Reversing Single Sessions*

Francesco Tiezzi and Nobuko Yoshida

[1] University of Camerino, Italy francesco.tiezzi@unicam.it
[2] Imperial College London, UK n.yoshida@imperial.ac.uk

Abstract. Session-based communication has gained a widespread acceptance in practice as a means for developing safe communicating systems via structured interactions. In this paper, we investigate how these structured interactions are affected by reversibility, which provides a computational model allowing executed interactions to be undone. In particular, we provide a systematic study of the integration of different notions of reversibility in both binary and multiparty single sessions. The considered forms of reversibility are: one for completely reversing a given session with one backward step, and another for also restoring any intermediate state of the session with either one backward step or multiple ones. We analyse the costs of reversing a session in all these different settings. Our results show that extending binary single sessions to multiparty ones does not affect the reversibility machinery and its costs.

1 Introduction

In modern ICT systems, the role of communication is more and more crucial. This calls for a communication-centric programming style supporting safe and consistent composition of protocols. In this regard, in the last decade, primitives and related type theories supporting structured interactions, namely *sessions*, among system participants have been extensively studied (see, e.g., [10,22,5,18,11]).

Another key aspect of ICT systems concerns their reliability. Recently, *reversibility* has been put forward as a convenient support for programming reliable systems. In fact, it allows a system that has reached an undesired state to undo, in automatic fashion, previously performed actions. Again, foundational studies of mechanisms for reversing action executions have been carried out (see, e.g., [7,8,9,19,14,13,6,4]).

In this paper, we investigate how the benefits of reversibility can be brought to structured communication and, hence, how reversibility and sessions affect each other. We concentrate on the primitives and mechanisms required to incorporate different notions of reversibility into two forms of session, and we analyse the costs of reversing a session in these different settings. To study the interplay between reversibility and sessions we rely on a uniform foundational framework, based on π-calculus [17].

Specifically, we focus on a simplified form of session, called *single*, in which the parties that have created a session can only continue to interact along that single session. This setting permits to consider a simpler theoretical framework than the one with

* This research has been partially founded by EPSRC EP/K011715/1, EP/K034413/1 and EP/L00058X/1, EU FP7 FETOpenX Upscale, MIUR PRIN Project CINA (2010LHT4KM), and the COST Actions BETTY (IC1201) and Reversible computation (IC1405).

```
Binary Single Sessions                              Multiparty Single Sessions
           ├── (1) Whole session reversibility                ├── (4) Whole session reversibility
           └── Session interactions reversibility             └── Session interactions reversibility
                               ├── (2) Multi-step                                 ├── (5) Multi-step
                               └── (3) Single-step                                └── (6) Single-step
```

Fig. 1. Sessions and Reversibility: considered combinations

the usual notion of session, called here *multiple*, where parties can interleave interactions performed along different sessions. This allows us to focus on the basic, key aspects of our investigation. Although single sessions are simpler, they are still largely used in practice, and differently from multiple sessions (see, e.g., [20,2]) their effect to reversibility is not studied yet in the literature.

Concerning the parties involved in the sessions, we take into account both *binary* and *multiparty* sessions, which involve two or multiple parties, respectively. For each kind of session, we investigate the use of two forms of reversibility: *(i) whole session* reversibility, where a single backward step reverses completely the given session, thus directly restoring its initialisation state; and *(ii) session interactions* reversibility, where any intermediate state of the session can be restored, either in a *(ii.a) multi-step* or a *(ii.b) single-step* fashion. Figure 1 sums up the different combinations of sessions and reversibility we consider.

We exemplify the reversible approaches throughout the paper by resorting to a typical business protocol example, drawn from [11]. In case of binary session (Figure 2.(a)), the protocol involves a Buyer willing to buy a book from a Seller. Buyer sends the book title to Seller, which replies with a quote. If Buyer is satisfied by the quote, then he sends his address and Seller sends back the delivery date; otherwise Buyer quits the conversation. In the multiparty case (Figure 2.(b)), the above protocol is refined by considering two buyers, Buyer1 and Buyer2, that wish to buy an expensive book from Seller by combining their money. Buyer1 sends the title of the book to Seller, which sends the quote to both Buyer1 and Buyer2. Then, Buyer1 tells Buyer2 how much he is willing to contribute. Buyer2 evaluates how much he has to pay and either accepts, and exchanges the shipping information, or terminates the session. In these scenarios, reversibility can be entered into the game to deal with errors that may occur during the interactions, or to make the protocols more flexible by enabling negotiation via re-iteration of some interactions. For example, a buyer, rather than only accepting or rejecting a quote, can ask the seller for a new quote by simply reverting the interaction where the current quote has been communicated. Similarly, Buyer2 can negotiate the division of the quote with Buyer1. Other possibilities allow the buyers to partially undo the current session, in

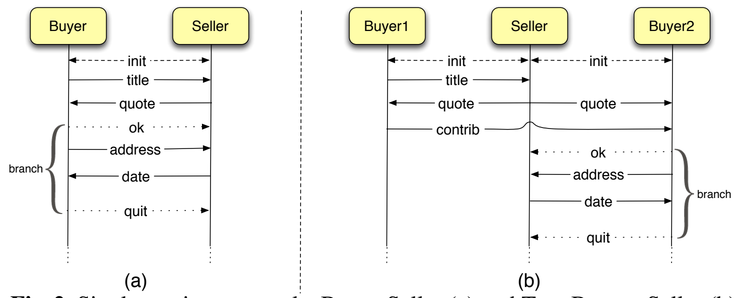

Fig. 2. Single session protocols: Buyer-Seller (a) and Two-Buyers-Seller (b)

order to take a different branch along the same session, or even start a new session with (possibly) another seller.

The contribution of this paper is twofold. Firstly, we show for each kind of session discussed above a suitable machinery that permits extending the corresponding non-reversible calculus in order to become reversible. Secondly, we compare the different cases, i.e. (1)-(6) in Figure 1, by means of their costs for reverting a session, given in terms of number of backward steps and occupancy of the data structures used to store the computation history (which is a necessary information to reverse the effects of session interactions). Our results about reversibility costs are summarised in Figure 3. It is worth noticing that linearity of sessions permits to achieve costs that are at most linear. Moreover, despite in case of complex interactions the multiparty approach provides a programming style more natural than the binary one, binary and multiparty sessions have the same reversibility costs. We discuss at the end of the paper how our work can extend to multiple sessions, which require a much heavier machinery for reversibility with respect to single ones, and have higher costs. This means that it is not convenient to use in the single session setting the same reversible machineries already developed for calculi with multiple sessions, which further motivates our investigation.

	Binary			Multiparty		
	(1)	(2)	(3)	(4)	(5)	(6)
# backward steps	1	n	1	1	n	1
# memory items	1	n	n	1	n	n

n: number of interactions along the session to be reversed

Fig. 3. Costs of Reversing Single Sessions

The practical benefit of our systematic study is that it supplies a support to system designers for a conscious selection of the combination of session notion and reversibility mechanism that is best suited to their specific needs.

