two branches of the choice, before sending two different messages to r it receives a second message from p, which identifies the chosen branch. Also the global type $\mathsf{G}_2 = (\ \mathsf{p} \xrightarrow{\lambda_1} \mathsf{q}; \mathsf{r} \xrightarrow{\lambda_2} \mathsf{q}\ ) \boxplus (\ \mathsf{p} \xrightarrow{\lambda_3} \mathsf{q}; \mathsf{r} \xrightarrow{\lambda_2} \mathsf{q}\ )$ is projectable, since r sends the same message λ_2 in both branches.

Instead, the global type $\mathsf{G}_3 = (\ \widehat{\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{q} \xrightarrow{\lambda_1} \mathsf{r}}\ ) \mathbin{\widehat{\boxplus}} (\ \mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{q} \xrightarrow{\lambda_2} \mathsf{r}\ )$ is not projectable on the participant q: in fact $(\ \widehat{\mathsf{p}?\lambda}; \mathsf{r}!\lambda_1\ ) \bigsqcup (\ \mathsf{p}?\lambda; \mathsf{r}!\lambda_2\ )$ is not defined, since it yields $\widehat{\mathsf{p}?\lambda}; (\mathsf{r}!\lambda_1 \bigsqcup \mathsf{r}!\lambda_2)$ and $\mathsf{r}!\lambda_1 \bigsqcup \mathsf{r}!\lambda_2$ is not defined. This is justified since q should send two different messages to r in the two branches, while no previous communication identifies the chosen branch.
The next example features a checkpoint label set containing both a simple label and an overlined one. Let $\mathsf{G}_4 = \mathsf{p} \xrightarrow{\lambda_1} \mathsf{q}; \mathsf{p} \xrightarrow{\lambda_1'} \mathsf{r}; \mathsf{G}_5\ {}_{\{C'\}} \boxplus\ \mathsf{p} \xrightarrow{\lambda_2} \mathsf{q}; \mathsf{p} \xrightarrow{\lambda_2'} \mathsf{r}; \mathsf{G}_5$ where $\mathsf{G}_5 = \mathsf{s} \xrightarrow{\lambda_3} \mathsf{p}; \mathsf{s} \xrightarrow{\lambda_3'} \mathsf{r}\ {}_{\{C\}} \boxplus\ \mathsf{s} \xrightarrow{\lambda_4} \mathsf{p}; \mathsf{s} \xrightarrow{\lambda_4'} \mathsf{r}$. Then the projection of G_4 onto s is $\mathsf{G}_4 \upharpoonright \mathsf{s} = \mathsf{p}!\lambda_3\ \mathsf{r}!\lambda_3'\ {}_{\{C', \overline{C}\}}\vee\ \mathsf{p}!\lambda_4\ \mathsf{r}!\lambda_4'$.

Let us now look back at the RSA protocol. Its global type G, which may be seen as a syntactic description of the graph in Fig.1a, is the following:

$$\mathsf{G} = \mathbf{R} \xrightarrow{\mathtt{dt}} \mathbf{S}\ ; (\mathsf{G}_1 \parallel \mathsf{G}_2); (\mathsf{G}_3\ {}_{\{C\}}\boxplus\ \mathsf{G}_4) \text{ where}$$

$$\mathsf{G}_1 = \mathbf{S} \xrightarrow{\mathtt{fxDt}} \mathbf{A}; \mathbf{A} \xrightarrow{\mathtt{frPl}} \mathbf{S} \qquad \mathsf{G}_2 = \mathbf{S} \xrightarrow{\mathtt{flDt}} \mathbf{A}; \mathbf{A} \xrightarrow{\mathtt{frMn}} \mathbf{S}$$

$$\mathsf{G}_3 = \mathbf{S} \xrightarrow{\mathtt{alDt}} \mathbf{R}; (\mathsf{G}_5\ {}_{\{C'\}}\boxplus\ \mathsf{G}_6) \qquad \mathsf{G}_4 = \mathbf{S} \xrightarrow{\mathtt{dbPl}} \mathbf{A}\ ; \mathbf{A} \xrightarrow{\mathtt{dTkPl}} \mathbf{S}; \mathbf{S} \xrightarrow{\mathtt{dnPl}} \mathbf{R}$$

$$\mathsf{G}_5 = \mathbf{R} \xrightarrow{\mathtt{yes}} \mathbf{S}\ ; \mathbf{S} \xrightarrow{\mathtt{dbMn}} \mathbf{A}; \mathbf{A} \xrightarrow{\mathtt{dTkMn}} \mathbf{S}; \mathbf{S} \xrightarrow{\mathtt{dnMn}} \mathbf{R}; \mathbf{R} \xrightarrow{\mathtt{htl}} \mathbf{S} \quad \mathsf{G}_6 = \mathbf{R} \xrightarrow{\mathtt{no}} \mathbf{S}\ ; \mathbf{S} \xrightarrow{\mathtt{sn}} \mathbf{A}; \mathbf{A} \xrightarrow{\mathtt{snTk}} \mathbf{S}$$

