Rollback Recovery in Session-based Programming

Claudio Antares Mezzina[1], Francesco Tiezzi[2], and Nobuko Yoshida[3]

[1] University of Leicester, United Kingdom
cam78@leicester.ac.uk
[2] University of Camerino, Italy
francesco.tiezzi@unicam.it
[3] Imperial College London, United Kingdom
n.yoshida@imperial.ac.uk

Abstract. To react to unforeseen circumstances or amend abnormal situations (e.g., runtime errors) in communication-centric systems, programmers are in charge of "undoing" the interactions which led to an undesired state. To assist this task, session-based languages can be endowed with reversibility mechanisms. In this paper we propose a language enriched with programming facilities to *commit* session interactions, to *roll back* the computation to a previous commit point, and to *abort* the session. We prove that rollbacks in our language always bring the system to previous visited states and that a rollback cannot bring the system back to a point prior to the last commit. Programmers are relieved from the burden of ensuring that a rollback never restores a checkpoint imposed by a session participant different from the rollback requester. Such undesired situations are prevented at design-time (statically) by relying on a decidable *compliance* check at the type level, implemented using MAUDE. We show that the language satisfies error-freedom and progress of a session, demonstrate the practical effectiveness of the proposed approach by a means of a speculative execution scenario from the reversibility literature, and illustrate how the approach applies to both binary and multiparty sessions.

Keywords: Reversible Computing · Session Types · Recovery · Rollback · Checkpoints · MAUDE

1 Introduction

Reversible computing, whose origin dates back to the 70's for circuits energy efficiency, is now attracting interest for its application to different fields: from modelling biological/chemical phenomena [18], to simulation [30], debugging [12] and modelling fault-tolerant systems [11,19]. Our interest focusses on this latter application and, in particular, stems from the fact that reversibility can be used to rigorously model, implement and revisit programming abstractions for reliable software systems.

Recent works [3,27,26,5,32] have studied the effect of reversibility in communication-centric scenarios, as a means to amend computations that led to an error by bringing back the system to a consistent previous state. In this setting, processes' behaviours are strongly disciplined by their types, prescribing the actions they have to perform within a *session*. A session consists in a structured series of message exchanges, whose flow can be controlled via conditional choices, branching and recursion. Correctness of communication is statically guaranteed by a framework based on a (session) type discipline [16]. Nonetheless, none of these works addresses systems in which the participants can *explicitly* abort the session, commit a computation and roll it back to a previous checkpoint. In this paper, we aim at filling this gap in programming reversible behaviours within communication-centric systems. We explain below the distinctive aspects of our checkpoint-based rollback recovery approach.

- **Linguistic primitives for explicitly and practically programming reversible sessions.** We introduce three primitives to: (i) *commit* a session, preventing undoing the interactions performed so far along the session; (ii) *roll back* a session, restoring the last saved process checkpoints; (iii) *abort* a session, to discard the session, and hence all interactions already performed in it, thus allowing another session of the same protocol to start with possible different participants. Notice that most proposals in the literature (e.g., [1,2,3]) only consider an abstract view, as they focus on reversible contracts (i.e., types). Instead, we focus on programming primitives at process level, and use types for guaranteeing a safe and consistent system evolution.
- **Asynchronous commits.** Our commit primitive does not require a session-wide synchronisation among all participants, as it is a local decision. However, its effect is on the whole session, as it affects the other session participants. This means that each participant can independently decide when to commit. Such flexibility comes at the cost of being error-prone, especially considering that the programmer has not only to deal with the usual forward executions, but also with the backward ones. Our type discipline allows for ruling out programs which may lead to these errors.

The key idea of our approach is that “a session participant executing a rollback action is interested in restoring the last checkpoint he/she has committed”. For the success of the rollback recovery it is irrelevant whether the ‘passive’ participants go back to their own last checkpoints. Instead, if the ‘active’ participant is unable to restore the last checkpoint he/she has created, because it has been replaced by a checkpoint imposed by another participant, the rollback recovery is considered unsatisfactory.

In our framework, programmers are relieved from the burden of ensuring the satisfaction of rollbacks, since undesired situations are prevented at design time (statically) by relying on a *compliance* check at the type level. To this aim, we introduce `cherry-pi` (checkpoint-based rollback recovery pi-calculus), a variant of the session-based π-calculus [38,17] enriched with rollback recovery primitives. We present first a binary version of the calculus, which is more convenient to demonstrate the essence of our rollback recovery approach, and then we extend it to multiparty sessions. A key difference with respect to the standard binary type discipline is the *relaxation* of the duality requirement. The types of two session participants are not required to be dual, but they will be compared with respect to a compliance relation (as in [2]), which also takes into account the effects of commit and rollback actions. Such relaxation also involves the requirements concerning selection and branching types, and those concerning branches of conditional choices. The `cherry-pi` type system is used to infer types of session participants, then combined together for the compliance check.

Reversibility in `cherry-pi` is *controlled* via two specific primitives: a rollback one telling when a reverse computation has to take place, and a commit one limiting the scope of a potential reverse computation. This implies that the calculus is not fully reversible (i.e., backward computations are not always enabled), leading to have relaxed and different properties with respect to other reversible calculi, such as RCCS [10], $R\pi$ [9], $\rho\pi$ [21,22] and *ReS*π [32]. We prove that cherry-pi satisfies the following properties: (i) a rollback always brings back the system to a previous visited state and (ii) it is not possible to bring the computation back to a point prior to the last checkpoint, which implies that our commits have a persistent effect. Concerning soundness properties, we prove that (a) our compliance check is decidable, and that compliance-checked cherry-pi specifications (b) never reduce to communication errors (e.g., a blocked communication where there is a receiver without the corresponding sender) and (c) never activate undesirable rollbacks (according to our notion of rollback recovery mentioned above). Property (b) resembles the type safety property of session-based calculi (see, e.g., [38]), while property (c) is a new property specifically defined for cherry-pi. The technical development of property proofs turns out to be more intricate than that of standard properties of session-based calculi, due to the combined use of type and compliance checking.

To demonstrate feasibility and effectiveness of our rollback recovery approach, we have concretely implemented[4] the compliance check using the MAUDE framework [7].

Outline. Sec. 2 illustrates the key idea of our rollback recovery approach; Sec. 3 introduces syntax and semantics of the cherry-pi calculus; Sec. 4 introduces typing and compliance checking; Sec. 5 presents the properties satisfied by cherry-pi; Sec. 6 shows an application of cherry-pi to a practical case study [31,14]; Sec. 7 describes how the approach extends to multiparty sessions; Sec. 8 concludes the paper with related and future work. The Appendix reports the proofs of our results.

2 A reversible video on demand service example

We discuss the motivations underlying our work by introducing our running example, a Video on Demand (VOD) scenario. The key idea is that a rollback requester is satisfied only if her restored checkpoint was set by herself. In Fig. 1(a), a service (S) offers to a user (U) videos with two different quality levels, namely high definition (HD) and standard definition (SD). After the *login*, U sends her video *request*, and receives the corresponding *price* and *metadata* (actors, directors, description, etc.) from S. According to this information, U selects the video quality. Then, she receives, first, a short *test* video (to check the audio and video quality in her device) and, finally, the requested *video*. If the vision of the HD test video is not satisfactory, U can roll back to her last checkpoint to possibly change the video quality, instead in the SD case U can abort the session.

Let us now add commit actions as in the run shown in Fig. 1(b). After receiving the price, U commits, while S commits after the quality selection. In this scenario, however, if U activates the rollback, she is unable to go back to the checkpoint she set with her commit action because the actual effect of rollback is to restore the checkpoint set by the commit action performed by S.

[4] Our MAUDE implementation of the compliance check is available at the cherry-pi's website: http://pros.unicam.it/tools/cherry-pi.

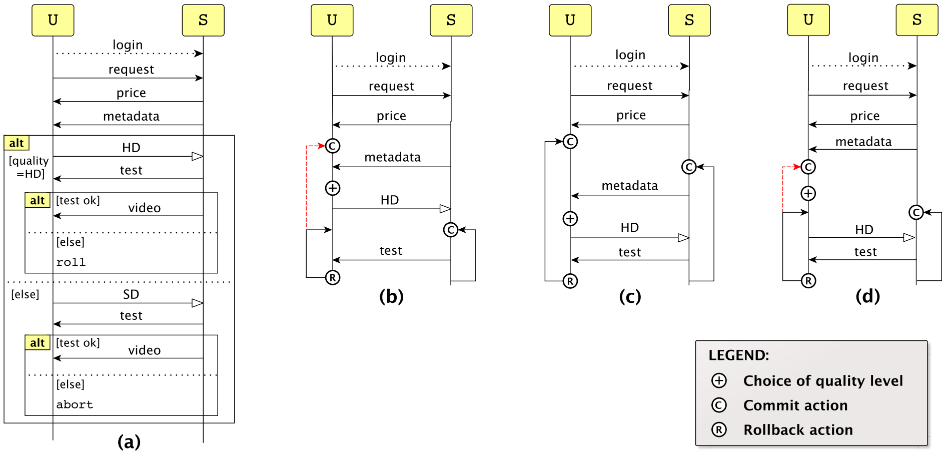

Fig. 1. VOD example: (a) a full description without commit actions; (b,d) runs with undesired rollback; (c) a run with satisfactory rollback.

In the scenario in Fig. 1(c), instead, `S` commits after sending the price to `U`. In this case, no matter who first performed the commit action, the rollback results to be satisfactory. Also if `S` commits later, the checkpoint of `U` remains unchanged, as `U` performed no other action between the two commit actions. This would not be the case if both `U` and `S` commited after the communication of the metadata, as in Fig. 1(d). If `S` commits before `U`, no rollback issue arises, but if `U` commits first it may happen that her internal decision is taken before that `S` commits. In this case, `U` would not be able to go back to the checkpoint set by herself, and she would be unable to change the video quality.

These undesired rollbacks are caused by bad choices of commit points. We propose a compliance check that identifies these situations at design time, avoiding unnecessary checks at runtime.

3 The `cherry-pi` calculus

In this section, we introduce `cherry-pi`, a calculus (extending that in [38]) devised for studying sessions equipped with our checkpoint-based rollback recovery mechanism.

3.1 Syntax

The syntax of the `cherry-pi` calculus relies on the following base sets: *shared channels* (ranged over by a), used to initiate sessions; *session channels* (ranged over by s), consisting of pairs of *endpoints* (ranged over by s, $\bar{s}$) used by the two parties to interact within an established session; *labels* (ranged over by l), used to select and offer branching choices; *values* (ranged over by v), including booleans, integers and strings (whose *sorts*, ranged over by S, are `bool`, `int` and `str`, respectively), which are exchanged within a

$C ::=$	**Collaborations**
$\bar{a}(x).P \mid a(x).P \mid C_1 \,|\, C_2$	request, accept, parallel
$P ::=$	**Processes**
$x!\langle e\rangle.P \mid x?(y:S).P$	output, input
$\mid x \lhd l.P \mid x \rhd \{l_1 : P_1, \ldots, l_n : P_n\}$	selection, branching
$\mid \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \mid X \mid \mu X.P \mid \mathbf{0}$	choice, recursion, inact
$\mid \mathsf{commit}.P \mid \mathsf{roll} \mid \mathsf{abort}$	commit, roll, abort
$e ::= v \mid +(e_1, e_2) \mid \wedge(e_1, e_2) \mid \ldots$	**Expressions**

Fig. 2. `cherry-pi` syntax.

session; *variables* (ranged over by x, y, z), storing values and session endpoints; *process variables* (ranged over by X), used for recursion.

`cherry-pi` specifications, called *collaborations* and ranged over by C, are given by the grammar in Fig. 2. The key ingredient of the calculus is the set of actions for controlling the session rollback. Actions commit, roll and abort are used, respectively, to commit a session (producing a checkpoint for each session participant), to trigger the session rollback (restoring the last committed checkpoints) or to abort the whole session. We discuss below the other constructs of the calculus, which are those typically used for session-based programming [15].

A `cherry-pi` collaboration is a collection of *session initiators*, i.e. terms ready to initiate sessions by synchronising on shared channels. A synchronisation of two initiators $\bar{a}(x).P$ and $a(y).Q$ causes the generation of a fresh session channel, whose endpoints replace variables x and y in order to be used by the triggered processes P and Q, respectively, for later communications. No subordinate sessions can be initiated within a running session.

When a session is started, each participant executes a *process*. Processes are built up from the process $\mathbf{0}$, which does nothing, and from basic actions by means of action prefix $\_ . \_$, conditional choice $\mathsf{if}\ e\ \mathsf{then}\ \_\ \mathsf{else}\ \_$, and recursion $\mu X.\_$. Actions $x!\langle e\rangle$ and $y?(z:S)$ denote output and input via session endpoints replacing x and y, respectively. These communication primitives realise the standard synchronous message passing, where messages result from the evaluation of *expressions*, which are defined by means of standard operators on boolean, integer and string values. Variables that are arguments of input actions are (statically) typed by sorts. There is no need for statically typing the variables occurring as arguments of session initiating actions, as they are always replaced by session endpoints. Notice that in `cherry-pi` the exchanged values cannot be endpoints, meaning that session delegation (i.e., channel-passing) is not considered. Actions $x \lhd l$ and $x \rhd \{l_1 : P_1, \ldots, l_n : P_n\}$ denote selection and branching (where l_1, $\ldots$, l_n are pairwise distinct).

Example 1. Let us consider the VOD example informally introduced in Sec. 2. The scenario described in Fig. 1(a) with commit actions placed as in Fig. 1(b) is rendered in `cherry-pi` as

$$C_{\mathsf{US}} \;=\; \overline{login}(x).\,P_{\mathsf{U}} \;\mid\; login(y).\,P_{\mathsf{S}}$$

$$
\begin{array}{llr}
C ::= \bar{a}(x).P \mid a(x).P \mid C_1 \,|\, C_2 \mid (\nu s : C_1)\; C_2 \mid \langle P_1 \rangle \blacktriangleright P_2 & & \textbf{Collaborations} \\
P ::= r\,!\langle e \rangle.P \mid r\,?(y : S).P \mid r \lhd l.P \mid r \rhd \{l_1 : P_1, \ldots, l_n : P_n\} \mid \cdots & & \textbf{Processes}
\end{array}
$$

Fig. 3. `cherry-pi` runtime syntax (the rest of processes P and expressions e are as in Fig. 2).

where:

$$
\begin{array}{rl}
P_{\mathrm{U}} \;=\; & x!\langle v_{\mathit{req}} \rangle.\, x?(x_{\mathit{price}} : \mathtt{int}).\, \mathsf{commit}.\, x?(x_{\mathit{meta}} : \mathtt{str}).\, \mathsf{if}\ (f_{\mathit{eval}}(x_{\mathit{price}}, x_{\mathit{meta}})) \\
& \quad \mathsf{then}\ x \lhd l_{\mathit{HD}}.\, x?(x_{\mathit{testHD}} : \mathtt{str}). \\
& \qquad (\mathsf{if}\ (f_{\mathit{HD}}(x_{\mathit{testHD}}))\ \mathsf{then}\ x?(x_{\mathit{videoHD}} : \mathtt{str}).\mathbf{0}\ \mathsf{else}\ \mathsf{roll}) \\
& \quad \mathsf{else}\ x \lhd l_{\mathit{SD}}.\, x?(x_{\mathit{testSD}} : \mathtt{str}). \\
& \qquad (\mathsf{if}\ (f_{\mathit{SD}}(x_{\mathit{testSD}}))\ \mathsf{then}\ x?(x_{\mathit{videoSD}} : \mathtt{str}).\mathbf{0}\ \mathsf{else}\ \mathsf{abort})
\end{array}
$$

$$
\begin{array}{rl}
P_{\mathrm{S}} \;=\; & y?(y_{\mathit{req}} : \mathtt{str}).\, y!\langle f_{\mathit{price}}(y_{\mathit{req}}) \rangle.\, y!\langle f_{\mathit{meta}}(y_{\mathit{req}}) \rangle. \\
& y \rhd \{\ l_{\mathit{HD}} : \mathsf{commit}.\, y!\langle f_{\mathit{testHD}}(y_{\mathit{req}}) \rangle.\, y!\langle f_{\mathit{videoHD}}(y_{\mathit{req}}) \rangle.\, \mathbf{0}\ , \\
& \qquad l_{\mathit{SD}} : \mathsf{commit}.\, y!\langle f_{\mathit{testSD}}(y_{\mathit{req}}) \rangle.\, y!\langle f_{\mathit{videoSD}}(y_{\mathit{req}}) \rangle.\, \mathbf{0}\ \}
\end{array}
$$

Notice that expressions used for decisions and computations are abstracted by functions $f_n(\cdot)$.

Considering the placement of commit actions depicted in Fig. 1(c), the `cherry-pi` specification of the service's process becomes:

$$
\begin{array}{l}
y?(y_{\mathit{req}} : \mathtt{str}).\, y!\langle f_{\mathit{price}}(y_{\mathit{req}}) \rangle.\, \mathsf{commit}.\, y!\langle f_{\mathit{meta}}(y_{\mathit{req}}) \rangle. \\
y \rhd \{\ l_{\mathit{HD}} : y!\langle f_{\mathit{testHD}}(y_{\mathit{req}}) \rangle.\, y!\langle f_{\mathit{videoHD}}(y_{\mathit{req}}) \rangle.\, \mathbf{0}\ , \\
\qquad l_{\mathit{SD}} : y!\langle f_{\mathit{testSD}}(y_{\mathit{req}}) \rangle.\, y!\langle f_{\mathit{videoSD}}(y_{\mathit{req}}) \rangle.\, \mathbf{0}\ \}
\end{array}
$$

Finally, considering the placement of commit actions depicted in Fig. 1(d), the `cherry-pi` specification of the user's process becomes:

$$
x!\langle v_{\mathit{req}} \rangle.\, x?(x_{\mathit{price}} : \mathtt{int}).\, x?(x_{\mathit{meta}} : \mathtt{str}).\, \mathsf{commit}.\, \mathsf{if}\ (f_{\mathit{eval}}(x_{\mathit{price}}, x_{\mathit{meta}}))\ \mathsf{then}\ \ldots
$$

3.2 Semantics

The operational semantics of `cherry-pi` is defined for *runtime* terms, generated by the extended syntax of the calculus in Fig. 3 (where the new constructs are highlighted by a grey background). We use k to denote *generic session endpoints* (s or $\bar{s}$); and r to denote *session identifiers*, i.e. session endpoints and variables together. We refer as *initial collaborations* to those runtime terms that can be also generated by the grammar in Fig. 2.