Summary of the rest of the paper. Section 2 provides background notions on binary and multiparty session-based variantes of π-calculus. Section 3 shows how reversibility can be incorporated in single binary sessions and what is its cost, while Section 4 focusses on multiparty ones. Section 5 concludes by reviewing strictly related work and by touching upon directions for future work. We refer to the companion technical report [21] for further background material and proofs of results.

2 Background on session-based π-calculi

In this section, we give the basic definitions concerning two variants of the π-calculus, enriched with primitives for managing binary and multiparty sessions, respectively.

Binary session calculus. The syntax definition of the binary session π-calculus [22] relies on the following base sets: *variables* (ranged over by x), storing values; *shared channels* (ranged over by a), used to initiate sessions; *session channels* (ranged over by s), consisting on pairs of *endpoints* (ranged over by s, $\bar{s}$) used by the two parties to exchange values within an established session; *labels* (ranged over by l), used to select and offer branching choices; and *process variables* (ranged over by X), used for recursion. Letter u denotes *shared identifiers*, i.e. shared channels and variables together; letter k denotes *session identifiers*, i.e. session endpoints and variables together; letter c denotes *channels*, i.e. session channels and shared channels together. *Values*, including booleans, integers, shared channels and session endpoints, are ranged over by v.

$P ::=$	**Processes**
$\bar{u}(x).P \mid u(x).P \mid k!\langle e\rangle.P \mid k?(x).P$	request, accept ,output, input
$\mid k \triangleleft l.P \mid k \triangleright \{l_1 : P_1, \ldots, l_n : P_n\} \mid \mathbf{0} \mid P_1 \,\vert\, P_2$	selection, branching, inact, parallel
$\mid (\nu c)P \mid \texttt{if}\, e \,\texttt{then}\, P_1 \,\texttt{else}\, P_2 \mid X \mid \mu X.P$	choice, restriction, recursion
$e ::= v \mid \mathrm{op}(e_1, \ldots, e_n)$	**Expressions**

Fig. 4. Binary session calculus: syntax

$\bar{a}(x_1).P_1 \mid a(x_2).P_2 \to (\nu s)(P_1[\bar{s}/x_1] \mid P_2[s/x_2])$	$s, \bar{s} \notin \mathrm{fse}(P_1, P_2)$	[CON]
$\bar{k}!\langle e\rangle.P_1 \mid k?(x).P_2 \to P_1 \mid P_2[v/x]$	$(k = s \text{ or } k = \bar{s}),\ e \downarrow v$	[COM]
$\bar{k} \triangleleft l_i.P \mid k \triangleright \{l_1 : P_1, \ldots, l_n : P_n\} \to P \mid P_i$	$(k = s \text{ or } k = \bar{s}),\ 1 \leq i \leq n$	[LAB]
$\texttt{if}\, e \,\texttt{then}\, P_1 \,\texttt{else}\, P_2 \to P_1$	$e \downarrow \texttt{true}$	[IF1]
$\texttt{if}\, e \,\texttt{then}\, P_1 \,\texttt{else}\, P_2 \to P_2$	$e \downarrow \texttt{false}$	[IF2]

$$\frac{P_1 \to P_1'}{P_1 \,\vert\, P_2 \to P_1' \,\vert\, P_2}\ [\textsc{Par}] \qquad \frac{P \to P'}{(\nu c)P \to (\nu c)P'}\ [\textsc{Res}] \qquad \frac{P_1 \equiv P_1' \to P_2' \equiv P_2}{P_1 \to P_2}\ [\textsc{Str}]$$

Fig. 5. Binary session calculus: reduction relation

Processes (ranged over by P) and *expressions* (ranged over by e and defined by means of a generic expression operator op representing standard operators on boolean and integer values) are given by the grammar in Figure 4.

The operational semantics of the calculus is given in terms of a structural congruence and of a reduction relation, and is only defined for *closed* terms, i.e. terms without free variables. The *structural congruence*, written $\equiv$, is standard (see [21]). The *reduction relation*, written $\to$, is the smallest relation on closed processes generated by the rules in Figure 5. We resort to the auxiliary function $\cdot \downarrow$ for evaluating closed expressions: $e \downarrow v$ says that expression e evaluates to value v. Notationally, for P a process, $\mathrm{fv}(P)$ denotes the set of free variables in P, and $\mathrm{fse}(P)$ the set of free session endpoints. We comment on salient points. A new session is established when two parallel processes synchronise via a shared channel a; this results on the generation of a fresh (private) session channel whose endpoints are assigned to the two session parties (rule [CON]). Along a session, the two parties can exchange values (for data- and channel-passing, rule [COM]) and labels (for branching selection, rule [LAB]). The other rules are standard and state that: conditional choice evolves according to the evaluation of the expression argument (rules [IF1] and [IF2]); if a part of a larger process evolves, the whole process evolves accordingly (rules [PAR] and [RES]); and structural congruent processes have the same reductions (rule [STR]).

The syntax of *sorts* (ranged over by S) and *types* (ranged over by α, β) used in the *binary session type discipline* is defined in Figure 6. The type $![S].\alpha$ represents the behaviour of first outputting a value of sort S, then performing the actions prescribed by type α; type $![\beta].\alpha$ represents a similar behaviour, which starts with session output (*delegation*) instead; types $?[S].\alpha$ and $?[\beta].\alpha$ are the dual ones, receiving values instead of sending. Type $\oplus[l_1 : \alpha_1, \ldots, l_n : \alpha_n]$ represents the behaviour which would select one of l_i and then behaves as α_i, according to the selected l_i (internal choice). Type $\&[l_1 : \alpha_1, \ldots, l_n : \alpha_n]$ describes a branching behaviour: it waits with n options, and behave as type α_i if the i-th action is selected (external choice). Type end represents

$$\begin{array}{ll}
S ::= & \textbf{Sorts} \\
\quad \mathsf{bool} \quad | \quad \mathsf{int} \quad | \quad \langle \alpha \rangle & \text{boolean, integer, shared channel} \\
 & \\
\alpha ::= & \textbf{Types} \\
\quad ![S].\alpha \quad | \quad ![\beta].\alpha \quad | \quad ?[S].\alpha \quad | \quad ?[\beta].\alpha & \text{output, input} \\
\quad | \ \oplus[l_1 : \alpha_1, \ldots, l_n : \alpha_n] \ | \ \&[l_1 : \alpha_1, \ldots, l_n : \alpha_n] & \text{selection, branching} \\
\quad | \ \mathsf{end} \quad | \quad t \quad | \quad \mu t.\alpha & \text{end, recursion}
\end{array}$$

Fig. 6. Binary session calculus: sorts and types

$$\begin{array}{ll}
P ::= & \textbf{Processes} \\
\quad \bar{u}[\mathtt{p}](x).P \ | \ u[\mathtt{p}](x).P \ | \ k[\mathtt{p}]!\langle e\rangle.P \ | \ k[\mathtt{p}]?(x).P & \text{request, accept, output, input} \\
\quad | \ k[\mathtt{p}] \triangleleft l.P \ | \ k[\mathtt{p}] \triangleright \{l_1 : P_1, \ldots, l_n : P_n\} \ | \ \mathbf{0} \ | \ P_1 \,|\, P_2 & \text{selection, branching, inact, par.} \\
\quad | \ \mathtt{if}\, e \,\mathtt{then}\, P \,\mathtt{else}\, Q \ | \ (\nu c)\, P \ | \ X \ | \ \mu X.P & \text{choice, restriction, recursion}
\end{array}$$

Fig. 7. Multiparty session calculus: syntax

inaction, acting as the unit of sequential composition. Type $\mu t.\alpha$ denotes a recursive behaviour, representing the behaviour that starts by doing α and, when variable t is encountered, recurs to α again.