The projection of G on the participant $\mathbf{S}$ ($\mathsf{T}^{\mathbf{S}}$) has the same structure as G, since $\mathbf{S}$ is involved in all communications of the protocol:

$$\mathsf{T}^{\mathbf{S}} = \mathbf{R}?\mathtt{dt}; (\mathsf{T}_1^{\mathbf{S}} \mid \mathsf{T}_2^{\mathbf{S}})\ ; (\mathsf{T}_3^{\mathbf{S}}\ {}_{\{\overline{C}\}}\vee\ \mathsf{T}_4^{\mathbf{S}}) \quad \text{where}$$

$$\mathsf{T}_1^{\mathbf{S}} = \mathbf{A}!\mathtt{fxDt}; \mathbf{A}?\mathtt{frPl} \qquad \mathsf{T}_2^{\mathbf{S}} = \mathbf{A}!\mathtt{flDt}; \mathbf{A}?\mathtt{frMn}$$

$$\mathsf{T}_3^{\mathbf{S}} = \mathbf{R}!\mathtt{alDt}; (\mathsf{T}_5^{\mathbf{S}}\ {}_{\{C'\}}\wedge\ \mathsf{T}_6^{\mathbf{S}}) \qquad \mathsf{T}_4^{\mathbf{S}} = \mathbf{A}!\mathtt{dbPl}; \mathbf{A}?\mathtt{dTkPl}; \mathbf{R}!\mathtt{dnPl}$$

$$\mathsf{T}_5^{\mathbf{S}} = \mathbf{R}?\mathtt{yes}; \mathbf{A}!\mathtt{dbMn}; \mathbf{A}?\mathtt{dTkMn}; \mathbf{R}!\mathtt{dnMn}; \mathbf{R}?\mathtt{htl} \qquad \mathsf{T}_6^{\mathbf{S}} = \mathbf{R}?\mathtt{no}; \mathbf{A}!\mathtt{sn}; \mathbf{A}?\mathtt{snTk}$$

On the other hand, the projections of G on $\mathbf{A}$ and $\mathbf{R}$ are simpler, and are not given here.

In the presence of sequentialisation and recursion, projectability of global types is not

enough to ensure *starvation freedom* in communication protocols. For example, the global type $\mu \mathbf{t}.\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathbf{t}; \mathsf{p} \xrightarrow{\lambda'} \mathsf{r}$ exhibits starvation, since the participant r will indefinitely wait for message λ' from participant p. This problem can be avoided by requiring that the communications which follow a loop involve only participants who are also communicating in the body of the loop. We formalise this by the predicate *sequentially well-formed* on global types. This predicate applied to $\mathsf{G}; \mathsf{G}'$ requires that the participants of G' be contained in the set of *allowed followers* of G, denoted by $\mathsf{af}(\mathsf{G})$ and defined by

$$\mathsf{af}(\gamma \widetilde{\boxplus}_{i\in I} \widetilde{\alpha_i^{\mathsf{p}}}; \mathsf{G}_i) = \bigcap_{i\in I} \mathsf{af}(G_i) \qquad \mathsf{af}(\mathsf{G} \parallel \mathsf{G}') = \mathsf{af}(\mathsf{G}) \cup \mathsf{af}(\mathsf{G}') \qquad \mathsf{af}(\mathsf{G}; \mathsf{G}') = \mathsf{af}(\mathsf{G}) \cap \mathsf{af}(\mathsf{G}')$$

$$\mathsf{af}(\mu \mathbf{t}.\mathsf{G}) = \begin{cases} \mathsf{pa}(\mathsf{G}) & \text{if } \mathbf{t} \in \mathsf{G} \\ \mathsf{af}(\mathsf{G}) & \text{otherwise} \end{cases} \qquad \mathsf{af}(\mathsf{Skip}) = \mathsf{Part}$$

In the definition of af for parallel composition of global types, the union is justified by the example $(\mu \mathbf{t}.\mathsf{p} \xrightarrow{\lambda_1} \mathsf{q}; \mathbf{t} \parallel \mu \mathbf{t}.\mathsf{r} \xrightarrow{\lambda_2} \mathsf{s}; \mathbf{t}); \mathsf{p} \xrightarrow{\lambda_3} \mathsf{r}$.