At collaboration level, two constructs are introduced: $(\nu s : C_1)\; C_2$ represents a *session* along the channel s with associated starting checkpoint C_1 (corresponding to the collaboration that has initialised the session) and code C_2; construct $\langle P_1 \rangle \blacktriangleright P_2$ represents a *log* storing the checkpoint P_1 associated to the code P_2. At process level, the only difference is that session identifiers r are used as first argument of communicating actions. We extend the standard notion of binders to take into account $(\nu s : C_1)\; C_2$, which binds session endpoints s and $\bar{s}$ in C_2 (in this respect, it acts similarly to the restriction of π-calculus, but its scope cannot be extended in order to avoid involving

$\twoheadrightarrow$ Forward reduction over collaborations $\rightsquigarrow$ Backward reduction over collaborations

$\rightarrowtail = (\twoheadrightarrow \cup \rightsquigarrow)$ Reduction over collaborations $\xrightarrow{\ell}$ Relation labelled by actions over processes

Fig. 4. `cherry-pi` transition relations.

$$k!\langle e\rangle.P \xrightarrow{k!\langle v\rangle} P \quad (e \downarrow v) \quad [\text{P-SND}] \qquad k?(x : S).P \xrightarrow{k?(x)} P \quad [\text{P-RCV}]$$

$$k \lhd l.P \xrightarrow{k \lhd l} P \quad [\text{P-SEL}] \qquad k \rhd \{l_1 : P_1, \ldots, l_n : P_n\} \xrightarrow{k \rhd l_i} P_i \quad (1 \leqslant i \leqslant n) \quad [\text{P-BRN}]$$

$$\frac{P_1 \equiv P_1' \xrightarrow{\ell} P_2' \equiv P_2}{P_1 \xrightarrow{\ell} P_2} \quad [\text{P-STR}] \qquad \begin{array}{ll} \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_1 \quad (e \downarrow \mathtt{true}) & [\text{P-IFT}] \\ \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2 \quad (e \downarrow \mathtt{false}) & [\text{P-IFF}] \end{array}$$

$$\mathsf{commit}.P \xrightarrow{cmt} P \quad [\text{P-CMT}] \qquad \mathsf{roll} \xrightarrow{roll} \mathbf{0} \quad [\text{P-RLL}] \qquad \mathsf{abort} \xrightarrow{abt} \mathbf{0} \quad [\text{P-ABT}]$$

Fig. 5. `cherry-pi` semantics: auxiliary labelled relation.

processes that do not belong to the session in the rollback effect). The derived notions of bound and free names (where *names* stand for variables, process variables and session endpoints), alpha-equivalence, and substitution are standard and we assume that bound names are pairwise distinct. The semantics of the calculus is defined for *closed* terms, i.e. terms without free variables and process variables.

Not all processes allowed by the extended syntax corresponds to meaningful collaborations. In a general term the processes stored in logs may not be consistent with the computation that has taken place. We get rid of such malformed terms, as we will only consider those runtime terms, called *reachable* collaborations, obtained by means of reductions from initial collaborations.

In `cherry-pi`, processes can perform, besides usual forward computations, also backward computations that restore the last committed checkpoints, thus undoing the effect of forward computations. The operational semantics is given in terms of a standard *structural congruence* [17], written $\equiv$, and a *reduction* relation, written $\rightarrowtail$, given as the union of the corresponding *forward reduction* relation $\twoheadrightarrow$ and *backward reduction* relations $\rightsquigarrow$. The definition of the relation $\twoheadrightarrow$ over closed collaborations relies on an auxiliary labelled relation $\xrightarrow{\ell}$ over processes that specifies the actions that processes can initially perform and the continuation process obtained after each such action. The notation of the above relations is summarised in Fig. 4. Given a reduction relation $\mathcal{R}$, we will indicate with $\mathcal{R}^+$ and $\mathcal{R}^*$ respectively the *transitive* and the *reflexive-transitive* closure of $\mathcal{R}$.

The operational rules defining the auxiliary labelled relation are reported in Fig. 5, where action label ℓ stands for either $k!\langle v\rangle$, $k?(x)$, $k \lhd l$, $k \rhd l$, cmt, $roll$, abt, or τ. The meaning of these rules is straightforward, as they just produce as labels the actions currently enabled in the process. In doing that, expressions of sending actions and conditional choices are evaluated (the auxiliary function $e \downarrow v$ says that closed expression e evaluates to value v).

The operational rules defining the reduction relation $\rightarrowtail$ are reported in Fig. 6. We comment on salient points. Once a session is created, its initiating collaboration is

$$\begin{array}{l}\bar{a}(x_1).P_1 \mid a(x_2).P_2 \twoheadrightarrow (\nu s : (\bar{a}(x_1).P_1 \mid a(x_2).P_2)) \\ \qquad\qquad (\langle P_1[\bar{s}/x_1]\rangle \blacktriangleright P_1[\bar{s}/x_1] \mid \langle P_2[s/x_2]\rangle \blacktriangleright P_2[s/x_2])\end{array} \quad \text{[F-Con]}$$

$$\frac{P_1 \xrightarrow{\bar{k}!\langle v\rangle} P_1' \qquad P_2 \xrightarrow{k?(x)} P_2'}{\langle Q_1\rangle \blacktriangleright P_1 \mid \langle Q_2\rangle \blacktriangleright P_2 \;\twoheadrightarrow\; \langle Q_1\rangle \blacktriangleright P_1' \mid \langle Q_2\rangle \blacktriangleright P_2'[v/x]} \;\text{[F-Com]}$$

$$\frac{P_1 \xrightarrow{\bar{k}\lhd l} P_1' \qquad P_2 \xrightarrow{k\rhd l} P_2'}{\langle Q_1\rangle \blacktriangleright P_1 \mid \langle Q_2\rangle \blacktriangleright P_2 \;\twoheadrightarrow\; \langle Q_1\rangle \blacktriangleright P_1' \mid \langle Q_2\rangle \blacktriangleright P_2'} \;\text{[F-Lab]}$$

$$\frac{P_1 \xrightarrow{cmt} P_1'}{\langle Q_1\rangle \blacktriangleright P_1 \;\mid\; \langle Q_2\rangle \blacktriangleright P_2 \;\twoheadrightarrow\; \langle P_1'\rangle \blacktriangleright P_1' \;\mid\; \langle P_2\rangle \blacktriangleright P_2} \;\text{[F-Cmt]}$$

$$\frac{C_1 \equiv C_1' \;\twoheadrightarrow\; C_2' \equiv C_2}{C_1 \;\twoheadrightarrow\; C_2} \;\text{[F-Str]} \qquad\qquad \frac{P \xrightarrow{\tau} P'}{\langle Q\rangle \blacktriangleright P \;\twoheadrightarrow\; \langle Q\rangle \blacktriangleright P'} \;\text{[F-If]}$$

$$\frac{C_2 \;\twoheadrightarrow\; C_2'}{(\nu s : C_1)\, C_2 \;\twoheadrightarrow\; (\nu s : C_1)\, C_2'} \;\text{[F-Res]} \qquad\qquad \frac{C_1 \;\twoheadrightarrow\; C_1'}{C_1 \,|\, C_2 \;\twoheadrightarrow\; C_1' \,|\, C_2} \;\text{[F-Par]}$$

$$\frac{P_1 \xrightarrow{roll} P_1'}{\langle Q_1\rangle \blacktriangleright P_1 \;\mid\; \langle Q_2\rangle \blacktriangleright P_2 \;\rightsquigarrow\; \langle Q_1\rangle \blacktriangleright Q_1 \;\mid\; \langle Q_2\rangle \blacktriangleright Q_2} \;\text{[B-Rll]}$$

$$\frac{P_1 \xrightarrow{abt} P_1'}{(\nu s : C)(\langle Q_1\rangle \blacktriangleright P_1 \;\mid\; \langle Q_2\rangle \blacktriangleright P_2) \;\rightsquigarrow\; C} \;\text{[B-Abt]} \qquad \frac{C_1 \equiv C_1' \;\rightsquigarrow\; C_2' \equiv C_2}{C_1 \;\rightsquigarrow\; C_2} \;\text{[B-Str]}$$

$$\frac{C_2 \;\rightsquigarrow\; C_2'}{(\nu s : C_1)C_2 \;\rightsquigarrow\; (\nu s : C_1)C_2'} \;\text{[B-Res]} \qquad\qquad \frac{C_1 \;\rightsquigarrow\; C_1'}{C_1 \,|\, C_2 \;\rightsquigarrow\; C_1' \,|\, C_2} \;\text{[B-Par]}$$

Fig. 6. `cherry-pi` semantics: forward and backward reduction relations.

stored in the session construct (rule [F-Con]). Communication, branching selection and internal conditional choices proceed as usual, without affecting logs (rules [F-Com], [F-Lab] and [F-If]). A commit action updates the checkpoint of a session, by replacing the processes stored in the logs of the two involved parties (rule [F-Cmt]). Conversely, a rollback action restores the processes in the two logs (rule [B-Rll]). The abort action, instead, restores the collaboration stored in the session construct formed by the two initiators that have started the session (rule [B-Abt]). The other rules simply extend the standard parallel, restriction, structural congruence rules to forward and backward relations.

Example 2. Consider the first `cherry-pi` specification of the VOD scenario given in Ex. 1. In the initial state $C_{\texttt{US}}$ of the collaboration, `U` and `S` can synchronise in order to initialise the session, thus evolving to

$$C^1_{\texttt{US}} \;=\; (\nu s : C_{\texttt{US}})\; (\langle P_{\texttt{U}}[\bar{s}/x]\rangle \blacktriangleright P_{\texttt{U}}[\bar{s}/x] \;\mid\; \langle P_{\texttt{S}}[s/y]\rangle \blacktriangleright P_{\texttt{S}}[s/y])$$

Let us consider now a possible run of the session. After three reduction steps, `U` executes the commit action, obtaining the following runtime term:

$$\begin{aligned}
C^2_{\mathtt{US}} &= (\nu s : C_{\mathtt{US}})\ (\langle P'_{\mathtt{U}}\rangle \blacktriangleright P'_{\mathtt{U}} \ \mid\ \langle P'_{\mathtt{S}}\rangle \blacktriangleright P'_{\mathtt{S}}) \\
P'_{\mathtt{U}} &= \bar{s}?(x_{\mathit{meta}} : \mathtt{str}).\, \mathsf{if}\ (f_{\mathit{eval}}(v_{\mathit{price}}, x_{\mathit{meta}}))\ \mathsf{then}\ \ldots \\
P'_{\mathtt{S}} &= s!\langle f_{\mathit{meta}}(v_{\mathit{req}})\rangle.\, y \rhd \{\ \ldots\ \}
\end{aligned}$$

After four further reduction steps, `U` chooses the HD video quality and `S` commits as well; the resulting runtime collaboration is as follows:

$$\begin{aligned}
C^3_{\mathtt{US}} &= (\nu s : C_{\mathtt{US}})\ (\langle P''_{\mathtt{U}}\rangle \blacktriangleright P''_{\mathtt{U}} \ \mid\ \langle P''_{\mathtt{S}}\rangle \blacktriangleright P''_{\mathtt{S}}) \\
P''_{\mathtt{U}} &= \bar{s}?(x_{\mathit{testHD}} : \mathtt{str}).\, \mathsf{if}\ (f_{\mathit{HD}}(x_{\mathit{testHD}}))\ \mathsf{then}\ \bar{s}?(x_{\mathit{videoHD}} : \mathtt{str}).\mathbf{0}\ \mathsf{else\ roll} \\
P''_{\mathtt{S}} &= s!\langle f_{\mathit{testHD}}(v_{\mathit{req}})\rangle.\, s!\langle f_{\mathit{videoHD}}(v_{\mathit{req}})\rangle.\, \mathbf{0}
\end{aligned}$$

In the next reductions, `U` evaluates the test video and decides to revert the session execution, resulting in

$$C^4_{\mathtt{US}} = (\nu s : C_{\mathtt{US}})\ (\langle P''_{\mathtt{U}}\rangle \blacktriangleright \mathsf{roll} \ \mid\ \langle P''_{\mathtt{S}}\rangle \blacktriangleright s!\langle f_{\mathit{videoHD}}(v_{\mathit{req}})\rangle.\, \mathbf{0})$$

The execution of the roll action restores the checkpoints $P''_{\mathtt{U}}$ and $P''_{\mathtt{S}}$, that is $C^4_{\mathtt{US}} \rightsquigarrow C^3_{\mathtt{US}}$. After the rollback, `U` is not able to change the video quality as her own commit point would have permitted; in fact, it holds $C^4_{\mathtt{US}} \not\rightsquigarrow C^2_{\mathtt{US}}$.

4 Rollback safety

The operational semantics of `cherry-pi` provides a description of the functioning of the primitives for programming the checkpoint-based rollback recovery in a session-based language. However, as shown in Ex. 2, it does not guarantee high-level properties about the safe execution of the rollback. To prevent such undesired rollbacks, we propose the use of *compliance checking*, to be performed at design time before actually executing the system. This check is not done on the full system specification, but only at the level of session types.

4.1 Session types and typing

The syntax of the `cherry-pi` *session types* is defined in Fig. 7. The type $![S].T$ represents the behaviour of first outputting a value of sort S, then performing the actions prescribed by type T. The type $?[S].T$ is the dual one, where a value is received instead of sent. Type `end` represents inaction. Type $\lhd[l].T$ represents the behaviour that selects the label l and then behaves as T. Type $\rhd[l_1 : T_1, \ldots, l_n : T_n]$ describes a branching behaviour: it waits with n options, and behaves as type T_i if the i-th label is selected (external choice). Type $T_1 \oplus T_2$ behaves as either T_1 or T_2 (internal choice). Type $\mu t.T$ represents a recursive behaviour that starts by doing T and, when variable t is encountered, recurs to T again. Type $\mathtt{cmt}.T$ represents a commit action followed by the actions prescribed by type T. Finally, types `roll` and `abt` represent rollback and abort actions.

$S ::=$	**Sorts**
$\mathtt{bool} \mid \mathtt{int} \mid \mathtt{str}$	boolean, integer, string
$T ::=$	**Session types**
$![S].T \mid ?[S].T$	output, input
$\mid \triangleleft[l].T \mid \triangleright[l_1 : T_1, \ldots, l_n : T_n]$	selection, branching
$\mid T_1 \oplus T_2 \mid t \mid \mu t.T \mid \mathtt{end}$	choice, recursion, end
$\mid \mathtt{cmt}.T \mid \mathtt{roll} \mid \mathtt{abt}$	commit, roll, abort

Fig. 7. `cherry-pi` type syntax.

$$\frac{\varnothing;\varnothing \vdash P \blacktriangleright x : T}{\bar{a}(x).P \blacktriangleright \{\bar{a} : T\}}\ [\text{T-REQ}] \qquad \frac{\varnothing;\varnothing \vdash P \blacktriangleright x : T}{a(x).P \blacktriangleright \{a : T\}}\ [\text{T-ACC}]$$

$$\frac{C_1 \blacktriangleright A_1 \quad C_2 \blacktriangleright A_2}{C_1 \mid C_2 \blacktriangleright A_1 \cup A_2}\ [\text{T-PAR}]$$

Fig. 8. Typing system for `cherry-pi` collaborations.

The `cherry-pi` type system does not perform compliance checks, but it just aims at inferring the types of collaboration participants, which will be then checked together according to the compliance relation. *Typing judgements* are of the form $C \blacktriangleright A$, where A, called *type associations*, is a set of session type associations of the form $\hat{a} : T$, where $\hat{a}$ stands for either $\bar{a}$ or a. Intuitively, the judgement $C \blacktriangleright A$ indicates that from the collaboration C the type associations in A are inferred. The definition of the type system for these judgments relies on auxiliary typing judgements for processes, of the form $\Theta; \Gamma \vdash P \blacktriangleright \Delta$, where Θ, Γ and Δ, called *basis*, *sorting* and *typing* respectively, are finite partial maps from process variables to type variables, from variables to sorts, and from variables to types, respectively. Updates of basis and sorting are denoted, respectively, by $\Theta \cdot X : t$ and $\Gamma \cdot y : S$, where $X \notin dom(\Theta)$, $t \notin cod(\Theta)$ and $y \notin dom(\Gamma)$. Intuitively, the judgement $\Theta; \Gamma \vdash P \blacktriangleright \Delta$ stands for "under the environment $\Theta; \Gamma$, process P has typing Δ". In its own turn, the typing of processes relies on auxiliary (standard) judgments for expressions, of the form $\Gamma \vdash e \blacktriangleright S$. The axioms and rules defining the typing system for `cherry-pi` collaborations, processes and expressions are given in Fig. 8 and 9, and 10, respectively. The type system is defined only for initial collaborations, i.e. for terms generated by the grammar in Fig. 2. Other runtime collaborations are not considered here, as no check will be performed at runtime. We comment on salient points. Typing rules at collaboration level simply collect the type associations of session initiators in the collaboration. Rules at process level instead determine the session type corresponding to each process, by mapping each process operator to the corresponding type operator. Data and expression used in communication actions are abstracted as sorts, and a conditional choice is rendered as an internal non-deterministic choice. Typing rules for expressions are standard.