Typing judgements are of the form $\Theta; \Gamma \vdash P \triangleright \Delta$, where Θ, Γ and Δ, called *basis*, *sorting* and *typing* respectively, are finite partial maps from shared identifiers to sorts, from session identifiers to types, and from process variables to typings, respectively Intuitively, the judgement $\Theta; \Gamma \vdash P \triangleright \Delta$ stands for "under the environment $\Theta; \Gamma$, process P has typing Δ". The axioms and rules defining the typing system are standard (see [21]).

Example 1 (Buyer-Seller protocol). We show how the protocol in Figure 2.(a) is rendered in the variant of π-calculus with binary sessions. The behaviour of Buyer is described by the following process:

$$\begin{array}{l}
\mathit{Buyer} \triangleq \bar{a}(x).\, x!\langle\text{“}\mathit{The\ Divine\ Comedy}\text{”}\rangle.\, x?(x_{quote}). \\
\qquad\qquad \mathtt{if}\ x_{quote} \leq 20\ \mathtt{then}\ x \triangleleft l_{ok}.\, x!\langle \mathit{addr}()\rangle.\, x?(x_{date}).\, P\ \mathtt{else}\ x \triangleleft l_{quit}.\, \mathbf{0}
\end{array}$$

This Buyer is interested in buying the Divine Comedy and is willing to pay not more than 20 euros. The Seller participant instead is rendered as follows:

$$\mathit{Seller} \triangleq a(z).\, z?(z_{title}).\, z!\langle \mathit{quote}(z_{title})\rangle.\, z \triangleright \{l_{ok} : z?(z_{addr}).\, z!\langle \mathit{date}()\rangle.Q\ , l_{quit} : \mathbf{0}\}$$

Note that $\mathit{addr}()$, $\mathit{quote}()$ and $\mathit{date}()$ are used to get a buyer address, a quote for a given book, and the delivery date, respectively. The overall specification is $\mathit{Buyer} \mid \mathit{Seller}$.

Multiparty session calculus. The base sets for the synchronous multiparty session calculus [12] are the same of the binary case, except for *session endpoints*, which now are denoted by $s[\mathtt{p}]$, with $\mathtt{p},\mathtt{q}$ ranging over *roles* (represented as natural numbers). Thus, *session identifiers* k now range over session endpoints $s[\mathtt{p}]$ or variables x.

The syntax of the calculus is defined by the grammar in Figure 7, where expressions e are defined as in the binary case (with values that extends to multiparty session endpoints). Primitive $\bar{u}[\mathtt{p}](x).P$ initiates a new session through identifier u on the other multiple participants, each one of the form $u[\mathtt{q}](x).P_{\mathtt{q}}$ where $1 \leq \mathtt{q} \leq \mathtt{p} - 1$. Variable x will be substituted with the session endpoint used for the interactions inside the established session. Primitive $k[\mathtt{p}]!\langle e\rangle.P$ denotes the intention of sending a value to role

$$\bar{a}[n](x).P_n \mid \prod_{i=\{1,..,n-1\}} a[i](x).P_i \;\to\; (\nu s)(P_n[s[n]/x] \mid \prod_{i=\{1,..,n-1\}} P_i[s[i]/x]) \qquad s \notin \text{fse}(P_i) \text{ with } i = \{1,..,n\} \qquad \text{[M-CON]}$$

$$s[\mathtt{p}][\mathtt{q}]!\langle e\rangle.P \mid s[\mathtt{q}][\mathtt{p}]?(x).Q \;\to\; P \mid Q[v/x] \qquad e \downarrow v \qquad \text{[M-COM]}$$

$$s[\mathtt{p}][\mathtt{q}] \triangleleft l_i.P \mid s[\mathtt{q}][\mathtt{p}] \triangleright \{l_1 : P_1, \ldots, l_n : P_n\} \;\to\; P \mid P_i \qquad 1 \leq i \leq n \qquad \text{[M-LAB]}$$

Fig. 8. Multiparty session calculus: reduction relation (excerpt of rules)

p; similarly, process $k[\mathtt{p}]?(x).P$ denotes the intention of receiving a value from role p. Selection and branching behave in a similar way.

As usual the operational semantics is given in terms of a structural congruence and of a reduction relation. The rules defining the structural congruence are the same ones used for the binary calculus, where the rule for the scope extension of session channels takes into account the new form of session endpoints. The *reduction relation* $\to$, instead, is the smallest relation on closed processes generated by the rules [IF1], [IF2], [PAR], [RES] and [STR] in Figure 5, and the additional rules in Figure 8. We comment on salient points. Rule [M-CON] synchronously initiates a session by requiring all session endpoints be present for a synchronous reduction, where each role p creates a session endpoint $s[\mathtt{p}]$ on a fresh session channel s. The participant with the maximum role ($\bar{a}[n](x).P_n$) is responsible for requesting a session initiation. Rule [M-COM] defines how a party with role p sends a value to the receiving party with role q. Selection and branching are defined in a similar way (rule [M-LAB]).

The type discipline of this synchronous multiparty session calculus is simpler than the asynchronous one in [11], but it is much more elaborate than the binary case, as it considers global and local types. Therefore, due to lack of space, we relegate the definitions of types, as well as the rules of the corresponding type system, to the companion technical report [21] and refer the interested reader to [12] for a detailed account.