In order to avoid deadlocks, we need to impose some restrictions on the use of checkpoint labels and of messages in parallel communications. The same checkpoint label should not occur in two parallel global types. For suppose we allowed the global type:

$$(\mathsf{p} \xrightarrow{\lambda_1} \mathsf{q}\,;\ \mathsf{p} \xrightarrow{\lambda_1'} \mathsf{r}\ {}_{\{C\}}\!\boxplus\ \mathsf{p} \xrightarrow{\lambda_2} \mathsf{q}\,;\ \mathsf{p} \xrightarrow{\lambda_2'} \mathsf{r}) \parallel (\mathsf{p} \xrightarrow{\lambda_3} \mathsf{q}; \mathsf{p} \xrightarrow{\lambda_3'} \mathsf{r}\ {}_{\{C\}}\!\boxplus\ \mathsf{p} \xrightarrow{\lambda_4} \mathsf{q}; \mathsf{p} \xrightarrow{\lambda_4'} \mathsf{r})$$

Here there are two parallel interactions, both involving participants p, q and r. Assume that message λ_1 has been exchanged in the first interaction, and message λ_3 has been exchanged in the second. Now participant p may want to roll back to the choice labelled by $\{C\}$ in the first interaction. Then participant q should roll back to the same choice, while the above type would allow her to roll back to the choice labelled by $\{C\}$ in the second interaction.

As regards messages, we already observed that we allow different branches of a choice to have equal senders, equal messages and equal receivers. Our definition of projection ensures that the behaviour of a participant is either independent of the chosen branch or identifiable through some communications. For parallel global types, instead, we require that different branches do not have equal senders, equal messages and equal receivers. Indeed, if we allowed two parallel communications with equal messages between the same pair of participants, then we could have a global type $\mathsf{p} \xrightarrow{\lambda} \mathsf{q}\,;\ \mathsf{G}_1 \parallel \mathsf{p} \xrightarrow{\lambda} \mathsf{q}\,;\ \mathsf{G}_2$, and processes implementing participants p and q could "cross" their communications.

A global type meeting the above two requirements on messages and checkpoint labels is called respectively *message well formed and checkpoint label well formed*.

Last but not least, following [7, 26] we require that the order of communications prescribed by a global type be witnessed by at least one of the session participants. This excludes, for instance, the global type $\mathsf{p} \xrightarrow{\lambda} \mathsf{q}; \mathsf{r} \xrightarrow{\lambda'} \mathsf{s}$, since the two communications have disjoint sets of participants, and therefore they should be independent. The correct global type representing this situation is $\mathsf{p} \xrightarrow{\lambda} \mathsf{q} \parallel \mathsf{r} \xrightarrow{\lambda'} \mathsf{s}$. This condition is necessary for a global type to be *implementable* by a collection of processes, and we refer to [7] for further justifications of this choice. It is easy to formalise this requirement as a well-formedness condition on the set of traces generated by global types, defined in the standard way.

To sum up, we define well-formedness of global types as follows.

▶ **Definition 7** (Well-formed global types). *A global type is well formed* when:

1. Its projections are defined on all participants;
2. It is sequentially well formed, message well formed and checkpoint label well formed;
3. Its set of traces is well formed.

In the following we will consider only well-formed global types.

$$\frac{\Gamma \vdash P_i : \mathsf{T}_i \ (i \in I)}{\Gamma \vdash {}_{\delta}\textstyle\sum_{i\in I} \pi_i; P_i : {}_{\delta}\bigwedge_{i\in I} \pi_i; \mathsf{T}_i} \ [\text{T-ExtCh}] \qquad \frac{\Gamma \vdash P_i : \mathsf{T}_i \ (i \in I)}{\Gamma \vdash {}_{\Delta}\bigoplus_{i\in I} \pi_i; P_i : {}_{\Delta}\bigvee_{i\in I} \pi_i; \mathsf{T}_i} \ [\text{T-IntCh}]$$