4.2 Compliance checking

To check compliance between pairs of session parties, we consider *type configurations* of the form $(T, T') : \langle \tilde{T_1} \rangle \blacktriangleright T_2 \parallel \langle \tilde{T_3} \rangle \blacktriangleright T_4$, consisting in a pair (T, T') of session types,

$$\frac{\Gamma \vdash e \blacktriangleright S \quad \Theta;\Gamma \vdash P \blacktriangleright x : T}{\Theta;\Gamma \vdash x!\langle e\rangle.P \blacktriangleright x : ![S].T}\ \text{[T-Snd]} \qquad \frac{\Theta;\Gamma \cdot y : S \vdash P \blacktriangleright x : T}{\Theta;\Gamma \vdash x?(y : S).P \blacktriangleright x : ?[S].T}\ \text{[T-Rcv]}$$

$$\Theta;\Gamma \vdash \mathbf{0} \blacktriangleright x : \texttt{end}\ \text{[T-Inact]} \qquad \frac{\Theta;\Gamma \vdash P \blacktriangleright x : T}{\Theta;\Gamma \vdash x \lhd l.P \blacktriangleright x : \lhd[l].T}\ \text{[T-Sel]}$$

$$\frac{\Theta;\Gamma \vdash P_1 \blacktriangleright x : T_1 \quad \dots \quad \Theta;\Gamma \vdash P_n \blacktriangleright x : T_n}{\Theta;\Gamma \vdash x \rhd \{l_1 : P_1, \dots, l_n : P_n\} \blacktriangleright x : \rhd[l_1 : T_1, \dots, l_n : T_n]}\ \text{[T-Br]}$$

$$\frac{\Gamma \vdash e \blacktriangleright \texttt{bool} \quad \Theta;\Gamma \vdash P_1 \blacktriangleright x : T_1 \quad \Theta;\Gamma \vdash P_2 \blacktriangleright x : T_2}{\Theta;\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \blacktriangleright x : T_1 \oplus T_2}\ \text{[T-If]}$$

$$\Theta \cdot X : t;\Gamma \vdash X \blacktriangleright t\ \text{[T-PVar]} \qquad \frac{\Theta \cdot X : t;\Gamma \vdash P \blacktriangleright T}{\Theta;\Gamma \vdash \mu X.P \blacktriangleright \mu t.T}\ \text{[T-Rec]}$$

$$\frac{\Theta;\Gamma \vdash P \blacktriangleright x : T}{\Theta;\Gamma \vdash \mathsf{commit}.P \blacktriangleright x : \texttt{cmt}.T}\ \text{[T-Cmt]}$$

$$\Theta;\Gamma \vdash \mathsf{roll} \blacktriangleright x : \texttt{roll}\ \text{[T-Rll]} \qquad \Theta;\Gamma \vdash \mathsf{abort} \blacktriangleright x : \texttt{abt}\ \text{[T-Abt]}$$

Fig. 9. Typing system for `cherry-pi` processes.

$$\Gamma \vdash \texttt{true} \blacktriangleright \texttt{bool}\ \text{[T-Bool}_{tt}\text{]} \qquad \Gamma \vdash \texttt{false} \blacktriangleright \texttt{bool}\ \text{[T-Bool}_{f\!f}\text{]}$$

$$\Gamma \cdot x : S \vdash x \blacktriangleright S\ \text{[T-Var]} \qquad \Gamma \vdash 1 \blacktriangleright \texttt{int}\ \text{[T-Int]} \qquad \Gamma \vdash \text{“}a\text{”} \blacktriangleright \texttt{str}\ \text{[T-Str]}$$

$$\frac{\Gamma \vdash e_1 \blacktriangleright \texttt{int} \quad \Gamma \vdash e_2 \blacktriangleright \texttt{int}}{\Gamma \vdash +(e_1, e_2) \blacktriangleright \texttt{int}}\ \text{[T-Sum]} \qquad \frac{\Gamma \vdash e_1 \blacktriangleright \texttt{bool} \quad \Gamma \vdash e_2 \blacktriangleright \texttt{bool}}{\Gamma \vdash \wedge(e_1, e_2) \blacktriangleright \texttt{bool}}\ \text{[T-And]}$$

Fig. 10. Typing system for `cherry-pi` expressions (excerpt of rules).

corresponding to the types of the parties at the initiation of the session, and in the parallel composition of two pairs $\langle \tilde{T_c} \rangle \blacktriangleright T$, where T is the session type of a party and $\tilde{T_c}$ is the type of the party's checkpoint. We use $\tilde{T}$ to denote either a type T, representing a checkpoint committed by the party, or $\underline{T}$, representing a checkpoint imposed by the other party of the session. The semantics of type configurations, necessary for the definition of the compliance relation, is given in Fig. 11, where label λ stands for either $![S]$, $?[S]$, $\lhd\, l$, $\rhd\, l$, τ, `cmt`, `roll`, or `abt`. We comment on the relevant rules. In case of a commit action, the checkpoints of both parties are updated, but the one of the passive party (i.e., the party that has not performed the commit action) is marked as 'imposed' (rule [TS-Cmt$_1$]). However, if the passive party did not perform any action from its current checkpoint, this checkpoint is not overwritten by the active party (rule [TS-Cmt$_2$]), as discussed in Sec. 2 (Fig. 1(c)). In case of a roll action (rule [TS-Rll$_1$]), the reduction step is performed only if the active party (i.e., the party that has performed the rollback action) has a non-imposed checkpoint; in all other situations the configuration cannot proceed with the rollback. Finally, in case of abort (rule [TS-Abt$_1$]), the configuration goes back to the initial state; this allows the type computation to proceed, in order to not affect the compliance check between the two parties.

$$!\![S].T \xrightarrow{![S]} T \ \text{[TS-Snd]} \qquad ?[S].T \xrightarrow{?[S]} T \ \text{[TS-Rcv]} \qquad \lhd[l].T \xrightarrow{\lhd\, l} T \ \text{[TS-Sel]}$$

$$\rhd[l_1 : T_1, \ldots, l_n : T_n] \xrightarrow{\rhd\, l_i} T_i \ \ (1 \leqslant i \leqslant n) \ \text{[TS-Br]} \qquad \frac{T[\mu t.T/t] \xrightarrow{\lambda} T'}{\mu t.T \xrightarrow{\lambda} T'} \ \text{[TS-rec]}$$

$$T_1 \oplus T_2 \xrightarrow{\tau} T_1 \ \text{[TS-If}_1\text{]} \qquad T_1 \oplus T_2 \xrightarrow{\tau} T_2 \ \text{[TS-If}_2\text{]}$$

$$\texttt{cmt}.T \xrightarrow{cmt} T \ \text{[TS-Cmt]} \qquad \texttt{roll} \xrightarrow{roll} \texttt{end} \ \text{[TS-Rll]} \qquad \texttt{abt} \xrightarrow{abt} \texttt{end} \ \text{[TS-Abt]}$$

$$\frac{T_1 \xrightarrow{\tau} T_1'}{(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2} \ \text{[TS-Tau]}$$

$$\frac{T_1 \xrightarrow{![S]} T_1' \qquad T_2 \xrightarrow{?[S]} T_2'}{(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2'} \ \text{[TS-Com]}$$

$$\frac{T_1 \xrightarrow{\lhd\, l} T_1' \qquad T_2 \xrightarrow{\rhd\, l} T_2'}{(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2'} \ \text{[TS-Lab]}$$

$$\frac{T_1 \xrightarrow{cmt} T_1' \qquad \tilde{U}_2 \neq T_2}{(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle T_1' \rangle \blacktriangleright T_1' \parallel \langle \underline{T_2} \rangle \blacktriangleright T_2} \ \text{[TS-Cmt}_1\text{]}$$

$$\frac{T_1 \xrightarrow{cmt} T_1' \qquad \tilde{U}_2 = T_2}{(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle T_1' \rangle \blacktriangleright T_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2} \ \text{[TS-Cmt}_2\text{]}$$

$$\frac{T_1 \xrightarrow{roll} T_1'}{(T,T') : \langle U_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle U_1 \rangle \blacktriangleright U_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright U_2} \ \text{[TS-Rll}_1\text{]}$$

$$\frac{T_1 \xrightarrow{abt} T_1'}{(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto (T,T') : \langle T \rangle \blacktriangleright T \parallel \langle T' \rangle \blacktriangleright T'} \ \text{[TS-Abt}_1\text{]}$$

Fig. 11. Semantics of types and type configurations (symmetric rules for configurations are omitted).

We define the compliance relation, inspired by the relation in [2], and prove its decidability (the proof is reported in Appendix A).

Definition 1 (Compliance). *Two types T_1 and T_2 are* compliant, *written $T_1 \dashv\!\vert T_2$, if $(T_1, T_2) : \langle \underline{T_1} \rangle \blacktriangleright T_1 \dashv\!\vert \langle \underline{T_2} \rangle \blacktriangleright T_2$. Relation $\dashv\!\vert$ on configurations is defined as follows: $(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \dashv\!\vert \langle \tilde{U}_2 \rangle \blacktriangleright T_2$ holds if for any U_1', T_1', U_2', T_2' such that $(T,T') : \langle \tilde{U}_1 \rangle \blacktriangleright T_1 \parallel \langle \tilde{U}_2 \rangle \blacktriangleright T_2 \longmapsto^* (T,T') : \langle \tilde{U}_1' \rangle \blacktriangleright T_1' \parallel \langle \tilde{U}_2' \rangle \blacktriangleright T_2' \not\longmapsto$ we have that $T_1' = T_2' = \texttt{end}$.*

Theorem 1. *Let T_1 and T_2 be two session types, checking if $T_1 \dashv\!\vert T_2$ holds is decidable.*

The above compliance relation is used to define the notion of *rollback safety*.

Definition 2 (Rollback safety). *Let C be an initial collaboration, then C is* rollback safe *(shortened* r-safe*) if $C \;\blacktriangleright\; A$ and for all pairs $\bar{a} : T_1$ and $a : T_2$ in A we have $T_1 \dashv\Vert T_2$.*

Example 3. Let us consider again the VOD example. As expected, the first `cherry-pi` collaboration defined in Ex. 1, corresponding to the scenario described in Fig. 1(b), is not rollback safe, because the types of the two parties are not compliant. Indeed, the session types T_{U} and T_{S} associated by the type system to the user and the service processes, respectively, are as follows:

$$\begin{aligned} T_{\mathrm{U}} \;=\; & ![\mathtt{str}].\,?[\mathtt{int}].\,\mathtt{cmt}.\,?[\mathtt{str}].\,(\,\lhd[l_{\mathit{HD}}].\,?[\mathtt{str}].\,(\,?[\mathtt{str}].\,\mathtt{end} \,\oplus\, \mathtt{roll}\,) \\ & \qquad\qquad\qquad\qquad\qquad \oplus\, \lhd\,[l_{\mathit{SD}}].\,?[\mathtt{str}].\,(\,?[\mathtt{str}].\,\mathtt{end} \,\oplus\, \mathtt{abt}\,)\,) \end{aligned}$$

$$\begin{aligned} T_{\mathrm{S}} \;=\; ?[\mathtt{str}].\,![\mathtt{int}].\,![\mathtt{str}].\; \rhd\,[\, & l_{\mathit{HD}} \,:\, \mathtt{cmt}.\,![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}\;, \\ & l_{\mathit{SD}} \,:\, \mathtt{cmt}.\,![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}] \end{aligned}$$

Thus, the resulting initial configuration is $(T_{\mathrm{U}}, T_{\mathrm{S}}) \,:\, \langle T_{\mathrm{U}}\rangle \blacktriangleright T_{\mathrm{U}} \;\Vert\; \langle T_{\mathrm{S}}\rangle \blacktriangleright T_{\mathrm{S}}$, which can evolve to the configuration $(T_{\mathrm{U}}, T_{\mathrm{S}}) \,:\, \langle \underline{T}\rangle \blacktriangleright \mathtt{roll} \;\Vert\; \langle U\rangle\blacktriangleright ![\mathtt{str}].\mathtt{end}$, with $T = ?[\mathtt{str}].\,(?[\mathtt{str}].\,\mathtt{end} \,\oplus\, \mathtt{roll})$ and $U = ![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}$. This configuration cannot evolve and is not in a final state (in fact, types `roll` and $![\mathtt{str}].\mathtt{end}$ are different from `end`), meaning that T_{U} and T_{S} are not compliant.

In the scenario described in Fig. 1(c), instead, the type of the server process is as follows:

$$\begin{aligned} T'_{\mathrm{S}} \;=\; ?[\mathtt{str}].\,![\mathtt{int}].\,\mathtt{cmt}.\,![\mathtt{str}].\; \rhd\,[\, & l_{\mathit{HD}} \,:![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}\;, \\ & l_{\mathit{SD}} \,:![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}] \end{aligned}$$

and we have $T_{\mathrm{U}} \dashv\Vert T'_{\mathrm{S}}$.

Finally, the types of the processes depicted in Fig. 1(d) are:

$$T'_{\mathrm{U}} \;=\; ![\mathtt{str}].\,?[\mathtt{int}].\,?[\mathtt{str}].\,\mathtt{cmt}.\,(\,\lhd[l_{\mathit{HD}}].\,\ldots\,\oplus\,\lhd[l_{\mathit{SD}}].\,\ldots\,)$$

$$\begin{aligned} T''_{\mathrm{S}} \;=\; ?[\mathtt{str}].\,![\mathtt{int}].\,![\mathtt{str}].\,\mathtt{cmt}.\; \rhd\,[\, & l_{\mathit{HD}} \,:![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}\;, \\ & l_{\mathit{SD}} \,:![\mathtt{str}].\,![\mathtt{str}].\,\mathtt{end}] \end{aligned}$$

and we have $T'_{\mathrm{U}} \not\dashv\Vert\; T''_{\mathrm{S}}$. Indeed, the corresponding initial configuration can evolve to the configuration $(T'_{\mathrm{U}}, T''_{\mathrm{S}}) : \langle\underline{\lhd[l_{\mathit{HD}}].\,\ldots}\rangle \blacktriangleright \mathtt{roll} \;\Vert\; \langle\rhd[l_{\mathit{HD}} \,:\, \ldots, l_{\mathit{SD}} \,:\, \ldots]\rangle\blacktriangleright ![\mathtt{str}].\mathtt{end}$, which again is not in a final state.

4.3 MAUDE implementation

To practically enable the compliance checking at the basis of our notion of rollback safety, we have implemented the semantics of type configurations in Fig. 11 within the MAUDE framework [7]. MAUDE provides an instantiation of rewriting logic [24] and it has been used to implement the semantics of a variety of formal languages [25].

The syntax of `cherry-pi` types and type configurations is specified by defining algebraic data types, while transitions and reductions are rendered as rewrites and, hence, inference rules are given in terms of (conditional) rewrite rules. Since MAUDE specifications are executable, we have obtained in this way an interpreter for `cherry-pi` type configurations, which permits to explore the reductions arising from the initial configuration of two given session types.

Our implementation consists of two MAUDE modules:

- CHERRY-TYPES-SYNTAX provides the definition of the sorts that characterise the syntax of `cherry-pi` types, such as session types, selection/branching labels, type variables and type configurations. In particular, basic terms of session types are rendered as constant *operations* on the sort `Type`; e.g., the roll type is defined as

  ```
  op roll : -> Type .
  ```

  The other syntactic operators are instead defined as operations with one or more input arguments; e.g., the output type takes as input a `Sort` (i.e., `bool`, `int` or `str`) and a continuation type:

  ```
  op ![_]._ : Sort Type -> Type [frozen prec 25] .
  ```

  To prevent undesired rewrites inside operator arguments, following the approach in [35], we declared these operations as `frozen`. The `prec` attribute has been used to define the precedence among operators.
- CHERRY-TYPES-SEMANTICS provides *rewrite rules*, and additional operators and equations, to define the `cherry-pi` type semantics. For example, the operational rule [TS-SND] of the output type, i.e. $![S].T \xrightarrow{![S]} T$, is rendered as the following rewrite rule:

  ```
  rl [TS-Snd] : ![S].T => {![S]}T .
  ```

  The correspondence between the operational rule and the rewrite one is one-to-one; the only peculiarity is the fact that, since rewrites have no labels, we have made the transition label part of the resulting term. Reduction rules for type configurations are instead rendered in terms of (conditional) rewrite rules with rewrites in their conditions. For example, the [TS-COM] rule is rendered as:

  ```
  crl [TS-Com] :
    init(T,T') CT1 > T1 || CT2 > T2  =>  init(T,T') CT1 > T1' || CT2 > T2'
    if T1 => {![S]}T1' /\ T2 => {?[S]}T2' .
  ```

The compliance check between two session types can be then conveniently realised on top of the implementation described above by resorting to the MAUDE command `search`. This permits indeed to explore the state space of configurations reachable from an initial configuration. Specifically, the compliance check between types `T1` and `T2` is rendered as follows:

```
search
  init(T1,T2) ckp(T1) > T1 || ckp(T2) > T2
  =>!
  init(T:Type,T':Type) CT1:CkpType > T1':Type || CT2:CkpType > T2':Type
such that T1' =/= end or T2' =/= end .
```

This command searches for all terminal states (`=>!`), i.e. states that cannot be rewritten any more (see $\not\longmapsto$ in Def. 1), and checks if at least one of the two session types in the corresponding configurations (T1′ and T2′) is different from the end type. Thus, if this search has no solution, T1 and T2 are compliant; otherwise, they are not compliant and a violating configuration is returned.