Example 2 (Two-Buyers-Seller protocol). We show how the protocol in Figure 2.(b) is rendered in the variant of π-calculus with the multiparty sessions. The behaviour of Buyer1 and Buyer2 are described by the following processes:

$$\mathit{Buyer1} \triangleq \bar{a}[3](x).\, x[1]!\langle\text{“}\mathit{The\ Divine\ Comedy}\text{”}\rangle.\, x[1]?(x_{quote}).\, x[2]!\langle \mathit{split}(x_{quote})\rangle.\, P_1$$

$$\begin{aligned}\mathit{Buyer2} \triangleq\ & a[2](y).\, y[1]?(y_{quote}).\, y[3]?(y_{contrib}).\, \mathtt{if}\ y_{quote} - y_{contrib} \leq 10 \\ & \mathtt{then}\ y[1] \triangleleft l_{ok}.\, y[1]!\langle \mathit{addr}()\rangle.\, y[1]?(y_{date}).\, P_2\ \mathtt{else}\ y[1] \triangleleft l_{quit}.\, \mathbf{0}\end{aligned}$$

Now, Buyer1 divides the quote by means of the $\mathit{split}()$ function. The Seller process is similar to the binary case, but for the form of session endpoints:

$$\begin{aligned}\mathit{Seller} \triangleq\ & a[1](z).\, z[3]?(z_{title}).\, z[2]!\langle \mathit{quote}(z_{title})\rangle.\, z[3]!\langle \mathit{lastQuote}(z_{title})\rangle. \\ & z[2] \triangleright \{l_{ok} : z[2]?(z_{addr}).\, z[2]!\langle \mathit{date}()\rangle.P_3\, ,\ l_{quit} : \mathbf{0}\}\end{aligned}$$

where $\mathit{lastQuote}()$ simply returns the last quote computed for a given book.

3 Reversibility of single binary sessions

This section formally introduces the notion of single session, and illustrates constructs, mechanisms and costs to support the reversibility in the binary cases.

P ::=			**Reversible processes**
	$\ldots$ |	$\langle s : m \rangle \blacktriangleright P$	π-calculus processes, single session
m ::=			**Memory stacks**
	P |	$P \cdot m$	bottom element, push

Fig. 9. Reversible extension

3.1 Single sessions

In the single sessions setting, when two processes start a session their continuations only interact along this single session. Thus, neither delegation (i.e., passing of session endpoints) nor initialisation of new sessions (also after the session closure) is allowed. As clarified below, the exclusive use of single sessions is imposed to processes by means of a specific type system, thus avoiding the use of syntactical constraints.

Reversibility is incorporated in a process calculus typically by adding memory devices to store information about the computation history, which otherwise would be lost during computations. In all single session cases, i.e. (1)-(6) in Figure 1, we will extend the syntax of (binary/multiparty) session-based π-calculus as shown in Figure 9. The term $\langle s : m \rangle \blacktriangleright P$ represents a *single session* along the channel s with associated memory m and code P. A *memory* m is a (non-empty) stack of processes, each one corresponding to a state of the session (the bottom element corresponds to the term that initialised the session). The term $\langle s : m \rangle \blacktriangleright P$ is a binder, i.e. it binds session channel s in P. In this respect, it acts similarly to operator $(\nu s)P$, but the scope of $\langle s : m \rangle \blacktriangleright P$ cannot be extended.

In the obtained reversible calculi, terms can perform, besides standard *forward computations*, also *backward computations* that undo the effect of the former ones.

We compare approaches (1)-(6) with respect to the *cost* of reverting a session in the worst case, i.e. the cost of completely reverting a session. The cost is given in terms of *(i)* the *number of backward reductions* ($\mathcal{C}_{br}$), necessary to complete the rollback, and *(ii) memory occupancy* ($\mathcal{C}_{mo}$), i.e. the number of element in the memory stack of the session when the rollback starts. The two kinds of cost depend on the *length* of the considered session, given by the number of (forward) steps performed along the session.

3.2 Binary session reversibility

Not all processes allowed by the syntax presented above corresponds to meaningful processes in the reversible single sessions setting. Indeed, on the one hand, the syntax allows terms violating the single sessions limitation. On the other hand, in a general term of the calculus the history stored in its memories may not be consistent with the computation that has taken place.

We address the above issues by only considering a class of well-formed processes, called *reachable* processes. In the definition of this class of processes, to ensure the use of single sessions only, as in [11] we resort to the notion of *simple* process.

Definition 1 (Simple process). *A process is* simple *if (i) it is generated by the grammar in Figure 4 and (ii) it is typable with a type derivation using prefix rules where the session typings in the premise and the conclusion are restricted to at most a singleton*[3].

[3] Using the standard typing system for the binary session π-calculus (see [21, Figure 13]), point (ii) boils down to: Δ of rules [Req], [Acc], [Send], [Rcv], [Sel] and [Br] are empty; neither [Thr] nor [Cat] is used; $\Delta \cdot \Delta'$ in [Conc] contains at most a singleton; and Δ of the remaining rules contain at most a singleton.

The point (i) of the above definition states that a simple process has no memory, while point (ii) states that each prefixed subterm in a simple process uses only a single session.

The following properties clarify the notion of simple process.

Property 1. In a simple process, delegation is disallowed.

Proof. The proof proceeds by contradiction and straightforwardly follows from Definition 1 (see [21]).

Property 2. In a simple process, subordinate sessions (i.e., new sessions initialised within the single session) are disallowed.

Proof. The proof proceeds by contradiction (see [21]).

We explain the meaning of subordinate session used in Property 2 by means of an example. Let us consider the process $a(x).b(y).x!\langle 1\rangle.P$ with $y \in \mathrm{fv}(P)$, which initialises a (subordinate) session using channel b within a session previously initialised using channel a. This process is not simple, because typing it requires to type the (sub)process $b(y).x!\langle 1\rangle.P$ under typing $x :![\mathsf{int}].\alpha$, which is not empty as required by Definition 1.

Now, to ensure history consistency, as in [6] we only consider *reachable* processes, i.e. processes obtained by means of forward reductions from simple processes.

Definition 2 (Reachable processes). *The set of* reachable *processes, for the case (i) in Figure 1 with* $i \in \{1, 2, 3\}$*, is the closure under relation* $\twoheadrightarrow_{(i)}$ *(see below) of the set of simple processes.*

We clarify this notion by means of an example. The process stored in the memory of term $\langle s : (\bar{a}(x).x!\langle 1\rangle \mid a(y).y?(z))\rangle \blacktriangleright (\bar{s}!\langle 2\rangle \mid s?(z))$ is not consistent with the related session process, because there is no way to generate the term $(\bar{s}!\langle 2\rangle \mid s?(z))$ from the stored process (in fact, in the session process, the value sent along the endpoint $\bar{s}$ should be 1 instead of 2).

We now present the semantics of the reversibility machinery in the three binary cases. As usual, the operational semantics is given in terms of a structural congruence and a reduction relation[4]. For all three cases, the laws defining the structural congruence are the same of the binary session calculus. Instead, the *reduction* relations for the cases (1)-(3), written $\rightarrowtail_{(i)}$ with i $\in \{1, 2, 3\}$, are given as the union of the corresponding *forward reduction* relations $\twoheadrightarrow_{(i)}$ and *backward reduction* relations $\rightsquigarrow_{(i)}$, which are defined by different sets of rules in the three cases. Notably, the rules in Figure 10, whose meaning is straightforward, are shared between the three cases.