$$\frac{\Gamma \vdash P_i : \mathsf{T}_i \ (i \in I) \quad \Gamma \vdash P : \mathsf{T}}{\Gamma \vdash {}_{\delta}\widehat{\textstyle\sum}_{i\in I}\pi_i; P_i \mathbin{\widehat{+}} \widehat{\pi}; P : {}_{\delta}\widehat{\bigwedge}_{i\in I}\pi_i; \mathsf{T}_i \mathbin{\widehat{\wedge}} \widehat{\pi}; \mathsf{T}} \ [\text{T-ExtCh-RT}]$$

$$\frac{\Gamma \vdash P_i : \mathsf{T}_i \ (i \in I) \quad \Gamma \vdash P : \mathsf{T}}{\Gamma \vdash {}_{\Delta}\widehat{\bigoplus}_{i\in I}\pi_i; P_i \mathbin{\widehat{\oplus}} \widehat{\pi}; P : {}_{\Delta}\widehat{\bigvee}_{i\in I}\pi_i; \mathsf{T}_i \mathbin{\widehat{\vee}} \widehat{\pi}; \mathsf{T}} \ [\text{T-IntCh-RT}]$$

$$\frac{\vdash P_i : \mathsf{G}\upharpoonright \mathsf{p}_i \quad (1 \le i \le n) \quad \mathsf{pa}(\mathsf{G}) \subseteq \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}}{\vdash \mathsf{p}_1[\![P_1]\!] \parallel \cdots \parallel \mathsf{p}_n[\![P_n]\!] : \mathsf{G}} \ [\text{T-Net}]$$

Figure 4 Main typing rules for processes and networks.

5 Type System and Soundness

The shape of *typing judgements* is $\Gamma \vdash P : \mathsf{T}$, where the environment Γ associates process variables with session types: $\Gamma ::= \emptyset \mid \Gamma, X : \mathsf{T}$. Process typing exploits the correspondence between external choices and intersections, internal choices and unions. Fig.4 shows the interesting rules. Typing respects hats, in rules [T-ExtCh], [T-IntCh], [T-ExtCh-RT], [T-IntCh-RT] intersections and unions have hats if choices have hats. Subtyping $\leqslant$ on session types takes into account the standard rules for intersection and union and preserves hats, while checkpoint sets can decrease with deletion of types in intersections and increase with addition of types in unions. Subtyping is used in a standard subsumption rule (omitted). The other omitted rules for processes are just homomorphisms as expected from the syntax of processes and session types. Rule [T-Net] is the only rule for typing networks: it requires that the types of all processes be projections of a unique global type. The condition $\mathsf{pa}(\mathsf{G}) \subseteq \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ ensures the presence of all session participants and allows the typing of sessions containing $\mathsf{p}[\![\mathsf{skip}]\!]$ for any p, a property needed to guarantee invariance of types under structural equivalence of networks. Clearly, typing imposes constraints on the way hats and checkpoint labels are placed within processes.