Example 4. Let us consider the cherry-pi types defined in Ex. 3 for the scenario described in Fig. 1(b). In our MAUDE implementation of the type syntax, the session types T_{U} and T_{S}, and the corresponding initial type configuration, are rendered as follows:

```
eq Tuser = ![str]. ?[int]. cmt. ?[str].
          ((sel['hd]. ?[str]. ((?[str]. end) (+) roll))
           (+) (sel['sd]. ?[str]. ((?[str]. end) (+) abt))) .

eq Tservice = ?[str]. ![int]. ![str].
              brn[brnEl('hd, cmt. ![str]. ![str]. end);
                  brnEl('sd, cmt. ![str]. ![str]. end)] .

eq InitConfig = init(Tuser,Tservice)
                ckp(Tuser) > Tuser || ckp(Tservice) > Tservice .
```

where (+) represents the internal choice operator, sel the selection operator, brn the branching operator, brnEl an option offered in a branching, and ckp a non-imposed checkpoint. The compliance between the two session types can be then checked by first loading the two modules of our MAUDE implementation [5]:

```
./maude.darwin64 cherry-pi_type_semantics.maude
```

and then executing the following command:

```
search InitConfig
       =>!
       init(T:Type,T':Type) CT1:CkpType > T1:Type || CT2:CkpType > T2:Type
such that T1 =/= end or T2 =/= end .
```

This search command returns the following solution:

```
CT1 --> ickp(?[str]. ((?[str]. end)(+)roll))
T1 --> roll
CT2 --> ckp(![str]. ![str]. end)
T2 --> ![str]. end
```

As already explained in Ex. 3, the two types are not compliant. Indeed, the configuration above is final, because T1 cannot perform the rollback, since its checkpoint is imposed (ickp), and T2 cannot perform the communication action, since its partner T1 is not ready to receive. Moreover, roll and ![str].end are clearly different from end.

The scenario in Fig. 1(c) is rendered by means of the following implementation of the service type:

```
eq Tservice' = ?[str]. ![int]. cmt. ![str].
               brn[brnEl('hd, ![str]. ![str]. end);
                   brnEl('sd, ![str]. ![str]. end)] .
```

In this case, as expected, the search command returns:

```
No solution.
```

[5] We have used the version 2.7.1 of MAUDE for Mac OS X.

meaning that the types are compliant.

Finally, the `search` command applied to the type configuration related to the scenario depicted in Fig. 1(d) returns a solution, meaning that the user and service types are not compliant.

5 Properties of `cherry-pi`

We present in this section the results regarding the properties enjoyed by `cherry-pi`. Their proofs are reported in Appendix A. The statement of some properties exploits labelled transitions that permit to easily distinguish the execution of commit and rollback actions from the other ones. To this aim, rules [F-CMT], [B-RLL] and [B-ABT] can be slightly modified by labelling the transition in their conclusions by cmt, $roll$ and abt, respectively, and rules [F/B-PAR], [F/B-RES] and [F/B-STR] change accordingly in order to propagate the labelled transitions.

5.1 Rollback properties

We show some properties related to the reversible behaviour of `cherry-pi` and in particular related to the interplay between rollback and commit primitives.

The following theorem states that any reachable collaboration is also a *forward only* reachable collaboration. This means that all the states a collaboration reaches via backward reductions are states that we can reach from the initial configuration with just forward reductions. This assures us that if the system goes back it will reach previous visited states.

Theorem 2. *Let C_0 be an initial collaboration. If $C_0 \rightarrowtail^* C_1$ then $C_0 \twoheadrightarrow^* C_1$.*

We now show a variant of the so-called Loop Lemma [10]. In a fully reversible calculus this lemma states that each computational step, either forward or backward, can be undone. Since reversibility in `cherry-pi` is controlled, we have to state that if a reversible step is possible (e.g., a rollback is *enabled*) then the effects of the rollback can be undone. Formally we have:

Lemma 1 (Safe rollback). *Let C_1 and C_2 be reachable collaborations. If $C_1 \rightsquigarrow C_2$ then $C_2 \twoheadrightarrow^* C_1$.*

The following result states that a rollback always brings the system to the last taken checkpoint.

Lemma 2 (Determinism). *Let C be a reachable collaboration. If $C \overset{roll}{\rightsquigarrow} C'$ and $C \overset{roll}{\rightsquigarrow} C''$ then $C' \equiv C''$.*

The last rollback property states that a `cherry-pi` collaboration cannot go back to a state prior to the execution of a commit action, that is commits have a persistent effect.

Theorem 3 (Undoability). *Let C be a reachable collaboration. If $C \overset{cmt}{\twoheadrightarrow} C'$ then there exists no C'' such that $C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$ and $C'' \twoheadrightarrow^+ C$.*

$$\frac{P_1 \xrightarrow{k!\langle v\rangle} P_1' \qquad \neg P_2 \Downarrow_{k?}}{\langle \tilde{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \texttt{com\_error}}\ \text{[E-Com1]} \qquad \frac{P_1 \xrightarrow{k?(x)} P_1' \qquad \neg P_2 \Downarrow_{k!}}{\langle \tilde{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \texttt{com\_error}}\ \text{[E-Com2]}$$

$$\frac{P_1 \xrightarrow{k\lhd l} P_1' \qquad \neg P_2 \Downarrow_{k\rhd l}}{\langle \tilde{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \texttt{com\_error}}\ \text{[E-Lab}_1\text{]} \qquad \frac{P_1 \xrightarrow{k\rhd l} P_1' \qquad \neg P_2 \Downarrow_{k\lhd l}}{\langle \tilde{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \texttt{com\_error}}\ \text{[E-Lab}_2\text{]}$$

$$\frac{P_1 \xrightarrow{cmt} P_1' \qquad \tilde{Q_2} \neq P_2}{\langle \tilde{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \langle P_1'\rangle \blacktriangleright P_1' \mid \langle \underline{P_2}\rangle \blacktriangleright P_2}\ \text{[E-Cmt}_1\text{]}$$

$$\frac{P_1 \xrightarrow{cmt} P_1' \qquad \tilde{Q_2} = P_2}{\langle \tilde{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \langle P_1'\rangle \blacktriangleright P_1' \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2}\ \text{[E-Cmt}_2\text{]}$$

$$\frac{P_1 \xrightarrow{roll} P_1'}{\langle Q_1\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \rightsquigarrow \langle Q_1\rangle \blacktriangleright Q_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright Q_2}\ \text{[E-Rll}_1\text{]}$$

$$\frac{P_1 \xrightarrow{roll} P_1'}{\langle \underline{Q_1}\rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2}\rangle \blacktriangleright P_2 \twoheadrightarrow \texttt{roll\_error}}\ \text{[E-Rll}_2\text{]}$$

Fig. 12. `cherry-pi` semantics: error reductions.

5.2 Soundness properties

The second group of properties concerns soundness guarantees. The definition of these properties requires to formally characterise the errors that may occur on the execution of an unsound collaboration. We rely on error reduction (as in [6]) rather than on the usual static characterisation of errors (as, e.g., in [38]), since rollback errors cannot be easily detected statically. In particular, we extend the syntax of `cherry-pi` collaborations with the `roll_error` and `com_error` terms, denoting respectively collaborations in rollback and communication error states:

$$\begin{aligned} C ::= {} & \bar{a}(x).P \mid a(x).P \mid C_1|C_2 \mid (\nu s : C_1)\, C_2 \\ \mid {} & \langle \tilde{P_1}\rangle \blacktriangleright P_2 \mid \texttt{roll\_error} \mid \texttt{com\_error} \end{aligned}$$

where $\tilde{P}$ denotes either a checkpoint P committed by the party or a checkpoint $\underline{P}$ imposed by the other party of the session. The semantics of `cherry-pi` is extended as well by the (error reduction) rules in Fig. 12, where [E-Cmt$_1$] and [E-Cmt$_2$] replace [F-Cmt], and [E-Rll$_1$] replaces [B-Rll], and ˜ is used in the checkpoints of the other rules. The error semantics does not affect the normal behaviour of `cherry-pi` specifications, but it is crucial for stating our soundness theorems. Its definition is based on the notion of *barb* predicate: $P \Downarrow_\mu$ holds if there exists P' such that $P \Rightarrow P'$ and P' can perform an action μ, where μ stands for $k?$, $k!$,$k \lhd l$ or $k \rhd l$ (i.e., input, output, select and branching action along session channel k); $\Rightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$. The meaning of the error semantics rules is as follows. A *communication error* takes place in a collaboration when a session participant is willing to perform an output but the other participant is not ready to perform the corresponding input (rule [E-Com$_1$]) or viceversa (rule [E-Com$_2$]), or one participant is willing to perform a selection but the corresponding branching is not available on the other side (rule [E-LAB1]) or viceversa (rule[E-LAB2]). Instead, a *rollback error* takes place in a collaboration when a participant is willing to perform a rollback action but its checkpoint has been imposed by the other participant ([E-RLL$_2$]). To enable this error check, the rules for commit and rollback have been modified to keep track of imposed overwriting of checkpoints. This information is not relevant for the runtime execution of processes, but it is necessary for characterising the rollback errors that our type-based approach prevents.

Besides defining the error semantics, we also need to define erroneous collaborations, based on the following notion of context: $\mathbb{C} \ ::= \ [\cdot] \ \mid \ \mathbb{C} | C \ \mid \ (\nu s : C)\, \mathbb{C}$.

Definition 3 (Erroneous collaborations). *A collaboration C is* communication (resp. rollback) erroneous *if $C = \mathbb{C}[\mathtt{com\_error}]$ (resp. $C = \mathbb{C}[\mathtt{roll\_error}]$).*

The key soundness results follow: a rollback safe collaboration never reduces to either a rollback erroneous collaboration (Theorem 4) or a communication erroneous collaboration (Theorem 5).

Theorem 4 (Rollback soundness). *If C is a r-safe collaboration, then we have that $C \not\rightarrowtail^* \mathbb{C}[\mathtt{roll\_error}]$.*

Theorem 5 (Communcation soundness). *If C is a r-safe collaboration, then we have that $C \not\rightarrowtail^* \mathbb{C}[\mathtt{com\_error}]$.*

We conclude with a property concerning the progress of `cherry-pi` sessions: given a rollback safe collaboration that can initiate a session, each collaboration reachable from it either is able to progress on the session with a forward/backward reduction step or has correctly reached the end of the session. This result straightforwardly follows from Theorems 4 and 5, and from the fact that we consider binary sessions without delegation and subordinate sessions.

Theorem 6 (Session progress). *Let $C = (\bar{a}(x_1).P_1 \mid a(x_2).P_2)$ be a r-safe collaboration. If $C \rightarrowtail^* C'$ then either $C' \rightarrowtail C''$ for some C'' or $C' \equiv (\nu s : C)(\langle \tilde{Q}_1 \rangle \blacktriangleright \mathbf{0} \mid \langle \tilde{Q}_2 \rangle \blacktriangleright \mathbf{0})$ for some $\tilde{Q}_1$ and $\tilde{Q}_2$.*

6 `cherry-pi` at work on a speculative parallelism scenario

To shed light on the practical effectiveness of `cherry-pi` and the related notion of rollback safety, we consider in this section a simple, yet realistic, scenario concerning a form of speculative execution borrowed from [31]. In this scenario, *value speculation* is used as a mechanism for increasing parallelism, hence system performance, by predicting values of data dependencies between tasks. Whenever a value prediction is incorrect, corrective actions must be taken in order to re-execute the data consumer code with the correct data value. In this regard, as shown in [14] for a shared-memory setting, reversible execution can permit to relieve programmers from the burden of properly undoing the actions subsequent to an incorrect prediction. Here, we tailor the scenario to the channel-based communication model of session-based programming, and show how

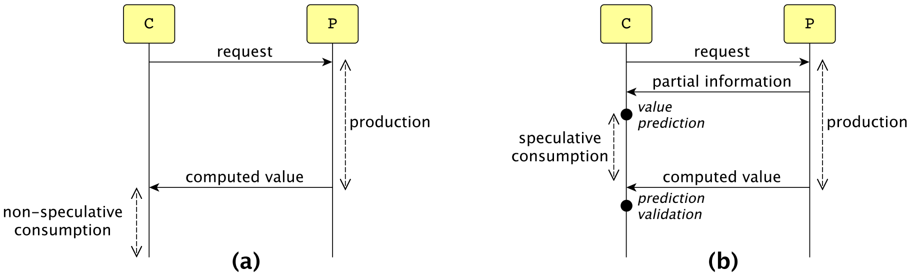

Fig. 13. Producer-consumer scenario with non-speculative (a) and speculative (b) consumers.

our rollback safety checking supports programmers in identifying erroneous rollback recovery settings.

In the producer-consumer scenario depicted in Fig. 13(a) the session participant `P` produces a value and the participant `C` consumes it. The data dependence between `P` and `C` serialises their executions, thus forcing `C` to wait for the completion of the value production that requires a fairly long time. In the scenario in Fig. 13(b), instead, `C` enacts a speculative behaviour, as it predicts ahead of time the value computed by `P` from a partial information. By using the predicted value, `C` can execute speculatively and concurrently with `P`. When `P` completes the production, `C` validates the prediction by comparing the actual value computed by `P` and the predicted one; if the prediction is precise, we gain performance because the execution of `C` and `P` overlapped in time, otherwise rollback is used to move `C` and `P` back to a state that precedes the speculative behaviour, in order to re-execute `C` using the correct value. The behaviours of `C` and `P` can be recursively defined in order to repeat the overall execution once a value is correctly consumed.

The scenario informally described above is rendered in `cherry-pi` as $\overline{start}(x).P_{\mathtt{C}} \mid start(y).P_{\mathtt{P}}$, where the consumer and producer processes are:

$$\begin{aligned} P_{\mathtt{C}} = \mu X.\, x!\langle f_{\mathit{req}}()\rangle.\, x \triangleright \{\, & l_{\mathit{spec}} \;:\; x?(x_{\mathit{partial}} : \mathtt{str}).\, x?(x_{\mathit{final}} : \mathtt{str}).\\ & \qquad\qquad \text{if } (f_{\mathit{compare}}(x_{\mathit{partial}}, x_{\mathit{final}})) \text{ then roll else commit}.\, X \,,\\ & l_{\mathit{nonSpec}} \;:\; x?(x_{\mathit{computed}} : \mathtt{str}).\, \text{commit}.\, X \,\} \end{aligned}$$

$$\begin{aligned} P_{\mathtt{P}} = {} & \mu Y.\, y?(y_{\mathit{req}} : \mathtt{str}).\\ & \quad \text{if } (f_{\mathit{eval}}(y_{\mathit{req}})) \text{ then } y \triangleleft l_{\mathit{spec}}.\, y!\langle f_{\mathit{partial}}(y_{\mathit{req}})\rangle.\, y!\langle f_{\mathit{final}}(y_{\mathit{req}})\rangle.\, Y\\ & \quad \text{else } y \triangleleft l_{\mathit{nonSpec}}.\, y!\langle f_{\mathit{compute}}(y_{\mathit{req}})\rangle.\, Y \end{aligned}$$

The producer evaluates each consumer's request in order to establish whether to provide directly the produced value or the partial information for the prediction. In the former case the consumer commits the session and both participants restart, while in the latter one the consumer commits or rolls back depending on the result of the comparison between the predicted value and the produced one.

According to our compliance check, the above collaboration is rollback safe. In case of incorrect prediction the session execution is moved back to the last checkpoint, corresponding to the successfully consumption of the previous requested value. When the producer receives again the same request, it can immediately send the already produced

$C ::= \cdots \mid \bar{a}[\mathtt{p}](x).P \mid a[\mathtt{p}](x).P$ **Collaborations**

$P ::= \cdots \mid r[\mathtt{p}]!\langle e\rangle.P \mid r[\mathtt{p}]?(y : S).P \mid r[\mathtt{p}] \lhd l.P \mid r[\mathtt{p}] \rhd \{l_1 : P_1, \ldots, l_n : P_n\}$ **Processes**

Fig. 14. Multiparty `cherry-pi` runtime syntax (the omitted parts are as in Fig. 3).

value. In case the first prediction is wrong, and hence no commit action is performed by the consumer yet, according to the `cherry-pi` semantics the checkpoint corresponds to the beginning of the session.

Let us consider instead the case of a producer that commits each time a value production is completed, which could apparently seem a reasonable behaviour from the producer side. The resulting collaboration, however, is not rollback safe: while the commit action in the non-speculative case does not affect the compliance between the two session participants, the other commit action overwrites the checkpoint set by the consumer, making it impossible to re-execute the consumer with the correct value. This situation, undesirable for the consumer, is detected by our compliance check.

7 Extension to multiparty sessions

We discuss in this section how to extend `cherry-pi`, its type discipline, the compliance checking and the related results, to multiparty sessions [17].

The base sets for the multiparty syntax of `cherry-pi` 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* r now range over session endpoints $s[\mathtt{p}]$ or variables x. The runtime syntax of multiparty `cherry-pi` is defined by the grammar in Fig. 14, where expressions e are defined as in the binary case (with values that extends to multiparty session endpoints). Primitive $\bar{a}[\mathtt{p}](x).P$ initiates a new session through identifier a on the other multiple participants, each one of the form $a[\mathtt{q}](x).P_{\mathtt{q}}$ where $1 \leqslant \mathtt{q} \leqslant \mathtt{p} - 1$. Variable x will be substituted with the session endpoint used for the interactions inside the established session. Primitive $r[\mathtt{p}]!\langle e\rangle.P$ denotes the intention of sending a value to role $\mathtt{p}$; similarly, process $r[\mathtt{p}]?(y : S).P$ denotes the intention of receiving a value from role $\mathtt{p}$. Selection and branching are extended 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 standard, while the forward and backward reduction relations are given by the rules in Fig. 15. We comment on salient points. Rule [M-F-CON] synchronously initiates a session by requiring all session endpoints be present for a forward reduction, where each role $\mathtt{p}$ creates a session endpoint $s[\mathtt{p}]$ on a fresh session channel s. The participant with the maximum role is responsible for requesting a session initiation. Rule [M-F-COM] defines how a party with role $\mathtt{p}$ synchronously sends a value to the receiving party with role $\mathtt{q}$. Rules [M-F-CMT], [M-B-RLL] and [M-B-ABT] are similar to those of the binary case, and affect all participants within the considered session.