The semantics is only defined for closed, reachable terms, where now the definition of *closed* term extends to session endpoints, in the sense that all occurrences of session endpoints s and $\bar{s}$ have to be bound by a single session term $\langle s : m\rangle \blacktriangleright \cdot$. This latter requirement is needed for ensuring that every running session in the considered process

[4] We use a reduction semantics with respect to a labelled one because the former is simpler (e.g., it does not require to deal with scope extension of names) and, hence, is preferable when the labelled semantics is not needed (e.g., here we are not interested in labelled bisimulations). Moreover, works about session-based π-calculus use a reduction semantics, as well as many reversible calculi (e.g., [14], [4], [16]).

$$\texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \twoheadrightarrow_{(i)} P_1 \qquad e \downarrow \texttt{true} \qquad [\text{FW-IF1}]$$

$$\texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \twoheadrightarrow_{(i)} P_2 \qquad e \downarrow \texttt{false} \qquad [\text{FW-IF2}]$$

$$\frac{P_1 \twoheadrightarrow_{(i)} P_1'}{P_1 \mid P_2 \twoheadrightarrow_{(i)} P_1' \mid P_2}\ [\text{FW-PAR}] \qquad \frac{P \twoheadrightarrow_{(i)} P'}{(\nu c)P \twoheadrightarrow_{(i)} (\nu c)P'}\ [\text{FW-RES}]$$

$$\frac{P \equiv P' \twoheadrightarrow_{(i)} Q' \equiv Q}{P \twoheadrightarrow_{(i)} Q}\ [\text{FW-STR}]$$

Fig. 10. Single sessions: shared forward and backward rules (for $i \in \{1..6\}$); rules [BW-PAR], [BW-RES], [BW-STR] are omitted (they are like the forward rules where $\rightsquigarrow_{(i)}$ replaces $\twoheadrightarrow_{(i)}$)

can be reverted; for example, in the reachable process $(\bar{s}!\langle 1 \rangle \mid s?(x) \mid Q)$ there is a running session s that cannot be reverted because no computation history information (i.e., no memory stack) is available for it. We discuss below the additional definitions for the operational semantics of the three binary cases.

(1) Whole session reversibility. In this case the reversibility machinery of the calculus permits to undo only whole sessions. The forward rules additional to those in Figure 10 are as follows:

$$P \twoheadrightarrow_{(1)} \langle s : P \rangle \blacktriangleright (P_1[\bar{s}/x] \mid P_2[s/y]) \qquad P = (\bar{a}(x).P_1 \mid a(y).P_2) \quad [\text{FW(1)-CON}]$$

$$\frac{P \rightarrow P'}{\langle s : m \rangle \blacktriangleright P \twoheadrightarrow_{(1)} \langle s : m \rangle \blacktriangleright P'}\ [\text{FW(1)-MEM}]$$

Rule [FW(1)-CON] initiates a single session s with the initialisation term P stored in the memory stack. As usual the two session endpoints $\bar{s}$ and s replace the corresponding variables x and y in the two continuations P_1 and P_2 (within the scope of the single session construct). Notably, there is no need of using the restriction operator, because in the single session setting the session endpoints cannot be communicated outside the session, i.e., delegation is disallowed (Property 1). Moreover, differently from the non-reversible case (see rule [CON] in Figure 5), there is also no need of requiring the session endpoints s and $\bar{s}$ to be fresh in the session code (i.e., in P_1 and P_2), because of the notion of closed process given in this section. Rule [FW(1)-MEM] simply states that a process within the scope of a single session evolves with a forward reduction according to its evolution with a standard reduction (defined in Figure 5).

The only additional backward rule is the following one:

$$\langle s : P \rangle \blacktriangleright Q \rightsquigarrow_{(1)} P \qquad [\text{BW(1)}]$$

This rule permits to rollback the whole session conversation in every moment during its execution. In particular, the term that initialised the session is restored with a single backward reduction step. Notably, the fact that scope extension is not allowed for the operator $\langle \cdot : \cdot \rangle \blacktriangleright \cdot$ ensures that the process Q in [BW(1)] does not contain processes not belonging to session s, i.e. unwanted deletions are prevented. It is also worth noticing that the rollback of session s does not involve other sessions, as no subordinate sessions can be active in Q (Property 2).

(2) Multi-step. In this case a session can be reversed either partially or totally. When the rollback starts, it proceeds step-by-step and can terminate in any intermediate state of the session, as well as in the initialisation state. The additional forward rules are:

$$P \twoheadrightarrow_{(2)} \langle s : P \rangle \blacktriangleright (P_1[\bar{s}/x] \mid P_2[s/y]) \qquad P = (\bar{a}(x).P_1 \mid a(y).P_2) \quad [\textsc{Fw}(2)\text{-}\textsc{Con}]$$

$$\frac{P \rightarrow P'}{\langle s : m \rangle \blacktriangleright P \twoheadrightarrow_{(2)} \langle s : P \cdot m \rangle \blacktriangleright P'} \quad [\textsc{Fw}(2)\text{-}\textsc{Mem}]$$

Differently from the case (1), here it is necessary to keep track in the memory stack of each (forward) interaction that has taken place in the session. Therefore, the forward rule [FW(2)-MEM] pushes the process P, representing the state before the transition, into the stack.

The backward reduction relation is defined by the following additional rules:

$$\langle s : P \rangle \blacktriangleright Q \rightsquigarrow_{(2)} P \quad [\textsc{Bw}(2)\text{-}1] \qquad \langle s : P \cdot m \rangle \blacktriangleright Q \rightsquigarrow_{(2)} \langle s : m \rangle \blacktriangleright P \quad [\textsc{Bw}(2)\text{-}2]$$

Rule [BW(2)-1] is like to [BW(1)], but here it can be used only when the memory stack contains just one element. The single session is removed because its initialisation state is restored. Rule [BW(2)-2], instead, permits undoing an intermediate state Q, by simply replacing it with the previous intermediate state P. In this case, since after the reduction the stack is not empty, the single session construct is not removed.

(3) Single-step. This is similar to the previous case, but the rollback (also of intermediate states) is always performed in a single step. The forward reduction relation is defined by the same rules of case (2), where $\twoheadrightarrow_{(3)}$ replaces $\twoheadrightarrow_{(2)}$, while the backward one is defined by the following additional rules:

$$\langle s : P \rangle \blacktriangleright Q \rightsquigarrow_{(3)} P \quad [\textsc{Bw}(3)\text{-}1] \qquad \langle s : P \cdot m \rangle \blacktriangleright Q \rightsquigarrow_{(3)} \langle s : m \rangle \blacktriangleright P \quad [\textsc{Bw}(3)\text{-}2]$$

$$\langle s : m \cdot P \rangle \blacktriangleright Q \rightsquigarrow_{(3)} P \; [\textsc{Bw}(3)\text{-}3] \qquad \langle s : m' \cdot P \cdot m \rangle \blacktriangleright Q \rightsquigarrow_{(3)} \langle s : m \rangle \blacktriangleright P \; [\textsc{Bw}(3)\text{-}4]$$

Rules [BW(3)-1] and [BW(3)-2] are like [BW(2)-1] and [BW(2)-2], respectively. In particular, rule [BW(3)-2] is still used to replace the current state by the previous one. In addition to this, now rule [BW(3)-4] permits to replace the current state Q also by an intermediate state P of the session computation; this is done in a single step. Notice that all interactions that took place after the one produced by P (i.e., the states stored in m') are erased when P is restored, while the previous interactions (i.e., the states stored in m) are kept. Notice also that the selection of the past state to restore is non-deterministic. A real-world reversible language instead should provide specific primitives and mechanisms to control reversibility (see discussion on Section 5); anyway the controlled selection of the past states does not affect the reversibility costs, hence this aspect is out-of-scope for this paper and left for future investigations. Rule [BW(3)-3] permits to directly undo the whole session from an intermediate state.