▶ **Example 8.** Let $P = \mu X.\mathsf{q}!\lambda_1; \mathsf{r}!\lambda_2; X \ {}_{\{C\}}\!\oplus\ \mathsf{q}!\lambda_3; \mathsf{r}!\lambda_4; X$ and $Q = \mu Y.\mathsf{p}?\lambda_1; Y \ {}_{\{C\}}\!+\ \mathsf{p}?\lambda_3; Y$ and $R = \mu Z.\mathsf{p}?\lambda_2; Z \ {}_{\{C\}}\!+\ \mathsf{p}?\lambda_4; Z$. The network $\mathsf{p}[\![P]\!] \parallel \mathsf{q}[\![Q]\!] \parallel \mathsf{r}[\![R]\!]$ reduces by forward reductions to $\mathsf{p}[\![P']\!] \parallel \mathsf{q}[\![Q']\!] \parallel \mathsf{r}[\![R']\!]$ where
$P' = \widehat{\mathsf{q}!\lambda_1}; \widehat{\mathsf{r}!\lambda_2}; (\mathsf{q}!\lambda_1; \mathsf{r}!\lambda_2; P \ {}_{\{C\}}\!\oplus\ \widehat{\mathsf{q}!\lambda_3}; \mathsf{r}!\lambda_4; P) \ {}_{\{C\}}\!\oplus\ \mathsf{q}!\lambda_3; \mathsf{r}!\lambda_4; P$ and
$Q' = \widehat{\mathsf{p}?\lambda_1}; (\mathsf{p}?\lambda_1; Q \ {}_{\{C\}}\!+\ \widehat{\mathsf{p}?\lambda_3}; Q) {}_{\{C\}}\!+\ \mathsf{p}?\lambda_3; Q$ and $R' = \widehat{\mathsf{p}?\lambda_2}; R \ {}_{\{C\}}\!+\ \mathsf{p}?\lambda_4; R$. By Rule [Back], $\mathsf{p}[\![P']\!] \parallel \mathsf{q}[\![Q']\!] \parallel \mathsf{r}[\![R']\!] \overset{C}{\circlearrowleft} \mathsf{p}[\![P'']\!] \parallel \mathsf{q}[\![Q]\!] \parallel \mathsf{r}[\![R]\!]$ where $P'' = \mathsf{q}!\lambda_3; \mathsf{r}!\lambda_4; P$. Without the condition $\mathcal{E}\ \mathsf{ok}$ for $\overline{C}$ on rule [CtBA], we could have also the backward move: $\mathsf{p}[\![P']\!] \parallel \mathsf{q}[\![Q']\!] \parallel \mathsf{r}[\![R']\!] \overset{C}{\circlearrowleft} \mathsf{p}[\![P''']\!] \parallel \mathsf{q}[\![Q'']\!] \parallel \mathsf{r}[\![R]\!]$, where
$P''' = (\widehat{\mathsf{q}!\lambda_1}; \widehat{\mathsf{r}!\lambda_2}; \mathsf{q}!\lambda_1; \mathsf{r}!\lambda_2; P) \ {}_{\{C\}}\!\oplus\ \mathsf{q}!\lambda_3; \mathsf{r}!\lambda_4; P$ and $Q'' = \widehat{\mathsf{p}?\lambda_1}; Q \ {}_{\{C\}}\!+\ \mathsf{p}?\lambda_3; Q$. Then Subject Reduction would fail, since the network $\mathsf{p}[\![P]\!] \parallel \mathsf{q}[\![Q]\!] \parallel \mathsf{r}[\![R]\!]$ is typable, while $\mathsf{p}[\![P''']\!] \parallel \mathsf{q}[\![Q'']\!] \parallel \mathsf{r}[\![R]\!]$ is not typable. In fact the output $\mathsf{r}!\lambda_2$ has a hat in the session type of P''', while the corresponding input $\mathsf{p}?\lambda_2$ does not have a hat in the session type of R. This example shows also why it would not be possible to roll back to the last checkpoints in recursive processes, since they could be different for different participants.

It is easy to verify that the type $\mathsf{T}^{\mathbf{S}}$ given at page 10 can be derived for the process $P^{\mathbf{S}}$ given at the end of Sect.3.

The present type system is not informative enough: $\vdash \mathbb{N} : \mathsf{G}$ does not imply that a communication in G can be done in a forward computation of $\mathbb{N}$. For example, consider $\vdash \mathbb{N}_0 : \mathsf{G}_0$ where $\mathbb{N}_0 = \mathsf{p}[\![\, \mathsf{q}!\lambda \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda + \mathsf{p}?\lambda' \,]\!]$ and $\mathsf{G}_0 = \mathsf{p} \xrightarrow{\lambda} \mathsf{q} \boxplus \mathsf{p} \xrightarrow{\lambda'} \mathsf{q}$. The communication $\mathsf{p} \xrightarrow{\lambda'} \mathsf{q}$ cannot occur in $\mathbb{N}_0$. To discuss properties, we introduce a type system which represents more closely the evolution of networks.

We write $\Gamma \vdash^\star P : \mathsf{T}$ if this judgement can be derived using the typing rules of Fig.4 without the help of the subtyping rule. Let $\leqslant^\star$ be the subtyping relation between session types obtained from $\leqslant$ by requiring that unions have the same number of disjuncts.

We write $\vdash^\star \mathbb{N} : \mathsf{G}$ if this judgement can be derived using the typing rule:

$$\frac{\vdash^\star P_i : \mathsf{T}_i \quad \mathsf{T}_i \leqslant^\star \mathsf{G} \upharpoonright \mathsf{p}_i \quad (1 \leq i \leq n) \quad \mathsf{pa}(\mathsf{G}) \subseteq \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}}{\vdash \mathsf{p}_1[\![\, P_1 \,]\!] \parallel \cdots \parallel \mathsf{p}_n[\![\, P_n \,]\!] : \mathsf{G}} \; [\text{T-NET}^\star]$$

The relation between the type systems $\vdash^\star$ and $\vdash$ is expressed by Theorem 9, whose proof uses standard Inversion Lemmas. The property of Subject Reduction for $\vdash^\star$ is then established by Theorem 10. Its proof relies on the close correspondence between the evolution of well-typed nets and the evolution of their types along forward and backward computations.