The syntax of session types extends to multiparty as shown in Fig. 16. The session types for output ($[\mathtt{p}][\mathtt{q}]!\langle S\rangle.T$) and input ($[\mathtt{p}][\mathtt{q}]?(S).T$) are extended with information about the interacting roles; selection and branching types are similarly extended. In the

$$s[\mathtt{p}][\mathtt{q}]!\langle e\rangle.P \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v\rangle} P \ \ (e\downarrow v)\ \textsc{[M-P-Snd]} \qquad s[\mathtt{p}][\mathtt{q}]?(y\,{:}\,S).P \xrightarrow{s[\mathtt{p}][\mathtt{q}]?(y)} P\ \textsc{[M-P-Rcv]}$$

$$\begin{array}{ll} \bar{a}[n](x).P_n \ \mid\ \prod_{i\in I} a[i](x).P_i \ \ \twoheadrightarrow & \\ \quad (\nu s\,{:}\,(\bar{a}[n](x).P_n \ \mid\ \prod_{i\in I} a[i](x).P_i)) & I=\{1,..,n-1\} \quad \textsc{[M-F-Con]} \\ \quad (\langle P_n[s[n]/x]\rangle \blacktriangleright P_n[s[n]/x] \mid \prod_{i\in I}\langle P_i[s[i]/x]\rangle \blacktriangleright P_i[s[i]/x]) & \end{array}$$

$$\frac{P_1 \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v\rangle} P_1' \qquad P_2 \xrightarrow{s[\mathtt{q}][\mathtt{p}]?(x)} P_2'}{\langle Q_1\rangle \blacktriangleright P_1 \mid \langle Q_2\rangle \blacktriangleright P_2 \ \twoheadrightarrow\ \langle Q_1\rangle \blacktriangleright P_1' \mid \langle Q_2\rangle \blacktriangleright P_2'[v/x]}\ \textsc{[M-F-Com]}$$

$$\frac{P \xrightarrow{cmt} P'}{(\nu s : C)(\langle Q\rangle \blacktriangleright P \ \mid\ \prod_{i\in I}\langle Q_i\rangle \blacktriangleright P_i) \ \twoheadrightarrow\ (\nu s : C)(\langle P'\rangle \blacktriangleright P' \ \mid\ \prod_{i\in I}\langle P_i\rangle \blacktriangleright P_i)}\ \textsc{[M-F-Cmt]}$$

$$\frac{P \xrightarrow{roll} P'}{(\nu s : C)(\langle Q\rangle \blacktriangleright P \ \mid\ \prod_{i\in I}\langle Q_i\rangle \blacktriangleright P_i) \ \rightsquigarrow\ (\nu s : C)(\langle Q\rangle \blacktriangleright Q \ \mid\ \prod_{i\in I}\langle Q_i\rangle \blacktriangleright Q_i)}\ \textsc{[M-B-Rll]}$$

$$\frac{P \xrightarrow{abt} P'}{(\nu s : C)(\langle Q\rangle \blacktriangleright P \ \mid\ \prod_{i\in I}\langle Q_i\rangle \blacktriangleright P_i) \ \rightsquigarrow\ C}\ \textsc{[M-B-Abt]}$$

Fig. 15. Multiparty `cherry-pi` semantics: forward and backward reductions (excerpt of rules).

$$T ::= \ldots \mid [\mathtt{p}][\mathtt{q}]!\langle S\rangle.T \ \mid\ [\mathtt{p}][\mathtt{q}]?(S).T \ \mid\ [\mathtt{p}][\mathtt{q}] \lhd [l].T \ \mid\ [\mathtt{p}][\mathtt{q}] \rhd [l_1\,{:}\,T_1,\ldots,l_n\,{:}\,T_n] \quad \textbf{Types}$$

Fig. 16. Multiparty `cherry-pi` type syntax (the omitted parts are as in Fig. 7).

$$\frac{\varnothing;\varnothing \ \vdash\ P \ \blacktriangleright\ x : T}{\bar{a}[\mathtt{p}](x).P \ \blacktriangleright\ \{\bar{a}[\mathtt{p}] : T\}}\ \textsc{[M-T-Req]} \qquad \frac{\varnothing;\varnothing \ \vdash\ P \ \blacktriangleright\ x : T}{a[\mathtt{p}](x).P \ \blacktriangleright\ \{a[\mathtt{p}] : T\}}\ \textsc{[M-T-Acc]}$$

$$\frac{\mathit{\Gamma} \ \vdash\ e \ \blacktriangleright\ S \qquad \mathit{\Theta};\mathit{\Gamma} \ \vdash\ P \ \blacktriangleright\ x : T}{\mathit{\Theta};\mathit{\Gamma} \ \vdash\ x[\mathtt{p}]!\langle e\rangle.P \ \blacktriangleright\ x : [\_][\mathtt{p}]!\langle S\rangle.T}\ \textsc{[M-T-Snd]}$$

$$\frac{\mathit{\Theta};\mathit{\Gamma}\cdot y : S \ \vdash\ P \ \blacktriangleright\ x : T}{\mathit{\Theta};\mathit{\Gamma} \ \vdash\ x[\mathtt{p}]?(y : S).P \ \blacktriangleright\ x : [\_][\mathtt{p}]?(S).T}\ \textsc{[M-T-Rcv]}$$

$$\frac{\mathit{\Theta};\mathit{\Gamma} \ \vdash\ P \ \blacktriangleright\ x : T}{\mathit{\Theta};\mathit{\Gamma} \ \vdash\ x[\mathtt{p}] \lhd l.P \ \blacktriangleright\ x : [\_][\mathtt{p}] \lhd [l].T}\ \textsc{[M-T-Sel]}$$

$$\frac{\mathit{\Theta};\mathit{\Gamma} \ \vdash\ P_1 \ \blacktriangleright\ x : T_1 \qquad \ldots \qquad \mathit{\Theta};\mathit{\Gamma} \ \vdash\ P_n \ \blacktriangleright\ x : T_n}{\mathit{\Theta};\mathit{\Gamma} \ \vdash\ x[\mathtt{p}] \rhd \{l_1\,{:}\,P_1,\ldots,l_n\,{:}\,P_n\} \ \blacktriangleright\ x : [\_][\mathtt{p}] \rhd [l_1\,{:}\,T_1,\ldots,l_n\,{:}\,T_n]}\ \textsc{[M-T-Br]}$$

Fig. 17. Multiparty `cherry-pi` typing system (the omitted rules as in Fig. 8, 9 and 10).

type inference, when one of such role is unknown, it is used a placeholder $\_$ to be filled with a given role; $T \cdot \mathtt{p}$ denotes the type obtained from T by filling all its placeholders with the role $\mathtt{p}$. The `cherry-pi` type system extends accordingly, as shown in Fig. 17.

$$[\mathtt{p}][\mathtt{q}]!\langle S\rangle.T \xrightarrow{[\mathtt{p}][\mathtt{q}]!\langle S\rangle} T \ \text{[M-TS-SND]} \qquad\qquad [\mathtt{p}][\mathtt{q}]?(S).T \xrightarrow{[\mathtt{p}][\mathtt{q}]?(S)} T \ \text{[M-TS-RCV]}$$

$$\frac{T_i \xrightarrow{[\mathtt{p}][\mathtt{q}]!\langle S\rangle} T_i' \qquad T_j \xrightarrow{[\mathtt{q}][\mathtt{p}]?(S)} T_j'}{\begin{array}{l}(T^I) : \langle \tilde{U}_i\rangle \blacktriangleright T_i \parallel \langle \tilde{U}_j\rangle \blacktriangleright T_j \parallel \prod_{h\in I-\{i,j\}}\langle \tilde{U}_h\rangle \blacktriangleright T_h \\ \quad\longmapsto\ (T^I) : \langle \tilde{U}_i\rangle \blacktriangleright T_i' \parallel \langle \tilde{U}_j\rangle \blacktriangleright T_j' \parallel \prod_{h\in I-\{i,j\}}\langle \tilde{U}_h\rangle \blacktriangleright T_h\end{array}} \ \text{[M-TS-COM]}$$

$$\frac{T_i \xrightarrow{cmt} T_i'}{\begin{array}{l}(T^I) : \langle \tilde{U}_i\rangle \blacktriangleright T_i \parallel \prod_{h\in I-\{i\}}\langle \tilde{U}_h\rangle \blacktriangleright T_h \\ \quad\longmapsto\ (T^I) : \langle T_i'\rangle \blacktriangleright T_i' \parallel \prod_{h\in I-\{i\}}\langle \tilde{U}_h'\rangle \blacktriangleright T_h\end{array}} \quad \tilde{U}_h' = \begin{cases}\tilde{U}_h \text{ if } \tilde{U}_h = T_h \\ \underline{T_h} \text{ otherwise}\end{cases} \ \text{[M-TS-CMT]}$$

$$\frac{T_i \xrightarrow{roll} T_i'}{(T^I) : \langle U_i\rangle \blacktriangleright T_i \parallel \prod_{h\in I-\{i\}}\langle \tilde{U}_h\rangle \blacktriangleright T_h \ \longmapsto\ (T^I) : \langle U_i\rangle \blacktriangleright U_i \parallel \prod_{h\in I-\{i\}}\langle \tilde{U}_h\rangle \blacktriangleright U_h} \ \text{[M-TS-RLL]}$$

$$\frac{T_i \xrightarrow{abt} T_i'}{(T^I) : \langle \tilde{U}_i\rangle \blacktriangleright T_i \parallel \prod_{h\in I-\{i\}}\langle \tilde{U}_h\rangle \blacktriangleright T_h \ \longmapsto\ (T^I) : \prod_{k\in I}\langle T^k\rangle \blacktriangleright T^k} \ \text{[M-TS-ABT]}$$

Fig. 18. Multiparty `cherry-pi` type semantics (excerpt of rules, where $I = \{1, \ldots, n\}$, $i, j \in I$, and T^I denotes $T^1, \ldots, T^n$).

The semantics of type configurations is defined only for filled types. Semantic rules in Fig. 18 are the natural extension of those for the binary case. Rule [M-TS-COM] shows that communication affects only the two interacting parties, without modifying any checkpoint. Rule [M-TS-CMT] sets the checkpoint of the committing party and sets an imposed checkpoint for each other party that has performed at least an action from its current checkpoint. Rule [M-TS-RLL] rolls all parties back to their checkpoints, provided that the checkpoint of the party requesting the rollback is not imposed. Rule [M-TS-ABT] brings all parties back to the initial configuration.

Our notion of *rollback safety*, and the related compliance relation, extend to multiparty sessions as follows. Notice that types T_i in Def. 4 contain placeholders, while in Def. 5 all types are filled.

Definition 4 (Multiparty Rollback safety). *Let C be an initial collaboration, then C is* rollback safe *(shortened* r-safe*) if $C \ \blacktriangleright\ A$ and for each n-tuple $\bar{a}[\mathtt{n}] : T_n, \ldots, a[\mathtt{1}] : T_1$ in A we have $\dashv\!\Vert\ (T_n \cdot \mathtt{n}, \ldots, T_1 \cdot \mathtt{1})$.*

Definition 5 (Compliance for Multiparty Sessions). *Types $T_1, \ldots, T_n$ are* compliant, *written $\dashv\!\Vert\ (T_1, \ldots, T_n)$, if $\dashv\!\Vert\ ((T_1, \ldots, T_n) : \langle T_1\rangle \blacktriangleright T_1, \ldots, \langle T_n\rangle \blacktriangleright T_n)$. Relation $\dashv\!\Vert$ on type configurations is defined as follows: $\dashv\!\Vert\ ((T^1, \ldots, T^n) : \langle \tilde{U}_1\rangle \blacktriangleright T_1, \ldots, \langle \tilde{U}_n\rangle \blacktriangleright T_n)$ holds if for any $\tilde{U}_1'$, T_1', $\ldots$, $\tilde{U}_n'$, T_n' such that $(T^1, \ldots, T^n) : \prod_{h\in\{1,\ldots,n\}}\langle \tilde{U}_h\rangle \blacktriangleright T_h \longmapsto^* (T^1, \ldots, T^n) : \prod_{h\in\{1,\ldots,n\}}\langle \tilde{U}_h'\rangle \blacktriangleright T_h' \not\longmapsto$ we have that $T_1' = \ldots = T_n' = \mathtt{end}$.*

All notions and concepts of our rollback recovery approach smoothly extend to the multiparty case. As consequence, all properties in Sec. 5 still hold in the extended setting;

their proofs are in Appendix A.4. We report below just the key theorem concerning session progress.

Theorem 7 (Multiparty session progress). *Let C be a r-safe collaboration of the form $(\bar{a}[n](x).P_n \mid \prod_{i\in\{1,..,n-1\}} a[i](x).P_i)$. If $C \rightarrowtail^* C'$ then either $C' \rightarrowtail C''$ for some C'' or $C' \equiv (\nu s : C)\prod_{i\in\{1,..,n\}}\langle\tilde{Q}_i\rangle \blacktriangleright \mathbf{0}$ for some $\tilde{Q}_1,\dots,\tilde{Q}_n$.*

8 Conclusion and related work

This paper proposes rollback recovery primitives for session-based programming, which permit checkpointing session participants, rolling back to the last committed checkpoints, or completely aborting a session. These primitives come together with session typing, enabling a design time compliance check that statically ensures checkpoint persistency properties (Lemma 1 and Theorem 3) and session soundness (Theorems 4 and 5), which differ from the properties of the existing (fully) reversible calculi. Our compliance check has been implemented in the MAUDE framework. We have first illustrated our rollback recovery approach in the binary sessions setting, and then we have shown how it smoothly extends to the multiparty case.

In the literature we can distinguish two ways of dealing with rollback: either using explicit rollbacks and implicit commits [20], or by using explicit commits and spontaneous aborts [11,36]. Differently from these works, we have introduced a way to control reversibility by both *triggering* it and *limiting* its scope. Reversibility is triggered by means of an explicit rollback primitive (as in [20]), while explicit commits limit the scope of potential future reverse executions (as in [11,36]). Differently from [11,36], commit does not require any synchronisation, as it is a local decision. This could lead to runtime misbehaviours where a process willing to roll back to its last checkpoint reaches a point which has been imposed by another participant of the session. Our type discipline allows for ruling out programs which may lead to such undesired states.

Reversibility in behavioural types has been studied in different formalisms: *behavioural contracts* [1,3], *binary session types* [27], *multi party session types* [26,5,32,33], and *global graphs* [28,13]. In [1,3] choices can be seen as *implicit* checkpoints and the system can go back to a previous choice and try another branch. In [1] rollback is triggered non-deterministically (e.g., it may happen at any time during the execution), while in [3] it is triggered by the system only when the computation is stuck (e.g., the client cannot further continue). Types information is used at runtime by monitors, for binary [27] and multiparty [26] settings, to keep track of the computational history of the system. There, reversibility is uncontrolled, and each computational step can be undone. In [5] global types are enriched with history information, and choices are seen as labelled checkpoints. At any moment, the party who decided which branch to take in a choice may decide to revert it, forcing the entire system to go back to a point prior to the choice. Hence, rollback is confined inside choices and it is spontaneous. Checkpoints are not seen as commits, and a rollback can bring the system to a state prior to several checkpoints. In [32] an *uncontrolled* reversible variant of session π-calculus is presented, while [33] studies different notions of reversibility for both binary and multiparty single sessions. In [28,13] global graphs are extended with conditions on branches. These conditions at runtime can trigger coordinated rollbacks to revert a distributed choice. Reversibility is confined into branches of a distributed choice and not all the computational steps are reversible; inputs, in fact, are irreversible unless they are inside an ongoing loop. Hence, it is not always possible to revert system computations.

We detach from these works in several ways. First our checkpoint facility is explicit and checkpointing is not relegated to choices: the programmer can decide at any point when to commit. This because the programmer may be interested in committing, besides choice points, a series of interactions (e.g., to make a payment irreversible). Thus, once a commit is taken, the system cannot revert to a state prior to it. Our rollback facility is explicit, meaning that it is the programmer who deliberately triggers a rollback, and it is not the system to decide when to go back. Our extension to the multiparty setting is natural and does not rely on a formalism to describe the global view of the system.

Concerning our compliance check, the relation at its basis resembles those given in [1,2,3], which however are defined for different rollback recovery approaches based on implicit checkpoints (see comments above). Like these compliance checks, also the check we propose is decidable. Concerning the MAUDE implementation of the compliance check, we have followed the approach of the seminal work by Verdejo and Martí-Oliet [35], providing the state-of-the-art implementation of CCS in MAUDE. Along the same line, many other MAUDE implementations of formalisms and languages have been proposed, such as BPMN [8], Twitlang [23], SCEL [4], and QFLan [34].

As future work, we plan to extend our approach to deal with sessions where parties can interleave interactions performed along different sessions. This requires to deal with subordinate sessions, which may affect enclosing sessions by performing, e.g., commit actions that make some interaction of the enclosing sessions irreversible, similarly to *nested transactions* [37]. To tackle this issue it would be necessary to extend the notion of compliance relation to take into account possible partial commits (in case of nested sub-sessions) that could be undone at the top level if a rollback is performed.