Results. The cost of reverting a session in setting (1), in terms of both backward reductions ($\mathcal{C}_{br}$) and memory occupancy ($\mathcal{C}_{mo}$), is *constant* w.r.t. the session length. In case (2), instead, the costs are linear in the length of the session (recall that we consider the worst case, where the session is completely reversed). Finally, in setting (3) the cost is constant in terms of backward reductions, and linear in terms of memory occupancy.

Theorem 1. *Let n be the length of a session, the costs of reverting it are: case (1) $\mathcal{C}_{br} = \mathcal{C}_{mo} = 1$; case (2) $\mathcal{C}_{br} = \mathcal{C}_{mo} = n$; case (3) $\mathcal{C}_{br} = 1$ and $\mathcal{C}_{mo} = n$.*

Proof. The proof of case (1) is straightforward, while proofs of cases (2) and (3) proceed by induction on n (see [21]).

The following result shows that single binary sessions of cases (2) and (3) enjoy a standard property of reversible calculi (Loop lemma, see [7]): backward reductions are the inverse of the forward ones and vice versa.

Lemma 1 (Loop lemma). *Let $P = \langle s : m \rangle \blacktriangleright Q$ and $P' = \langle s : m' \rangle \blacktriangleright Q'$ be two reachable processes in setting (i), with $i \in \{2, 3\}$. $P \twoheadrightarrow_{(i)} P'$ if and only if $P' \rightsquigarrow_{(i)} P$.*

Proof. The proof for the *if* (resp. *only if*) part is by induction on the derivation of the forward (resp. backward) reduction (see [21]).

Notably, case (1) does not enjoy this lemma because backward reductions do not allow to restore intermediate states of sessions.

We conclude the section with an example showing the three approaches at work on the Buyer-Seller protocol.

Example 3 (Reversible Buyer-Seller protocol). Let us consider a reversible scenario concerning the Buyer-Seller protocol specified in Example 1, where there are two sellers and a buyer.

In case (1), the system evolves as follows:

$$\begin{array}{l} Seller_1 \mid Seller_2 \mid Buyer \\ \twoheadrightarrow_{(1)} \; Seller_1 \;\mid\; \langle s : Seller_2 \mid Buyer \rangle \blacktriangleright (s?(z_{title}).P_s \mid \bar{s}!\langle v_{title} \rangle.P_b) \\ \twoheadrightarrow^*_{(1)} \; Seller_1 \;\mid\; \langle s : Seller_2 \mid Buyer \rangle \blacktriangleright (Q[\ldots] \mid P[\ldots, v_{date}/x_{date}]) \;=\; R \end{array}$$

where v_{title} stands for "*The Divine Comedy*". After these interactions between $Buyer$ and $Seller_2$, the buyer has received a delivery date from the seller. In the unfortunate case that this date is not suitable for the buyer, the session can be reversed as follows:

$$R \rightsquigarrow_{(1)} Seller_1 \mid Seller_2 \mid Buyer$$

Now, $Buyer$ can start a new session with $Seller_2$ as well as with $Seller_1$.

In case (2), the parties can reach the same state as follows:

$$Seller_1 \mid Seller_2 \mid Buyer \twoheadrightarrow^*_{(2)} Seller_1 \;\mid\; \langle s : m \rangle \blacktriangleright (Q[\ldots] \mid P[\ldots, v_{date}/x_{date}]) \;=\; R'$$

where m is $R_{date} \cdot R_{addr} \cdot R_{ok} \cdot R_{if} \cdot R_{quote} \cdot R_{title}$, with R_i denoting the process generating the interaction i. In this case, the buyer can undo only the last two session interactions as follows:

$$R' \rightsquigarrow_{(2)} Seller_1 \mid \langle s : m' \rangle \blacktriangleright R_{date} \rightsquigarrow_{(2)} Seller_1 \mid \langle s : m'' \rangle \blacktriangleright R_{addr}$$

with $m' = R_{addr} \cdot m''$ and $m'' = R_{ok} \cdot R_{if} \cdot R_{quote} \cdot R_{title}$. Now, the buyer can possibly send a different address to the seller in order to get a more suitable date (as we assume $addr()$ and $date()$ be two non-deterministic functions abstracting the interaction with buyer and seller backends).

Finally, in case (3), the system can reach again the state R', but this time the session can be also partially reversed by means of a single backward step:

$$R' \rightsquigarrow_{(3)} Seller_1 \mid \langle s : m'' \rangle \blacktriangleright R_{addr}$$

4 Reversibility of single multiparty sessions

For the same motivations of the binary case, we do not consider all processes allowed by the syntax of the reversible multiparty single-session calculus, obtained by extending the grammar in Figure 7 with the single sessions construct in Figure 9. Again, we consider only reachable processes, whose definition relies on the notion of simple process.

Definition 3 (Multiparty simple process). *A multiparty process is* simple *if (i) it is generated by the grammar in Figure 7 and (ii) it is typable with a type derivation using the prefix rules where the session typings in the premise and the conclusion are restricted to at most a singleton*[5].

Definition 4 (Multiparty reachable processes). *The set of* reachable *processes, for the case (i) in Figure 1 with $i \in \{4, 5, 6\}$, is the closure under relation $\twoheadrightarrow_{(i)}$ (see below) of the set of multiparty simple processes.*

We now present the semantics of the three multiparty cases. For the definition of the reduction relations we still rely on the shared rules in Figure 10.

(4) Whole session reversibility. In case the reversibility machinery only permits to undo a whole session, the forward reduction relation is defined by these additional rules:

$$P \twoheadrightarrow_{(4)} \langle s : P\rangle \blacktriangleright (P_n[s[n]/x] \mid \textstyle\prod_{i=\{1,..,n-1\}} P_i[s[i]/x]) \qquad [\text{F}\textsc{w}(4)\text{-M-C}\textsc{on}]$$
$$P = (\bar{a}[n](x).P_n \mid \textstyle\prod_{i=\{1,..,n-1\}} a[i](x).P_i)$$

$$\frac{P \rightarrow P'}{\langle s : m\rangle \blacktriangleright P \twoheadrightarrow_{(4)} \langle s : m\rangle \blacktriangleright P'} \; [\text{F}\textsc{w}(4)\text{-M}\textsc{em}]$$

The meaning of these rules is similar to that of [FW(1)-CON] and [FW(1)-MEM], with the only difference that the initialised single session is multiparty.