▶ **Theorem 9.** *If* $\vdash^\star \mathbb{N} : \mathsf{G}$*, then* $\vdash \mathbb{N} : \mathsf{G}$*. If* $\vdash \mathbb{N} : \mathsf{G}$*, then* $\vdash^\star \mathbb{N} : \mathsf{G}'$ *for some* G'*.*

▶ **Theorem 10** (Subject Reduction)**.** $\vdash^\star \mathbb{N} : \mathsf{G}$ *and* $\mathbb{N} \overset{\curvearrowright}{\longrightarrow}^* \mathbb{N}'$ *imply* $\vdash^\star \mathbb{N}' : \mathsf{G}'$ *for some* G'*.*

Notice that Theorems 9 and 10 imply Subject Reduction for $\vdash$.

A standard property enforced by session types is Session Fidelity: all communications occur as specified by global types. To deal with backward reductions we introduce a mapping $\wr\!\!\wr \;\; \int\!\!\!\int$ on global types, which erases all communications with hats and all discarded branches of choices. A checkpoint label C is *alive* in G if $\wr\!\!\wr\, \mathsf{G} \,\int\!\!\!\int$ contains a choice ${}_{\{C\}}\boxplus_{i\in I}\ \mathsf{p} \xrightarrow{\lambda_i} \mathsf{q}_i; \mathsf{G}_i$ with more than one branch, such that for some $i \in I, \lambda, \mathsf{q}$, the global type G_i contains $\mathsf{p} \xrightarrow{\lambda} \mathsf{q}$.

▶ **Theorem 11** (Session Fidelity)**.** *If* $\vdash \mathbb{N} : \mathsf{G}$*, then all traces in forward computations of* $\mathbb{N}$ *are suffixes of traces of* G*. Moreover if* $\mathbb{N} \longrightarrow^* \mathbb{N}' \overset{C}{\curvearrowright} \mathbb{N}''$*, then* C *is alive in* G*.*

We discuss now forward and backward progress.

▶ **Theorem 12** (Forward Progress)**.** *If* $\vdash \mathsf{p}[\![\, \mathcal{E}[P] \,]\!] \parallel \mathbb{N} : \mathsf{G}$ *with* P *user process,* $P \neq \mathsf{skip}$*, then there are* P'*,* $\mathbb{N}'$ *such that* $\mathsf{p}[\![\, \mathcal{E}[P] \,]\!] \parallel \mathbb{N} \longrightarrow^* \mathsf{p}[\![\, \mathcal{E}[P'] \,]\!] \parallel \mathbb{N}'$ *and* P' *has one hat.*

Notice that the standard formulation of progress [12], which requires that each input/output that is persistently offered be eventually consumed, is an easy consequence of this theorem.

▶ **Theorem 13** (Backward Progress)**.** *If* $\vdash^\star \mathbb{N} : \mathsf{G}$ *and* C *is alive in* G*, then there is* $\mathbb{N}'$ *such that* $\mathbb{N} \longrightarrow^* \mathbb{N}'$ *and* $\mathbb{N}' \overset{C}{\curvearrowright} \mathbb{N}''$*.*

We end this section with a remark on causal consistency and full reversibility. Clearly, our calculus enjoys causal consistency since a rollback to a checkpointed choice cancels all communications done after that checkpoint. Instead, full reversibility does not hold, since rollbacks erase explored branches in internal choices.

6 Related Work and Conclusion

Since the seminal work by Danos and Krivine on reversible CCS [9], reversible computation has been widely studied in process calculi. In [29], Phillips and Ulidowski proposed a method

for reversing process operators defined in a general SOS format, and noted that thread identifiers and histories were needed to record the past of computations. In [21], Lanese et al. defined a reversible variant of the higher-order π-calculus, using tags to identify threads, and explicit memory processes. This calculus was enriched with a fine-grained rollback primitive in [20]. In [8], Cristescu et al. proposed a causal semantic model for the reversible π-calculus.