References

1. Barbanera, F., Dezani-Ciancaglini, M., de'Liguoro, U.: Compliance for reversible client/server interactions. In: BEAT. EPTCS, vol. 162, pp. 35–42 (2014)
2. Barbanera, F., Dezani-Ciancaglini, M., Lanese, I., de'Liguoro, U.: Retractable contracts. In: PLACES 2015. EPTCS, vol. 203, pp. 61–72 (2016)
3. Barbanera, F., Lanese, I., de'Liguoro, U.: Retractable and speculative contracts. In: COORDINATION. LNCS, vol. 10319, pp. 119–137. Springer (2017)
4. Belzner, L., De Nicola, R., Vandin, A., Wirsing, M.: Reasoning (on) service component ensembles in rewriting logic. In: Specification, Algebra, and Software. LNCS, vol. 8373, pp. 188–211. Springer (2014)
5. Castellani, I., Dezani-Ciancaglini, M., Giannini, P.: Concurrent reversible sessions. In: CONCUR. LIPIcs, vol. 85, pp. 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
6. Chen, T., Dezani-Ciancaglini, M., Scalas, A., Yoshida, N.: On the preciseness of subtyping in session types. Logical Methods in Computer Science **13**(2) (2017)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, LNCS, vol. 4350. Springer (2007)

8. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F.: A formal approach to modeling and verification of business process collaborations. Sci. Comput. Program. **166**, 35–70 (2018)
9. Cristescu, I., Krivine, J., Varacca, D.: A Compositional Semantics for the Reversible p-Calculus. In: LICS. pp. 388–397. IEEE (2013)
10. Danos, V., Krivine, J.: Reversible communicating systems. In: CONCUR. LNCS, vol. 3170, pp. 292–307. Springer (2004)
11. Danos, V., Krivine, J.: Transactions in rccs. In: CONCUR. LNCS, vol. 3653, pp. 398–412. Springer (2005)
12. Engblom, J.: A review of reverse debugging. In: System, Software, SoC and Silicon Debug Conference (S4D). pp. 1–6 (Sept 2012)
13. Francalanza, A., Mezzina, C.A., Tuosto, E.: Reversible choreographies via monitoring in erlang. In: Bonomi, S., Rivière, E. (eds.) Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018. Lecture Notes in Computer Science, vol. 10853, pp. 75–92. Springer (2018)
14. Giachino, E., Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent rollback in a tuple-based language. J. Log. Algebr. Meth. Program. **88**, 99–120 (2017)
15. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP. LNCS, vol. 1381, pp. 122–138. Springer (1998)
16. Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1), 3:1–3:36 (2016)
17. Kouzapas, D., Yoshida, N.: Globally governed session semantics. Logical Methods in Computer Science **10**(4) (2014)
18. Kuhn, S., Ulidowski, I.: Local reversibility in a calculus of covalent bonding. Sci. Comput. Program. **151**, 18–47 (2018)
19. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.: Concurrent flexible reversibility. In: ESOP. LNCS, vol. 7792, pp. 370–390. Springer (2013)
20. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Controlling Reversibility in Higher-Order Pi. In: CONCUR. LNCS, vol. 6901, pp. 297–311. Springer (2011)
21. Lanese, I., Mezzina, C.A., Stefani, J.B.: Reversing higher-order pi. In: CONCUR. LNCS, vol. 6269, pp. 478–493. Springer (2010)
22. Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility. Bulletin of the EATCS **114** (2014), `http://eatcs.org/beatcs/index.php/beatcs/article/view/305`
23. Maggi, A., Petrocchi, M., Spognardi, A., Tiezzi, F.: A language-based approach to modelling and analysis of twitter interactions. J. Log. Algebr. Meth. Program. **87**, 67–91 (2017)
24. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96**(1), 73–155 (1992)
25. Meseguer, J.: Twenty years of rewriting logic. J. Log. Algebr. Program. **81**(7-8), 721–781 (2012)
26. Mezzina, C.A., Pérez, J.A.: Causally consistent reversible choreographies: a monitors-as-memories approach. In: PPDP. pp. 127–138. ACM (2017)
27. Mezzina, C.A., Pérez, J.A.: Reversibility in session-based concurrency: A fresh look. J. Log. Algebr. Meth. Program. **90**, 2–30 (2017)
28. Mezzina, C.A., Tuosto, E.: Choreographies for automatic recovery. CoRR **abs/1705.09525** (2017), `http://arxiv.org/abs/1705.09525`
29. Milner, R.: Communication and concurrency. Prentice-Hall (1989)
30. Perumalla, K.S., Protopopescu, V.A.: Reversible simulations of elastic collisions. ACM Trans. Model. Comput. Simul. **23**(2), 12:1–12:25 (2013)

31. Prabhu, P., Ramalingam, G., Vaswani, K.: Safe programmable speculative parallelism. In: Zorn, B.G., Aiken, A. (eds.) PLDI. pp. 50–61. ACM (2010)
32. Tiezzi, F., Yoshida, N.: Reversible session-based pi-calculus. J. Log. Algebr. Meth. Program. **84**(5), 684–707 (2015)
33. Tiezzi, F., Yoshida, N.: Reversing single sessions. In: RC. LNCS, vol. 9720, pp. 52–69. Springer (2016)
34. Vandin, A., ter Beek, M.H., Legay, A., Lluch-Lafuente, A.: Qflan: A tool for the quantitative analysis of highly reconfigurable systems. In: FM. LNCS, vol. 10951, pp. 329–337. Springer (2018)
35. Verdejo, A., Martí-Oliet, N.: Implementing CCS in Maude 2. In: WRLA. ENTCS, vol. 71, pp. 239–257. Elsevier (2002)
36. de Vries, E., Koutavas, V., Hennessy, M.: Communicating transactions - (extended abstract). In: Gastin, P., Laroussinie, F. (eds.) CONCUR. LNCS, vol. 6269. Springer (2010)
37. Weikum, G., Schek, H.J.: Concepts and applications of multilevel transactions and open nested transactions. In: Database Transaction Models for Advanced Applications. pp. 515–553. Morgan Kaufmann (1992)
38. Yoshida, N., Vasconcelos, V.T.: 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)

A Proofs

A.1 Decidability result

Theorem 1. Let T_1 and T_2 be two session types, checking if $T_1 \dashv\!| \; T_2$ holds is decidable.

Proof. By Def. 1, checking $T_1 \dashv\!| \; T_2$ consists in checking that types T'_1 and T'_2 of each configuration $(T_1, T_2) : \langle \tilde{U'_1} \rangle \blacktriangleright T'_1 \parallel \langle \tilde{U'_2} \rangle \blacktriangleright T'_2$ such that $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^* (T_1, T_2) : \langle \tilde{U'_1} \rangle \blacktriangleright T'_1 \parallel \langle \tilde{U'_2} \rangle \blacktriangleright T'_2 \not\longmapsto$ (i.e., type configurations that are reachable from the initial one and that cannot further evolve) are `end` types. Thus, to prove that the compliance check is decidable we have to show that the number of these reachable configurations is finite. Let us consider the transition system $TS = \langle \mathcal{S}, \mathcal{R} \rangle$ associated to the type configuration $t = (T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2$ by the reduction semantics of types (Fig. 11): the set $\mathcal{S}$ of states corresponds to the set of type configurations reachable from t, i.e. $\mathcal{S} = \{ t' \mid t \longmapsto^* t' \}$, while the set $\mathcal{R}$ of system transitions corresponds to set of the type reductions involving configurations in $\mathcal{S}$, i.e. $\mathcal{R} = \{(t', t'') \in \mathcal{S} \times \mathcal{S} \mid t' \longmapsto t''\}$. Hence, checking $T_1 \dashv\!| \; T_2$ boils down to check the type configurations corresponding to the leaves (i.e., states without outgoing transitions) of TS. Specifically, given a leaf of TS corresponding to $(T_1, T_2) : \langle \tilde{V_1} \rangle \blacktriangleright V_2 \parallel \langle \tilde{V_3} \rangle \blacktriangleright V_4$, we have to check if $V_2 = V_4 = \texttt{end}$. The decidability of this check therefore depends on the finiteness of TS. This result is ensured by the fact that: : *(i)* backward reductions connect states of TS only to previously visited states of TS (Theorem 2), and *(ii)* our language of types (Fig. 7) corresponds to a CCS-like process algebra without static operators (i.e., parallel and restriction operators) within recursion (see [29, Sec. 7.5]).

A.2 Reversibility results

We can instrument our semantics in order to carry the information on which session the reduction is taking place. Hence we will indicate with $C \overset{s}{\rightarrowtail} C'$ the fact that the reduction is taking place on session s. We can show that sessions are independent:

Lemma 3 (Swap Lemma). *Let C be a collaboration and s and r two sessions. If $C \overset{s}{\rightarrowtail} C_1 \overset{r}{\rightarrowtail} C_2$ then there exists a collaboration C_3 such that $C \overset{r}{\rightarrowtail} C_3 \overset{s}{\rightarrowtail} C_2$.*

Proof. By case analysis on the reductions $\overset{s}{\rightarrowtail}$ and $\overset{r}{\rightarrowtail}$.

Lemma 4. *Let C be a collaboration. If $C \rightarrowtail^* C_1$, then for any session s in C_1 there exists a collaboration C_0 such that $C \rightarrowtail^* C_0 \overset{s}{\rightarrowtail} {}^* C_1$ and s is never used in the trace $C \rightarrowtail^* C_0$.*

Proof. By induction on the number n of reduction on s. If there are no reductions then the thesis banally holds. Otherwise we can take the very last reduction on s, that is the closest one to C_1 and iteratively apply Lemma 3 in order to bring it to the very end. Then we can conclude by induction on a trace with less occurrences of reductions on s.

Thanks to Lemma 4 we can rearrange any trace as a sequence of *independent* sessions. Moreover given an initial collaboration C for any reachable collaboration C_1 and session s such that $C \rightarrowtail^* C_1 \overset{s}{\rightarrowtail}{}^*$ and $s \notin\rightarrowtail^*$, we indicate C_1 as the *initial* collaboration for s. This will allows us to focus just on a sigle session, say s, and to consider collaboration initial for s without loosing of generality.

Lemma 5. *Let C be an initial collaboration such that $C \rightarrowtail^* C_1$. If $C_1 \overset{abt}{\rightsquigarrow} C_2$ then $C_2 \equiv C$.*

Proof. Since C is *initial*, without losing of generality we can assume

$$C \equiv \bar{a}(x_1).P_1 \mid a(x_2).P_2$$

The first reduction of $C \rightarrowtail^* C_1$ has to be an application of rule [F-CON], that is

$$\begin{aligned} C \twoheadrightarrow & (\nu s : (\bar{a}(x_1).P_1 \mid a(x_2).P_2)) \\ & (\langle P_1[\bar{s}/x_1]\rangle \blacktriangleright P_1[\bar{s}/x_1] \ \mid \ \langle P_2[s/x_2]\rangle \blacktriangleright P_2[s/x_2]) = C' \end{aligned}$$

and, by hypothesis, $C' \rightarrowtail^* C_1$.

Now, no matter the shape of processes in C_1 by applying rule [B-ABT], and possibly [B-STR], we will go back to C, that is $C_1 \overset{abt}{\rightsquigarrow} C$, as desired.

The following lemma states that a rollback leads back to the last committed checkpoint.

Lemma 6. *Let C be a reachable collaboration, such that $C \overset{cmt}{\twoheadrightarrow} C_1$. If $C_1 \twoheadrightarrow^* C_2 \overset{roll}{\rightsquigarrow} C_3$ and there is no commit in $C_1 \twoheadrightarrow^* C_2$, then $C_3 \equiv C_1$.*

Proof. Since C is a reachable collaboration, this implies it has been generated from an initial collaboration C_0. Without losing of generality, similarly to the Lemma 5's proof, we can assume $C \equiv (\nu s : C_0)(\langle P_1\rangle \blacktriangleright P_2 \mid \langle Q_1\rangle \blacktriangleright Q_2) \mid C^\circ$. Therefore, we have that $C_1 = (\nu s : C_0)(\langle P_2\rangle \blacktriangleright P_2 \mid \langle Q_2\rangle \blacktriangleright Q_2) \mid C^\circ$. By hypothesis, there is no commits in $C_1 \twoheadrightarrow^* C_2$, and this implies that the log part of the C_1 will never change. Hence, we have that $C_2 \equiv (\nu s : C_0)(\langle P_2\rangle \blacktriangleright P \mid \langle Q_2\rangle \blacktriangleright Q) \mid C^\circ$ for some processes P and Q. By applying [B-RLL] and [B-PAR] we have that $C_2 \overset{roll}{\rightsquigarrow} (\nu s : C_0)(\langle P_2\rangle \blacktriangleright P_2 \mid \langle Q_2\rangle \blacktriangleright Q_2) \mid C^\circ \equiv C_1$, as desired.

Any reachable collaboration is also a *forward only* reachable collaboration. Formally:
Theorem 2. Let C_0 be an initial collaboration. If $C_0 \rightarrowtail^* C_1$ then $C_0 \twoheadrightarrow^* C_1$.

Proof. By induction on the number n of backward reductions contained into $C_0 \rightarrowtail^* C_1$. The base case ($n = 0$) trivially holds. In the inductive case, let us take the backward reduction which is the nearest to C_0. That is:

$$C_0 \twoheadrightarrow^* C' \rightsquigarrow C'' \rightarrowtail^* C_1$$

Depending whether it is an $\overset{abt}{\rightsquigarrow}$ or a $\overset{roll}{\rightsquigarrow}$ we can apply respectively Lemma 5 or Lemma 6 to obtain a forward trace of the form

$$C_0 \twoheadrightarrow^* C'' \rightarrowtail^* C_1$$

and we can conclude by applying the inductive hypothesis on the obtained trace which contains less backward moves with respect to the original one.

Lemma 1. *Let C_1 and C_2 be two reachable collaborations. $C_1 \rightsquigarrow C_2$ then $C_2 \twoheadrightarrow^* C_1$.*

Proof. Since C_1 is a reachable collaboration, we have that there exists an initial collaboration C_0 such that $C_0 \rightarrowtail^* C_1$. By applying Theorem 2 we can rearrange the trace such that it contains just forward transitions as follows

$$C_0 \twoheadrightarrow^* C_1$$

If the backward reduction is obtained by applying [B-Abt], by Lemma 5 we have $C_2 \equiv C_0$, from which the thesis trivially follows. Instead, if the backward reduction is obtained by applying [B-Rll], we proceed by case analysis depending on the presence of commit reductions in the trace. If they are present, we select the last of such commit, that is we can decompose the trace in the following way:

$$C_0 \twoheadrightarrow^* \overset{cmt}{\twoheadrightarrow} C_{cmt} \twoheadrightarrow^* C_1 \overset{roll}{\rightsquigarrow} C_2$$

and by applying Lemma 6 we have that $C_2 \twoheadrightarrow^* \equiv C_1$ as desired.

In the case there is no commit in the trace, without losing of generality we can assume

$$C_0 \equiv \bar{a}(x_1).P_1 \mid a(x_2).P_2 \mid C^\circ$$
$$C_0 \twoheadrightarrow (\nu s : C_0)(\langle P_2 \rangle \blacktriangleright P_2 \mid \langle Q_2 \rangle \blacktriangleright Q_2) \mid C^\circ$$

By rule [B-Rll], $C_2 \equiv (\nu s : C_0)(\langle P_2 \rangle \blacktriangleright P_2 \mid \langle Q_2 \rangle \blacktriangleright Q_2) \mid C^\circ$. Thus, we can conclude by noticing that $C_0 \twoheadrightarrow C_2$ must be the first reduction in $C_0 \twoheadrightarrow^* C_1$.

Lemma 2. *Let C be a reachable collaboration. If $C \overset{roll}{\rightsquigarrow} C'$ and $C \overset{roll}{\rightsquigarrow} C''$ then $C' \equiv C''$.*

Proof. Since C is a reachable collaboration, it is has been generated by an initial collaboration C_0 of the form $C_0 = \bar{a}(x).P_1 \mid a(x).P_2$, and by Theorem 2 we have that $C_0 \twoheadrightarrow^* C$. We distinguish two cases, whether in the trace there has been at least one commit or not. In the first case, we can decompose the trace in such a way to single out the last commit as follows:

$$C_0 \twoheadrightarrow^* C_{cmt} \twoheadrightarrow^* C$$

so that in the reduction $C_{cmt} \twoheadrightarrow^* C$ there is no commit. If from C the rollbacks $C \overset{roll}{\rightsquigarrow} C'$ and $C \overset{roll}{\rightsquigarrow} C''$ are triggered by the same process, the thesis trivially follows. In the other case, we have that:

$$C_{cmt} \equiv (\nu s : C_0)(\langle P_1 \rangle \blacktriangleright P_2 \mid \langle Q_1 \rangle \blacktriangleright Q_2) \mid C^\circ$$

with both P_2 and Q_2 are able to trigger a rollback. If the roll action is executed by P_2 we have that

$$(\nu s : C_0)(\langle P_1\rangle \blacktriangleright P_2 \mid \langle Q_1\rangle \blacktriangleright Q_2) \mid C^\circ \overset{roll}{\rightsquigarrow} (\nu s : C_0)(\langle P_1\rangle \blacktriangleright P_1 \mid \langle Q_1\rangle \blacktriangleright Q_1) \mid C^\circ = C'$$

If the roll is triggered by Q_2 we have that

$$(\nu s : C_0)(\langle P_1\rangle \blacktriangleright P_2 \mid \langle Q_1\rangle \blacktriangleright Q_2) \mid C^\circ \overset{roll}{\rightsquigarrow} (\nu s : C_0)(\langle P_1\rangle \blacktriangleright P_1 \mid \langle Q_1\rangle \blacktriangleright Q_1) \mid C^\circ = C''$$

And we can conclude by noticing that $C' \equiv C''$, as desired.

Theorem 3. Let C be a reachable collaboration. If $C \overset{cmt}{\twoheadrightarrow} C'$ then there exists no C'' such that $C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$ and $C'' \twoheadrightarrow^+ C$.