The backward reduction relation is given by the same rules of case (1), of course defined for relation $\rightsquigarrow_{(4)}$ instead of $\rightsquigarrow_{(1)}$.

(5) Multi-step. When sessions can be reversed also partially, in a step-by-step fashion, the additional forward rules are as follows:

$$P \twoheadrightarrow_{(5)} \langle s : P\rangle \blacktriangleright (P_n[s[n]/x] \mid \textstyle\prod_{i=\{1,..,n-1\}} P_i[s[i]/x]) \qquad [\text{F}\textsc{w}(5)\text{-M-C}\textsc{on}]$$
$$P = (\bar{a}[n](x).P_n \mid \textstyle\prod_{i=\{1,..,n-1\}} a[i](x).P_i)$$

$$\frac{P \rightarrow P'}{\langle s : m\rangle \blacktriangleright P \twoheadrightarrow_{(5)} \langle s : P \cdot m\rangle \blacktriangleright P'} \; [\text{F}\textsc{w}(5)\text{-M}\textsc{em}]$$

[5] Using the typing system for the synchronous multiparty session π-calculus (see [21, Figure 15]), point (ii) boils down to: Δ of rules [MREQ], [MACC], [SEND], [RECV], [SEL] and [BRA] are empty; neither [DELEG] nor [SRECV] is used; $\Delta \cdot \Delta'$ in [CONC] contains at most a singleton; and Δ of the remaining rules contain at most a singleton.

These rules are the natural extension of the corresponding rules of the binary version, i.e. case (2). Their meaning, indeed, is the same.

The backward reduction relation in setting (5) is given by the same rules of case (2) where relation $\rightsquigarrow_{(2)}$ is replaced by $\rightsquigarrow_{(5)}$.

It is worth noticing that, by Definitions 3 and 4, concurrent interactions along the same session are prevented (by the type discipline in [12], which forces a linear use of session channels). Therefore, there is no need here to use a more complex reversible machinery (as in [20]) for enabling a causal-consistent form of session reversibility.

(6) Single-step. As in the corresponding binary session setting, here the forward reduction relation is defined by the same rules of the multi-step case, i.e. case (5), where $\twoheadrightarrow_{(6)}$ replaces $\twoheadrightarrow_{(5)}$. Instead, the backward reduction relation is defined by the same rules of case (3), where $\rightsquigarrow_{(6)}$ replaces $\rightsquigarrow_{(3)}$.

Results. The cost of reverting a session in setting (4) is constant, while it is linear in the length of the session in case (5). In setting (6), instead, the cost is constant in terms of backward reductions, and linear in terms of memory occupancy.

Theorem 2. *Let n be the length of a session, the costs of reverting it are: case (4) $\mathcal{C}_{br} = \mathcal{C}_{mo} = 1$; case (5) $\mathcal{C}_{br} = \mathcal{C}_{mo} = n$; case (6) $\mathcal{C}_{br} = 1$ and $\mathcal{C}_{mo} = n$.*
Proof. The proof of case (4) is a trivial adaptation of the proof of case (1) in Theorem 1, while proofs of cases (5) and (6) proceed by induction on n (see [21]).

In all multiparty approaches, cases (4)-(6), the backward computations have the same semantics of the corresponding binary approaches, cases (1)-(3), respectively. An important consequence of this fact is that the cost of reverting a session in a multiparty case is the same of the corresponding binary case. In other words, we can claim that *extending binary sessions to multiparty ones, in the single session setting, does not affect the machinery for the reversibility and its costs.*

As in the binary case, also the multiparty sessions of cases (5) and (6) enjoy the Loop lemma.

Lemma 2 (Loop lemma). *Let $P = \langle s : m\rangle \blacktriangleright Q$ and $P' = \langle s : m'\rangle \blacktriangleright Q'$ be two reachable processes in setting (i), with $i \in \{5, 6\}$. $P \twoheadrightarrow_{(i)} P'$ if and only if $P' \rightsquigarrow_{(i)} P$.*

Proof. The proof for the *if* (resp. *only if*) part is by induction on the derivation of the forward (resp. backward) reduction (see [21]).

We conclude by showing the three multiparty approaches at work on the Two-Buyers-Seller protocol.

Example 4 (Reversible Two-Buyers-Seller protocol). We consider a reversible scenario of the Two-Buyers-Seller session protocol specified in Example 2.

In case (4), the system can evolve as follows:

$$\begin{array}{l}
\mathit{Buyer}_1 \mid \mathit{Buyer}_2 \mid \mathit{Seller} \\
\twoheadrightarrow_{(4)} \ \langle s : \mathit{Buyer}_1 \mid \mathit{Buyer}_2 \mid \mathit{Seller}\rangle \blacktriangleright \\
\qquad (s[3][1]!\langle v_{title}\rangle.\, s[3][1]?(x_{quote}).\, P_{b1} \ \mid \ s[2][1]?(y_{quote}).\, P_{b2} \\
\qquad \mid \ s[1][3]?(z_{title}).\, s[1][2]!\langle \mathit{quote}(z_{title})\rangle.\, s[1][3]!\langle \mathit{lastQuote}(z_{title})\rangle.\, P_s) \\
\twoheadrightarrow^{*}_{(4)} \ \langle s : \mathit{Buyer}_1 \mid \mathit{Buyer}_2 \mid \mathit{Seller}\rangle \blacktriangleright \\
\qquad (P_{b1}[v_{quote}/x_{quote}] \mid P_{b2}[v_{quote}/y_{quote}] \mid P_{b1}[v_{title}/x_{title}]) \ = \ R
\end{array}$$

These interactions lead to a state where both buyers have received the seller's quote for the requested book. Now, if one of the two buyers is not satisfied with the proposed quote, he can immediately stop the session execution and reverse it with a single step:

$$R \rightsquigarrow_{(4)} Buyer_1 \mid Buyer_2 \mid Seller$$

In case (5), instead, the protocol execution can lead to a similar state, say R', with the difference that the session memory m keeps track of the traversed states, i.e. m is $R_{quote2} \cdot R_{quote1} \cdot R_{title}$. In this case, the unsatisfied buyer can enact a sort of negotiation by undoing the last two session interactions:

$$R' \rightsquigarrow_{(5)} \langle s : R_{quote1} \cdot R_{title} \rangle \blacktriangleright R_{quote2} \rightsquigarrow_{(5)} \langle s : R_{title} \rangle \blacktriangleright R_{quote1}$$

From the state R_{quote1} the seller can compute again the quote for the requested book.