Reversibility for structured communications was first studied in [10, 11, 19], where *transactions* with rollback and coordinated checkpoints were modelled in an extended CCS. More recently, reversibility has been incorporated into *contracts* [1, 4] and *session calculi* [17, 18]. In [2] and [3], Barbanera et al. investigated the notions of compliance and sub-behaviour for contracts with checkpoints.While rollbacks are forgetful in [2], in [3] they are used as a strategy to achieve compliance: in this case, after a rollback a process cannot engage again in the previously explored branch, presumably unsuccessful.

Our work is most closely related to the recent proposals [33, 34, 23, 22, 24, 28, 26, 14]. Tiezzi and Yoshida [33] use tags and memories to allow full reversibility of binary sessions with delegation. In [34], two forms of reversibility are considered: either a session is completely reversed in a single backward step, or any intermediate state is restored. Mezzina and Pérez [23, 22] use monitors as memories for reversing binary sessions. A key novelty of this work is the use of session types with present and past. In [24], this approach is generalised to multiparty sessions, asynchronous higher-order communications, and decoupled rollbacks. In [28], Neykova and Yoshida provide an algorithm to analyse and extract causal dependencies from a given multiparty global type, and use it to ensure that communicating processes are safely recovered from consistent states in the presence of a failure. In [26], Mezzina and Tuosto propose a semantic control of reversibility: a computation along a branch is reversed according to the guards on the current configuration. A feature of [26] is that inputs are potentially irreversible actions, unless they appear within a loop. Finally, our paper builds on [14] and improves it in several respects: it has a more liberal syntax for types and processes, gives a more compact representation of past communications and implements a fine-tuned strategy for backward moves, geared towards achieving compliance.

It should be noted that in all the above-mentioned proposals, the syntax of global types (if any) does not include recursion, parallel composition nor sequential composition. Thus, some of our examples cannot be expressed in these models. On the other hand, the context-free session types of [32], recently proposed to capture the type-safe serialisation of recursive data types, include sequential composition and recursion. Our session types generalise them by offering in addition parallel composition, and by handling multiparty sessions.

The properties of our calculus are standard for session types, but their proofs require some ingenuity due to the generality of our syntax and to the specificity of our rollback mechanism. A limitation of our work is that the starting points of rollbacks are statically determined. By contrast, these points are determined dynamically in [26], offering a more realistic solution. We plan to introduce similar runtime conditions for rollback in our calculus.

Since our session types and global types are "truly concurrent", we would also like to study their interpretation into some non-interleaving semantic model. A natural candidate that comes to mind is the model of Event Structures, for which reversible variants have already been proposed [30, 16]. We plan to explore a reversible variant of *Flow Event Structures* [5], a model that has already been used to interpret CCS processes with past in [6].

Acknowledgements. We are grateful to the anonymous reviewers for their useful suggestions, which led to substantial improvements. We also thank Claudio Antares Mezzina for pointing out a technical problem in an earlier version of the paper.

References

1 Franco Barbanera and Ugo de' Liguoro. Sub-behaviour relations for session-based client/server systems. *Mathematical Structures in Computer Science*, 25(6):1339–1381, 2015. doi:10.1017/S096012951400005X.

2 Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Reversible client/server interactions. *Formal Aspects of Computing*, 28(4):697–722, 2016. doi:10.1007/s00165-016-0358-2.

3 Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese, and Ugo de' Liguoro. Retractable contracts. In *PLACES*, volume 203 of *EPTCS*, pages 61–72, 2016. doi:10.4204/EPTCS.203.

4 Giovanni Bernardi and Matthew Hennessy. Modelling session types using contracts. *Mathematical Structures in Computer Science*, 26(3):510–560, 2016. doi:10.1017/S0960129514000243.

5 Gérard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In *REX*, volume 354 of *LNCS*, pages 411–427. Springer, 1988. doi:10.1007/BFb0013028.

6 Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: three equivalent semantics for CCS. *Information and Computation*, 114(2):247–314, 1994. doi:10.1006/inco.1994.1088.

7 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On Global Types and Multi-Party Sessions. *Logical Methods in Computer Science*, 8:1–45, 2012. doi:10.2168/LMCS-8(1:24)2012.

8 Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for the reversible π-calculus. In *RC*, volume 9720 of *LNCS*, pages 3–19. Springer, 2016. doi:10.1007/978-3-319-40578-0_1.

9 Vincent Danos and Jean Krivine. Reversible communicating systems. In *CONCUR*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004. doi:10.1007/978-3-540-28644-8_19.