Proof. We proceed by contradiction. Suppose that there exists C'' such that $C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$ and $C'' \twoheadrightarrow^+ C$. Since C is a reachable collaboration, thanks to Theorem 2 we have that there exists an initial collaboration C_0 such that $C_0 \twoheadrightarrow^* C \overset{cmt}{\twoheadrightarrow} C'$. Since a rollback brings back the collaboration to a point before a commit, this means it has been restored a commit previous to the last one (by [B-RLL], indeed, only processes stored in logs can be committed). This implies that there exist at least two different commits in the trace such that

$$C_0 \twoheadrightarrow^* C_{cmt} \twoheadrightarrow^* C \overset{cmt}{\twoheadrightarrow} C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$$

with $C'' = C_{cmt}$. We have that $C'' \twoheadrightarrow^+ C \overset{cmt}{\twoheadrightarrow} C' \twoheadrightarrow^* C_{rl}$, where C_{rl} is the collaboration that perform the roll reduction. Now since the last commit has been done by C, supposing that the commit is triggered by P_c, which evolves to P_c' in doing that, we have that:

$$C \equiv (\nu s : C_0)(\langle P\rangle \blacktriangleright P_c \mid \langle Q\rangle \blacktriangleright Q_c) \;\text{ and }\; C' = (\nu s : C_0)(\langle P_c'\rangle \blacktriangleright P_c' \mid \langle Q_c\rangle \blacktriangleright Q_c)$$

Now, by hypothesis we have that $C' \twoheadrightarrow^* C_{rl}$ without any commit being present in the trace, hence:

$$C_{rl} \equiv (\nu s : C_0)(\langle P_c'\rangle \blacktriangleright P_{rl} \mid \langle Q_c\rangle \blacktriangleright Q_{rl})$$

By hypothesis, from C_{rl} a rollback is possible. Regardless the rollback is triggered by P_{rl} or Q_{rl}, we have that $C_{rl} \overset{roll}{\rightsquigarrow} C'$. Now, from C' we cannot reach C ($C' \not\twoheadrightarrow^+ C$), as C' is derived from C and the rollback can only bring the collaboration back to C'. This violates the hypothesis, and hence we conclude.

A.3 Soundness results

To prove our soundness results, we need to introduce some auxiliary lemmas, which rely on the following definitions:

- a process $\tilde{P}$ and a type $\tilde{T}$ are in *checkpoint accordance* if $\tilde{P} = P$ implies $\tilde{T} = T$, and $\tilde{P} = \underline{P}$ implies $\tilde{T} = \underline{T}$;
- let ℓ a process label, its *dual label* $\bar{\ell}$ is defined as follows: $\overline{k!\langle v\rangle} = k?$, $\overline{k?(x)} = k!$, $\overline{k \lhd l} = k \rhd l$, $\overline{k \rhd l} = k \lhd l$; this notion of duality straightforwardly extends to type labels;

- the function $tl_{\Gamma}(\cdot)$, mapping process labels to type labels under sorting Γ, is defined as follows: $tl_{\Gamma}(k!\langle v\rangle) = ![S]$ with $\Gamma \vdash v \blacktriangleright S$, $tl_{\Gamma}(k?(x)) = ?[S]$ with $\Gamma \vdash x \blacktriangleright S$, $tl_{\Gamma}(k \lhd l) = \lhd l$, $tl_{\Gamma}(k \rhd l) = \rhd l$, $tl_{\Gamma}(\mathit{cmt}) = \texttt{cmt}$, $tl_{\Gamma}(\mathit{roll}) = \texttt{roll}$, $tl_{\Gamma}(\mathit{abt}) = \texttt{abt}$, and $tl_{\Gamma}(\tau) = \tau$.

The following lemma states that each reduction of a reachable collaboration corresponds to a reduction of its configuration types.

Lemma 7. *Let* $C = (\nu s : C')(\langle \tilde{P_1}\rangle \blacktriangleright P_2 \mid \langle \tilde{Q_1}\rangle \blacktriangleright Q_2)$ *be a reachable collaboration, with* $C' = (\bar{a}(x).P \mid a(y).Q)$, $(\varnothing;\varnothing \vdash P \blacktriangleright x : T)$, $(\varnothing;\varnothing \vdash Q \blacktriangleright y : T')$, $T \dashv\!\vert\, T'$, $(\Theta_1;\Gamma_1 \vdash P_1[x/\bar{s}] \blacktriangleright x : T_1)$, $(\Theta_2;\Gamma_2 \vdash P_2[x/\bar{s}] \blacktriangleright x : T_2)$, $(\Theta'_1;\Gamma'_1 \vdash Q_1[y/s] \blacktriangleright y : U_1)$, *and* $(\Theta'_2;\Gamma'_2 \vdash Q_2[y/s] \blacktriangleright y : U_2)$. *If* $C \rightarrowtail (\nu s : C')(\langle \tilde{P'_1}\rangle \blacktriangleright P'_2 \mid \langle \tilde{Q'_1}\rangle \blacktriangleright Q'_2)$ *then there exist* T'_1, T'_2, U'_1, U'_2 *such that* $(T,T') : \langle \tilde{T_1}\rangle \blacktriangleright T_2 \parallel \langle \tilde{U_1}\rangle \blacktriangleright U_2 \longmapsto (T,T') : \langle \tilde{T'_1}\rangle \blacktriangleright T'_2 \parallel \langle \tilde{U'_1}\rangle \blacktriangleright U'_2$ *with* $\tilde{P'_1}$ *(resp.* $\tilde{Q'_1}$*) in checkpoint accordance with* $\tilde{T'_1}$ *(resp.* $\tilde{U'_1}$*),* $(\hat{\Theta}_1;\hat{\Gamma}_1 \vdash P'_1[x/\bar{s}] \blacktriangleright x : T'_1)$, $(\hat{\Theta}_2;\hat{\Gamma}_2 \vdash P'_2[x/\bar{s}] \blacktriangleright x : T'_2)$, $(\hat{\Theta}'_1;\hat{\Gamma}'_1 \vdash Q'_1[y/s] \blacktriangleright y : U'_1)$, *and* $(\hat{\Theta}'_2;\hat{\Gamma}'_2 \vdash Q'_2[y/s] \blacktriangleright y : U'_2)$.

Proof. We have two cases depending whether the reduction $\rightarrowtail$ has forward or backward direction.

($\rightarrowtail = \twoheadrightarrow$). From rule [F-RES], we have $\langle \tilde{P_1}\rangle \blacktriangleright P_2 \mid \langle \tilde{Q_1}\rangle \blacktriangleright Q_2 \twoheadrightarrow \langle \tilde{P'_1}\rangle \blacktriangleright P'_2 \mid \langle \tilde{Q'_1}\rangle \blacktriangleright Q'_2$. We prove the result by case analysis on the last rule applied in the inference of the above reduction.

- [F-COM]. In this case we have $P_2 \xrightarrow{\bar{s}!\langle v\rangle} P'_2$, $Q_2 \xrightarrow{s?(z)} Q''_2$, with $Q'_2 = Q''_2[v/z]$, $\tilde{P'_1} = \tilde{P_1}$ and $\tilde{Q'_1} = \tilde{Q_1}$. Thus, $P_2[x/\bar{s}] = x!\langle e\rangle.P'_2[x/\bar{s}]$ for some e such that $e \downarrow v$, and $Q_2[y/s] = y?(z : S').Q'_2[y/s]$. By rule [T-SND], we have that $T_2 = ![S].T'_2$, with $\Gamma_2 \vdash e \blacktriangleright S$, (hence $\Gamma_2 \vdash v \blacktriangleright S$), and $\Theta_2;\Gamma_2 \vdash P'_2[x/\bar{s}] \blacktriangleright x : T'_2$ (hence $\hat{\Theta}_2 = \Theta_2$ and $\hat{\Gamma}_2 = \Gamma_2$). Similarly, by rule [T-RCV], we have that $U_2 = ?[S'].U'_2$ and $\Theta'_2;\Gamma'_2 \cdot z : S' \vdash Q''_2[y/s] \blacktriangleright y : U'_2$ (hence $\hat{\Theta}'_2 = \Theta'_2$ and $\hat{\Gamma}'_2 = \Gamma'_2 \cdot z : S'$). By rules [TS-SND] and [TS-RCV], we get $T_2 \xrightarrow{![S]} T'_2$ and $U_2 \xrightarrow{?[S']} U_2$. Now, reasoning by contradiction, let us suppose that $S \neq S'$. Thus, the term $(T,T') : \langle \tilde{T_1}\rangle \blacktriangleright T_2 \parallel \langle \tilde{U_1}\rangle \blacktriangleright U_2 \not\longmapsto$, since no rule in Fig. 11 can be applied. However, since C is a reachable collaboration, this type configuration is originated from $(T,T') : \langle T\rangle \blacktriangleright T \parallel \langle T'\rangle \blacktriangleright T'$. By Def. 1, $T \dashv\!\vert\, T'$ implies $T_2 = U_2 = \texttt{end}$, which is a contradiction. Therefore, it holds that $S = S'$. Hence, by applying rule [TS-COM] we can conclude.
- [F-LAB], [E-CMT$_1$] and [E-CMT$_2$]. Similar to the previous case.
- [F-STR]. The use of $\equiv$ leads us back to one of the other cases.
- [F-PAR]. In this case we have that $\langle \tilde{P_1}\rangle \blacktriangleright P_2 \twoheadrightarrow \langle \tilde{P'_1}\rangle \blacktriangleright P'_2$. Since this transition involves only one log term, it can be inferred only by applying rule [F-IF], from which we have $P_2 \xrightarrow{\tau} P'_2$ and $\tilde{P'_1} = \tilde{P_1}$. By rule [P-IFT] (the case of rule [P-IFF] is similar), we have $P_2[x/\bar{s}] = \textsf{if}\ e\ \textsf{then}\ P'_2[x/\bar{s}]\ \textsf{else}\ R$ with $e \downarrow \texttt{true}$. By rule [T-IF] we get $T_2 = T'_2 \oplus V$ and $\Theta_2;\Gamma_2 \vdash P'_2[x/\bar{s}] \blacktriangleright x : T'_2$. By rule [TS-IF$_1$], $T_2 \xrightarrow{\tau} T'_2$. By applying rule [TS-TAU] we can conclude.

($\rightarrowtail = \rightsquigarrow$). From rule [B-RES], we have $\langle \tilde{P_1}\rangle \blacktriangleright P_2 \mid \langle \tilde{Q_1}\rangle \blacktriangleright Q_2 \rightsquigarrow \langle \tilde{P'_1}\rangle \blacktriangleright P'_2 \mid \langle \tilde{Q'_1}\rangle \blacktriangleright Q'_2$. We prove the result by case analysis on the last rule applied in the inference of the above reduction.

- [B-RLL]. In this case $P_2 \xrightarrow{roll} P_2'$, $\tilde{P}_1' = \tilde{P}_1$, $\tilde{Q}_1' = Q_1$ and $Q_2' = Q_1$. By rule [P-RLL], we have $P_2[x/\bar{s}] = \mathsf{roll}$ and $P_2'[x/\bar{s}] = \mathbf{0}$. By rule [T-RLL] we get $T_2 = \mathtt{roll}$. By rule [TS-RLL], $T_2 \xrightarrow{\mathtt{roll}} T_2'$, with $T_2' = \mathtt{end}$. By rule [T-INACT], we have $\Theta_2; \Gamma_2 \vdash P_2'[x/\bar{s}] \blacktriangleright x : T_2'$. Finally, by applying rule [TS-RLL$_1$] we can conclude.
- [B-STR] and [B-PAR]. Similarly to the forward cases.

The following lemma relates collaboration reductions to type reductions when a `roll_error` is produced.

Lemma 8. *Let $C = (\bar{a}(x).P \mid a(y).Q)$ such that $C \blacktriangleright \{\bar{a} : T_1, a : T_2\}$ and $T_1 \dashv\Vert T_2$. If $C \rightarrowtail^* (\nu s : C)(\langle \underline{P_1} \rangle \blacktriangleright P_2 \mid \langle \tilde{Q}_1 \rangle \blacktriangleright Q_2)$ and $P_2 \xrightarrow{roll} P_2'$, then there exist U_1, U_2, U_1', U_2', U_1'' such that $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^* (T_1, T_2) : \langle \underline{U_1} \rangle \blacktriangleright U_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright U_2'$ and $U_1' \xrightarrow{\mathtt{roll}} U_1''$.*

Proof. From $C \blacktriangleright \{\bar{a} : T_1, a : T_2\}$, by applying [T-PAR], [T-ACC] and [T-REQ], we have that $\emptyset; \emptyset \vdash P \blacktriangleright x : T_1$ and $\emptyset; \emptyset \vdash Q \blacktriangleright x : T_2$. By applying rule [F-CON] to the collaboration C, we obtain $C \twoheadrightarrow (\nu s : C)(\langle P[\bar{s}/x] \rangle \blacktriangleright P[\bar{s}/x] \mid \langle Q[s/y] \rangle \blacktriangleright Q[s/y]) = C'$. Now, by repeatedly applying Lemma 7, from $C' \rightarrowtail^* (\nu s : C)(\langle \underline{P_1} \rangle \blacktriangleright P_2 \mid \langle \tilde{Q}_1 \rangle \blacktriangleright Q_2)$ we get $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^* (T_1, T_2) : \langle \underline{U_1} \rangle \blacktriangleright U_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright U_2'$ for some U_1, U_2, U_1', U_2', with $\Theta_2; \Gamma_2 \vdash P_2[x/\bar{s}] \blacktriangleright x : U_1'$. Now, let us consider the transition $P_2 \xrightarrow{roll} P_2'$. This can be derived only by the application of rules [P-RLL]. Thus, $P_2 = \mathsf{roll}$, from which we have $P_2[x/\bar{s}] = \mathsf{roll}$. From $\Theta_2; \Gamma_2 \vdash \mathsf{roll} \blacktriangleright x : U_1'$, by rule [T-RLL], we get $U_1' = \mathsf{roll}$. Therefore, by rule [TS-RLL], we can conclude $U_1' \xrightarrow{\mathtt{roll}} U_1''$ with $U_1'' = \mathtt{end}$.

The following lemma relates collaboration reductions to type reductions when a `com_error` is produced.

Lemma 9. *Let $C = (\bar{a}(x).P \mid a(y).Q)$ such that $C \blacktriangleright \{\bar{a} : T_1, a : T_2\}$ and $T_1 \dashv\Vert T_2$. If $C \rightarrowtail^* (\nu s : C)(\langle \tilde{P}_1 \rangle \blacktriangleright P_2 \mid \langle \tilde{Q}_1 \rangle \blacktriangleright Q_2)$, $P_2 \xrightarrow{\ell} P_2'$ and $\neg Q_2 \Downarrow_{\bar{\ell}}$ with ℓ of the form $k!\langle v \rangle$, $k?(x)$, $k \lhd l$ or $k \rhd l$, then there exist U_1, U_2, U_1', U_2', U_1'' such that $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^* (T_1, T_2) : \langle \tilde{U}_1 \rangle \blacktriangleright U_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright U_2'$ with $\tilde{P}_1$ (resp. $\tilde{Q}_1$) in checkpoint accordance with $\tilde{U}_1$ (resp. $\tilde{U}_2$), $U_1' \xrightarrow{tl_\Gamma(\ell)} U_1''$, with Γ sorting for typing P_2', and for all U_2'' such that $U_2' \xrightarrow{\tau}^* U_2''$ we have $U_2'' \not\xrightarrow{\overline{tl_\Gamma(\ell)}}$.*

Proof. From $C \blacktriangleright \{\bar{a} : T_1, a : T_2\}$, by applying [T-PAR], [T-ACC] and [T-REQ], we have that $\emptyset; \emptyset \vdash P \blacktriangleright x : T_1$ and $\emptyset; \emptyset \vdash Q \blacktriangleright x : T_2$. By applying rule [F-CON] to the collaboration C, we obtain $C \twoheadrightarrow (\nu s : C)(\langle P[\bar{s}/x] \rangle \blacktriangleright P[\bar{s}/x] \mid \langle Q[s/y] \rangle \blacktriangleright Q[s/y]) = C'$. Now, by repeatedly applying Lemma 7, from $C' \rightarrowtail^* (\nu s : C)(\langle \tilde{P}_1 \rangle \blacktriangleright P_2 \mid \langle \tilde{Q}_1 \rangle \blacktriangleright Q_2)$ we get $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^* (T_1, T_2) : \langle \tilde{U}_1 \rangle \blacktriangleright U_1' \parallel \langle \tilde{U}_2 \rangle \blacktriangleright U_2'$ for some U_1, U_2, U_1', U_2', with $\Theta_2; \Gamma_2 \vdash P_2[x/\bar{s}] \blacktriangleright x : U_1'$ and $\Theta_2'; \Gamma_2' \vdash Q_2[y/s] \blacktriangleright y : U_2'$. Now, let us reason by case analysis on the rule for deriving the transition $P_2 \xrightarrow{\ell} P_2'$.

Rule [P-SND]. Thus, $P_2 = \bar{s}!\langle e \rangle.P_2'$ and $\ell = \bar{s}!\langle v \rangle$ with $e \downarrow v$. From $\Theta_2; \Gamma_2 \vdash P_2[x/\bar{s}] \blacktriangleright x : U_1'$, by rule [T-SND], we get $U_1' =![S].U_1''$ with $\Gamma \vdash e \blacktriangleright S$. Therefore, by rule [TS-SND], we get $U_1' \xrightarrow{![S]} U_1''$.

Rule [P-RCV]. Thus, $P_2 = \bar{s}?(y : S).P_2'$ and $\ell = \bar{s}?(y)$. From $\Theta_2; \Gamma_2 \vdash P_2[x/\bar{s}] \blacktriangleright x : U_1'$, by rule [T-RCV], we get $U_1' = ?[S].U_1''$. Therefore, by rule [TS-RCV], we get $U_1' \xrightarrow{?[S]} U_1''$.

Rules [P-SEL] and [P-BRN]. Similar to the previous cases.

Rule [P-STR]. According to the definition of $\equiv$, the only relevant case is when $P_2 = \mu X.R$. Thus, by rule [P-STR] we have $P_2 \equiv R[\mu X.R/X] = P_2''$, and $P_2'' \xrightarrow{\ell} P_2'''$ with $P_2''' \equiv P_2'$. If P_2'' has not the form $\mu Y.R'$, we can proceed as in one of the cases above, otherwise we repeat this reasoning.