Finally, in case (6), the system can reach again the state R' and, likewise case (3), the session can be partially reversed by means of a single backward step:

$$R' \rightsquigarrow_{(6)} \langle s : R_{title} \rangle \blacktriangleright R_{quote1}$$

5 Concluding remarks

This work falls within a large body of research that aims at studying at foundational level the integration of reversibility in concurrent and distributed systems. In particular, reversible variants of well-established process calculi, such as CCS and π-calculus, have been proposed as *untyped* formalisms for studying reversibility mechanisms in these systems. Relevant works along this line of research have been surveyed in [15]. Among them, we would like to mention the works that are closely related to ours, as they have been source of inspiration: RCCS [7], from which we borrow the use of memory stacks for keeping track of computation history; $R\pi$ [6], from which we borrow the notion of reachable process (see Definition 2); $R\mu Oz$ [16], which analyses reversibility costs in terms of space overhead; and $\rho\pi$ [14], from which we borrow the use of a reversible reduction semantics (which is motivated by the fact that a labelled semantics would complicate our theoretical framework). However, all works mentioned above only focus on causal-consistent reversibility mechanisms for untyped concurrent systems, without taking into account how they may impact on linearity-based structured interactions, which is indeed our aim. Moreover, none of the above work provides a systematic study of the different forms of reversibility we consider, namely whole session, multi-step and single-step, and of their costs.

The works with the aim closest to ours are [2] and [20]. The former paper studies session reversibility on a formalism based on session behaviours [1], which is a sort of sub-language of CCS with a checkpoint-based backtracking mechanism. The commonality with our work is the use of a one-size memory for each behaviour, which records indeed only the behaviour prefixed by the lastly traversed checkpoint. This resembles the one-size memories that we use in cases (1) and (4), with the difference that our checkpoints correspond to the initialisation states of sessions. On the other hand, session behaviours provide a formalism much simpler than session-based π-calculus, as e.g. message passing is not even considered. Differently from our work, [2] does not consider different solutions for enabling alternative forms of reversibility and does not provide a study of session reversibility complexity. The latter paper introduces $ReS\pi$, a reversible variant of the π-calculus with binary multiple sessions. $ReS\pi$ embeds a multi-step form of reversibility and, rather than using a single stack memory per session, it uses a graph-like data structure and unique thread identifiers. Each element of this structure is devoted to record data concerning a single event, corresponding to the taking place of a communication action, a choice or a thread forking. Thread identifiers are used as links among memory elements, in order to form a structure for conveniently keeping track of the causal dependences among the session interactions. These dependences are crucial in the multiple session setting, where computations have to be undone in a *causal-consistent* fashion [7,3], that is independent concurrent interactions can be undone in an order different from that of the forward computation. Differently from the present work, which considers both binary and multiparty session types, [20] only focusses on the binary version. In addition, it does not address any cost issues about reversible sessions.

We plan to study the cost of reversing multiple sessions, where interactions along different sessions can be interleaved. By looking at $ReS\pi$, we can see that passing from single sessions to multiple ones has significant impacts: firstly, in terms of complexity of the memory structure, and secondly in terms of costs. In the multiple case, reverting a session corresponds to revert a concurrent computation in a causal-consistent way, which requires to revert all interactions performed along other sessions that have a casual dependence with the interactions of the session to be reverted. This means that, in general, the cost is not defined only in terms of the length of the considered session, but it must include also the cost of reverting the depending interactions of other sessions.

Another future direction that we plan to consider for our study concerns how the use of primitives and mechanisms to *control* reversibility (see, e.g., [13]) affect our results. In a controlled approach for session reversibility, backward steps would not be always enabled, but they would be triggered by specific rollback actions.

Moreover, we intend to extend our analysis on memory cost. In fact, the approach used in this work is *coarse-grained*, as it is based on the number of elements of the stacks rather than on the amount of memory necessary for storing such elements. A fine-grained view is indeed not necessary for the purpose of this work, as we just want to compare the different combinations of session and reversibility approaches and, in particular, to distinguish the whole session reversibility with respect to the other cases, and we want to show that single-step interaction reversibility and multi-step ones require memory with the same number of elements. Nevertheless, a fine-grained analysis on memory cost and an investigation of more compact representations of computation history would be interesting extensions of this work.

Finally, the enactment of reversibility is currently based on the information stored in the syntactical terms representing the involved processes. We plan to investigate the use of type information to enact and manage reversibility.

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.

2. Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Compliance for reversible client/server interactions. In *BEAT*, volume 162 of *EPTCS*, pages 35–42, 2014.
3. Gérard Berry and Jean-Jacques Lévy. Minimal and Optimal Computations of Recursive Programs. *J. ACM*, 26(1):148–175, 1979.
4. Luca Cardelli and Cosimo Laneve. Reversible structures. In *CMSB*, pages 131–140. ACM, 2011.
5. Mario Coppo, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Asynchronous Session Types and Progress for Object Oriented Languages. In *FMOODS*, volume 4468 of *LNCS*, pages 1–31. Springer, 2007.
6. Ioana Cristescu, Jean Krivine, and Daniele Varacca. A Compositional Semantics for the Reversible p-Calculus. In *LICS*, pages 388–397. IEEE, 2013.
7. Vincent Danos and Jean Krivine. Reversible communicating systems. In *CONCUR*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.
8. Vincent Danos and Jean Krivine. Transactions in rccs. In *CONCUR*, volume 3653 of *LNCS*, pages 398–412. Springer, 2005.
9. Vincent Danos and Jean Krivine. Formal Molecular Biology Done in CCS-R. *Electr. Notes Theor. Comput. Sci.*, 180(3):31–49, 2007.
10. Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.
11. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 2015. To appear. Available at `http://mrg.doc.ic.ac.uk`. An extended abstract appeared in the Proc. of POPL'08.
12. Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics. *Logical Methods in Computer Science*, 10(4), 2014.
13. 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.
14. Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversing higher-order pi. In *CONCUR*, volume 6269 of *LNCS*, pages 478–493. Springer, 2010.
15. Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-Consistent Reversibility. *Bulletin of the EATCS*, 114:121–139, 2014.
16. Michael Lienhardt, Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. A reversible abstract machine and its space overhead. In *FMOODS/FORTE*, volume 7273 of *LNCS*, pages 1–17. Springer, 2012.
17. Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40, 41–77, 1992.
18. Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015.
19. Iain C.C. Phillips and Irek Ulidowski. Reversing algebraic process calculi. *J. Log. Algebr. Program.*, 73(1-2):70–96, 2007.
20. Francesco Tiezzi and Nobuko Yoshida. Reversible session-based pi-calculus. *J. Log. Algebr. Meth. Program.*, 84(5):684–707, 2015.
21. Francesco Tiezzi and Nobuko Yoshida. Reversing single sessions. *CoRR*, abs/1510.07253, 2015.
22. Nobuko Yoshida and Vasco Thudichum Vasconcelos. Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication. *Electr. Notes Theor. Comp. Sci.*, 171(4):73–93, 2007.