10 Edsko de Vries, Vasileios Koutavas, and Matthew Hennessy. Communicating transactions - (extended abstract). In *CONCUR*, volume 6269 of *LNCS*, pages 569–583. Springer, 2010. doi:10.1007/978-3-642-15375-4_39.

11 Edsko de Vries, Vasileios Koutavas, and Matthew Hennessy. Liveness of communicating transactions - (extended abstract). In *APLAS*, volume 6461 of *LNCS*, pages 392–407. Springer, 2010. doi:10.1007/978-3-642-17164-2_27.

12 Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446. ACM Press, 2011. doi:10.1145/1926385.1926435.

13 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012. doi:10.1007/978-3-642-28869-2_10.

14 Mariangiola Dezani-Ciancaglini and Paola Giannini. Reversible multiparty sessions with checkpoints. In *EXPRESS/SOS*, volume 222 of *EPTCS*, pages 60–74, 2016. doi:10.4204/EPTCS.222.5.

15 Elena Giachino, Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-consistent rollback in a tuple-based language. *Journal of Logic and Algebraic Methods in Programming*, 88:99–120, 2017. URL: https://doi.org/10.1016/j.jlamp.2016.09.003, doi:10.1016/j.jlamp.2016.09.003.

16 Eva Graversen, Iain Phillips, and Nobuko Yoshida. Towards a categorical representation of reversible event structures. In *PLACES*, volume 246 of *EPTCS*, pages 49–60, 2017. doi:10.4204/EPTCS.246.9.

17 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998. doi:10.1007/BFb0053567.

18 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM Press, 2008. doi:10.1145/1328897.1328472.

19 Vasileios Koutavas, Carlo Spaccasassi, and Matthew Hennessy. Bisimulations for communicating transactions - (extended abstract). In *FOSSACS*, volume 8412 of *LNCS*, pages 320–334. Springer, 2014. doi:10.1007/978-3-642-54830-7_21.

20 Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt, and Jean-Bernard Stefani. Controlling reversibility in higher-order pi. In *CONCUR*, volume 6901 of *LNCS*, pages 297–311. Springer, 2011. doi:10.1007/978-3-642-23217-6_20.

21 Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversing higher-order pi. In *CONCUR*, volume 6269 of *LNCS*, pages 478–493. Springer, 2010. doi:10.1007/978-3-642-15375-4_33.

22 Claudio Antares Mezzina and Jorge A. Pérez. Reversible semantics in session-based concurrency. In *ICTCS*, volume 1720 of *CEUR Workshop Proceedings*, pages 221–226. CEUR-WS.org, 2016.

23 Claudio Antares Mezzina and Jorge A. Pérez. Reversible sessions using monitors. In *PLACES*, volume 211 of *EPTCS*, pages 56–64, 2016. doi:10.4204/EPTCS.211.6.

24 Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies. *CoRR*, abs/1703.06021, 2017.

25 Claudio Antares Mezzina, Rudolf Schlatte, and al. COST Action on Reversible Computation, State of the Art Report, Working Group 2 on Software and Systems, 2012. URL: http://www.informatik.uni-bremen.de/ictcost/wg2_soar.pdf.

26 Claudio Antares Mezzina and Emilio Tuosto. Choreographies for automatic recovery. *CoRR*, abs/1705.09525, 2017.

27 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. doi:10.1007/3-540-10235-3.

28 Rumyana Neykova and Nobuko Yoshida. Let it recover: multiparty protocol-induced recovery. In *CC*, pages 98–108. ACM Press, 2017. doi:10.1145/3033019.

29 Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Methods in Programming*, 73(1-2):70–96, 2007. doi:10.1016/j.jlap.2006.11.002.

30 Iain Phillips and Irek Ulidowski. Reversibility and asymmetric conflict in event structures. *Journal of Logic and Algebraic Methods in Programming*, 84(6):781–805, 2015. doi:10.1016/j.jlamp.2015.07.004.

31 Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

32 Peter Thiemann and Vasco T. Vasconcelos. Context-free session types. In *ICFP*, pages 462–475. ACM Press, 2016. doi:10.1145/2951913.2951926.

33 Francesco Tiezzi and Nobuko Yoshida. Reversible session-based pi-calculus. *Journal of Logical and Algebraic Methods in Programming*, 84(5):684–707, 2015. doi:10.1016/j.jlamp.2015.03.004.

34 Francesco Tiezzi and Nobuko Yoshida. Reversing single sessions. In *RC*, volume 9720 of *LNCS*, pages 52–69. Springer, 2016. doi:10.1007/978-3-319-40578-0_4.