Finally, from $\neg Q_2 \Downarrow_{\bar{\ell}}$, following a similarly reasoning, we can conclude $U_2' \xrightarrow{\tau}{}^{*} \not\xrightarrow{\overline{tl_{\Gamma}(\ell)}}$.

We can now prove our soundness results.

Theorem 4. If C is a rollback safe collaboration, then $C \not\rightarrowtail^{*} \mathbb{C}[\mathtt{roll\_error}]$.

Proof. The proof proceeds by contradiction. Suppose that there exists an initial collaboration C that is rollback safe and such that $C \rightarrowtail^{*} \mathbb{C}[\mathtt{roll\_error}]$. The erroneous collaboration $\mathtt{roll\_error}$ can be only produced by applying rule [E-RLL$_2$]. Thus, to infer at least one reduction of the sequence $C \rightarrowtail^{*} \mathbb{C}[\mathtt{roll\_error}]$, rule [E-RLL$_2$] must be used. From this, we have that there exists a runtime collaboration $C' \equiv \mathbb{C}'[C'']$, with $C'' = (\langle \underline{Q_1} \rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2} \rangle \blacktriangleright P_2)$, such that $C \rightarrowtail^{*} C'$, $P_1 \xrightarrow{roll} P_1'$, and $C' \twoheadrightarrow \mathbb{C}'[\mathtt{roll\_error}] \rightarrowtail^{*} \mathbb{C}[\mathtt{roll\_error}]$. By rules [F-CON], [F-RES] and [B-RES], and the fact that the scope of operator $(\nu s : \_)$ is statically defined (i.e., neither the operational rules nor $\equiv$ allow scope extension), the term C'' can only be the argument of the operator $(\nu s : C_1)$, i.e. $\mathbb{C}' = C_2 \mid (\nu s : C_1)[\cdot]$, with $C_1 = \bar{a}(x).P \mid a(y).Q$ for some a, x, y, P and Q. In its own turn, the term $(\nu s : C_1)C''$ can only be generated by applying rule [F-CON] from C_1, which must be a subterm of C, i.e. $C \equiv C_1 \mid C_2'$ for some C_2'. Since the scope of $(\nu s : \_)$ operator cannot be extended, all reductions performed by terms in parallel with it by applying rules [F-PAR] and [B-PAR] do not affect the argument of such operator. Therefore, focussing on the subterm C_1 of C, by exploiting rules [F-PAR] and [B-PAR] we can set apart the reductions in $C \rightarrowtail^{*} \mathbb{C}[\mathtt{roll\_error}]$ involving C_1 and its derivatives, thus obtaining $C_1 \rightarrowtail^{*} (\nu s : C_1)C'' \twoheadrightarrow (\nu s : C_1)\mathtt{roll\_error}$.

Now, since C is rollback safe, by Def. 2 we have that $C \blacktriangleright A$ and for all pairs $\bar{b} : V_1$ and $b : V_2$ in A we have $V_1 \dashv\| V_2$. Since $C \equiv C_1 \mid C_2'$, by rule [T-PAR] we obtain $C_1 \blacktriangleright A_1$ with $A_1 \subseteq A$. By rules [T-REQ] and [T-ACC], we have $A_1 = \{\bar{a} : T_1, a : T_2\}$. Since A_1 is a subset of A_2, that is $A_1 \subseteq A_2$, we have that $T_1 \dashv\| T_2$.

By Lemma 8, we have that there exist U_1, U_2, U_1', U_2', U_1'' such that $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^{*} (T_1, T_2) : \langle \underline{U_1} \rangle \blacktriangleright U_1' \parallel \langle \tilde{U_2} \rangle \blacktriangleright U_2' = t$ and $U_1' \xrightarrow{\mathtt{roll}} U_1''$. No rule in Fig 11 allows the term t to evolve, i.e. $t \not\longmapsto$, because U_1' can only perform $\mathtt{roll}$ and rule [TS-RLL$_1$] cannot be applied due to the imposed checkpoint $\underline{U_1}$. Since $T_1 \dashv\| T_2$, by Def. 1 it must hold that $U_1' = \mathtt{end}$. However, Since U_1' is able to perform an action (as it holds that $U_1' \xrightarrow{\mathtt{roll}} U_1''$), we get that it cannot be an end type, i.e. $U_1' \neq \mathtt{end}$, which is a contradiction.

Theorem 5. If C is a rollback safe collaboration, then $C \not\rightarrowtail^* \mathbb{C}[\texttt{com\_error}]$.

Proof. The proof proceeds by contradiction. Suppose that there exists an initial collaboration C that is rollback safe and such that $C \rightarrowtail^* \mathbb{C}[\texttt{com\_error}]$. The erroneous collaboration $\texttt{com\_error}$ can be produced by applying one of the rules [E-COM1], [E-COM2], [E-LAB1] and [E-LAB2]. Let us consider the case [E-COM1], the other cases are similar. Proceeding as in the proof of Theorem 4, without loss of generality we can focus on the subterm $C_1 = (\bar{a}(x).P \mid a(y).Q)$ of C, such that $C_1 \rightarrowtail^* (\nu s : C_1)C'' \twoheadrightarrow (\nu s : C_1)\texttt{com\_error}$, with $C'' = (\langle \tilde{Q_1} \rangle \blacktriangleright P_1 \mid \langle \tilde{Q_2} \rangle \blacktriangleright P_2)$, and $C_1 \blacktriangleright \{\bar{a} : T_1, a : T_2\}$, with $T_1 \dashv\| T_2$.

By Lemma 9, we have that there exist U_1, U_2, U_1', U_2', U_1'' such that U_1, U_2, U_1', U_2', U_1'' such that $(T_1, T_2) : \langle T_1 \rangle \blacktriangleright T_1 \parallel \langle T_2 \rangle \blacktriangleright T_2 \longmapsto^* (T_1, T_2) : \langle \tilde{U_1} \rangle \blacktriangleright U_1' \parallel \langle \tilde{U_2} \rangle \blacktriangleright U_2' = t$ with $\tilde{Q_1}$ (resp. $\tilde{Q_2}$) in checkpoint accordance with $\tilde{U_1}$ (resp. $\tilde{U_2}$), $U_1' \xrightarrow{![S]} U_1''$, and for all U_2'' such that $U_2' \xrightarrow{\tau}^* U_2''$ we have $U_2'' \overset{?[S]}{\not\longrightarrow}$. Thus, for all U_2'' as above, we have $t \longmapsto^* (T_1, T_2) : \langle \tilde{U_1} \rangle \blacktriangleright U_1' \parallel \langle \tilde{U_2} \rangle \blacktriangleright U_2'' = t'$. No rule in Fig 11 allows the term t' to evolve, i.e. $t' \not\longmapsto$, because U_1' can only perform $![S]$ and rule [TS-COM] cannot be applied since $U_2'' \overset{?[S]}{\not\longrightarrow}$. Since $T_1 \dashv\| T_2$, by Def. 1 it must hold that $U_1' = \texttt{end}$. However, since U_1' is able to perform an action (as it holds that $U_1' \xrightarrow{![S]} U_1''$), we get that it cannot be an end type, i.e. $U_1' \neq \texttt{end}$, which is a contradiction.

We conclude with the session progress theorem.

Theorem 6. Let $C = (\bar{a}(x_1).P_1 \mid a(x_2).P_2)$ be a rollback safe collaboration. If $C \rightarrowtail^* C'$ then either $C' \rightarrowtail C''$ for some C'' or $C' \equiv (\nu s : C)(\langle \tilde{Q_1} \rangle \blacktriangleright \mathbf{0} \mid \langle \tilde{Q_2} \rangle \blacktriangleright \mathbf{0})$ for some $\tilde{Q_1}$ and $\tilde{Q_2}$.

Proof. The proof proceeds by contradiction. Suppose that C is rollback safe and $C \rightarrowtail^* C'$ with $C' \not\rightarrowtail$ and $C' \not\equiv (\nu s : C)(\langle \tilde{Q_1} \rangle \blacktriangleright \mathbf{0} \mid \langle \tilde{Q_2} \rangle \blacktriangleright \mathbf{0})$ for any Q_1 and Q_2. The only situations that prevents C' from progressing are $C' = \mathbb{C}[\texttt{roll\_error}]$ and $C' = \mathbb{C}[\texttt{com\_error}]$. However, from Theorems 4 and 5, respectively, we have $C' \neq \mathbb{C}[\texttt{roll\_error}]$ and $C' \neq \mathbb{C}[\texttt{com\_error}]$, which is a contradiction.

A.4 Multiparty session results

All notions and concepts of our rollback recovery approach smoothly extend to the multiparty case. As consequence, all properties in Sec. 5 still hold in the extended setting. Their proofs indeed follow the same structure of the binary case and only differ for the technicalities concerning the extended definitions.

Decidability result Theorem 1. Let $T_1, \dots, T_n$ be filled multiparty session types, checking if $\dashv\| (T_1, \dots, T_n)$ holds is decidable.

Proof. By definition of Compliance in Sec. 7, checking $\dashv\| (T_1, \dots, T_n)$ consists in checking that types $T_1', \dots, T_n'$ of each configuration $(T^1, \dots, T^n) : \prod_{h \in \{1,\dots,n\}} \langle \tilde{U_h'} \rangle \blacktriangleright$

T'_h such that $(T^1, \dots, T^n) : \prod_{h\in\{1,\dots,n\}}\langle T_h\rangle \blacktriangleright T_h \longmapsto^* (T^1, \dots, T^n) : \prod_{h\in\{1,\dots,n\}}\langle \tilde{U}'_h\rangle \blacktriangleright T'_h \not\longmapsto$ (i.e., type configurations that are reachable from the initial one and that cannot further evolve) are end types. Thus, to prove that the compliance check is decidable we have to show that the number of these reachable configurations is finite. Let us consider the transition system $TS = \langle \mathcal{S}, \mathcal{R}\rangle$ associated to the type configuration $t = (T^1, \dots, T^n) : \prod_{h\in\{1,\dots,n\}}\langle T_h\rangle \blacktriangleright T_h$ by the reduction semantics of types (Fig. 18): the set $\mathcal{S}$ of states corresponds to the set of type configurations reachable from t, i.e. $\mathcal{S} = \{\, t' \mid t \longmapsto^* t' \,\}$, while the set $\mathcal{R}$ of system transitions corresponds to set of the type reductions involving configurations in $\mathcal{S}$, i.e. $\mathcal{R} = \{(t', t'') \in \mathcal{S} \times \mathcal{S} \mid t' \longmapsto t''\}$. Hence, checking $\dashv\!\Vert\ (T_1, \dots, T_n)$ boils down to check the type configurations corresponding to the leaves (i.e., states without outgoing transitions) of TS. Specifically, given a leaf of TS corresponding to $t = (T^1, \dots, T^n) : \prod_{h\in\{1,\dots,n\}}\langle \tilde{V_h}'\rangle \blacktriangleright V_h$, we have to check if $V_1 = \dots = V_n = \texttt{end}$. The decidability of this check therefore depends on the finiteness of TS. This result is ensured by the fact that: : *(i)* backward reductions connect states of TS only to previously visited states of TS (Theorem 8), and *(ii)* our language of types (Fig. **??**) corresponds to a CCS-like process algebra without static operators (i.e., parallel and restriction operators) within recursion (see [29, Sec. 7.5]).

Reversibility results As in Appendix A.2, we will indicate with $C \overset{s}{\rightarrowtail} C'$ the fact that the reduction is taking place on session s.

Lemma 10 (Swap Lemma). *Let C be a collaboration and s and r two sessions. If $C \overset{s}{\rightarrowtail} C_1 \overset{r}{\rightarrowtail} C_2$ then there exists a collaboration C_3 such that $C \overset{r}{\rightarrowtail} C_3 \overset{s}{\rightarrowtail} C_2$.*

Proof. By case analysis on the reductions $\overset{s}{\rightarrowtail}$ and $\overset{r}{\rightarrowtail}$.

Lemma 11. *Let C be a collaboration. If $C \rightarrowtail^* C_1$, then for any session s in C_1 there exists a collaboration C_0 such that $C \rightarrowtail^* C_0 \overset{s}{\rightarrowtail}{}^* C_1$ and s is never used in the trace $C \rightarrowtail^* C_0$.*

Proof. By induction on the number n of reduction on s. If there are no reductions then the thesis banally holds. Otherwise we can take the very last reduction on s, that is the closest one to C_1 and iteratively apply Lemma 10 in order to bring it to the very end. Then we can conclude by induction on a trace with less occurrences of reductions on s.

As in Appendix A.2, thanks to Lemma 11 without loosing of generality we can focus just on a sigle session, say s, and to consider collaboration initial for s.

Lemma 12. *Let C be an initial collaboration such that $C \rightarrowtail^* C_1$. If $C_1 \overset{abt}{\rightsquigarrow} C_2$ then $C_2 \equiv C$.*

Proof. Since C is *initial*, without losing of generality we can assume

$$C \equiv \bar{a}[n](x).P_n \ \mid\ \prod_{i\in I} a[i](x).P_i$$

with $I = \{1, .., n-1\}$.

The first reduction of $C \rightarrowtail^* C_1$ has to be an application of rule [M-F-Con], that is

$$C \twoheadrightarrow (\nu s\colon (\bar{a}[n](x).P_n \mid \prod_{i\in I} a[i](x).P_i))$$
$$(\langle P_n[s[n]/x]\rangle \blacktriangleright P_n[s[n]/x] \,|\, \prod_{i\in I}\langle P_i[s[i]/x]\rangle \blacktriangleright P_i[s[i]/x]) = C'$$

and, by hypothesis, $C' \rightarrowtail^* C_1$.

Now, no matter the shape of processes in C_1 by applying rule [M-B-Abt], and possibly [M-B-Str], we will go back to C, that is $C_1 \overset{abt}{\rightsquigarrow} C$, as desired.

Lemma 13. *Let C be a reachable collaboration, such that* $C \overset{cmt}{\twoheadrightarrow} C_1$*. If* $C_1 \twoheadrightarrow^* C_2 \overset{roll}{\rightsquigarrow} C_3$ *and there is no commit in* $C_1 \twoheadrightarrow^* C_2$*, then* $C_3 \equiv C_1$*.*

Proof. As in Lemma 6's proof, by hypothesis, there is no commits in $C_1 \twoheadrightarrow^* C_2$, and this implies that the log part of the C_1 will never change. Hence, by applying [M-B-Rll] and [M-B-Par] to C_2we can conclude.

Theorem 8. *Let* C_0 *be an initial collaboration. If* $C_0 \rightarrowtail^* C_1$ *then* $C_0 \twoheadrightarrow^* C_1$*.*

Proof. By induction on the number n of backward reductions contained into $C_0 \rightarrowtail^* C_1$. The base case ($n = 0$) trivially holds. In the inductive case, let us take the backward reduction which is the nearest to C_0. That is:

$$C_0 \twoheadrightarrow^* C' \rightsquigarrow C'' \rightarrowtail^* C_1$$

Depending whether it is an $\overset{abt}{\rightsquigarrow}$ or a $\overset{roll}{\rightsquigarrow}$ we can apply respectively Lemma 12 or Lemma 13 to obtain a forward trace of the form

$$C_0 \twoheadrightarrow^* C'' \rightarrowtail^* C_1$$

and we can conclude by applying the inductive hypothesis on the obtained trace which contains less backward moves with respect to the original one.

Lemma 14. *Let* C_1 *and* C_2 *be two reachable collaborations.* $C_1 \rightsquigarrow C_2$ *then* $C_2 \twoheadrightarrow^* C_1$*.*

Proof. (Skecth) Since C_1 is a reachable collaboration, we have that there exists an initial collaboration C_0 such that $C_0 \rightarrowtail^* C_1$. By applying Theorem 8 we can rearrange the trace such that it contains just forward transitions as follows

$$C_0 \twoheadrightarrow^* C_1$$

If the backward reduction is obtained by applying [M-B-Abt], by Lemma 12 we have $C_2 \equiv C_0$, from which the thesis trivially follows. Instead, if the backward reduction is obtained by applying [M-B-Rll], we proceed by case analysis depending on the presence of commit reductions in the trace. If they are present, we select the last of such commit, that is we can decompose the trace in the following way:

$$C_0 \twoheadrightarrow^* \overset{cmt}{\twoheadrightarrow} C_{cmt} \twoheadrightarrow^* C_1 \overset{roll}{\rightsquigarrow} C_2$$

and by applying Lemma 13 we have that $C_2 \twoheadrightarrow^* \equiv C_1$ as desired. In the case there is no commit in the trace, we can conclude by noticing that $C_0 \twoheadrightarrow C_2$ must be the first reduction in $C_0 \twoheadrightarrow^* C_1$.

Lemma 15. *Let C be a reachable collaboration. If $C \overset{roll}{\rightsquigarrow} C'$ and $C \overset{roll}{\rightsquigarrow} C''$ then $C' \equiv C''$.*

Proof. (Skecth) Since C is a reachable collaboration, it is has been generated by an initial collaboration C_0, and by Theorem 8 we have that $C_0 \twoheadrightarrow^* C$. We distinguish two cases, whether in the trace there has been at least one commit or not. In the first case, we can decompose the trace in such a way to single out the last commit as follows:

$$C_0 \twoheadrightarrow^* C_{cmt} \twoheadrightarrow^* C$$

so that in the reduction $C_{cmt} \twoheadrightarrow^* C$ there is no commit. If from C the rollbacks $C \overset{roll}{\rightsquigarrow} C'$ and $C \overset{roll}{\rightsquigarrow} C''$ are triggered by the same process, the thesis trivially follows. In the other case, we have that at least two processes, say P and Q, are able to trigger a rollback. If the roll action is executed by P we have that $C_{cmt} \overset{roll}{\rightsquigarrow} C'$. If the roll is triggered by Q we have that $C_{cmt} \overset{roll}{\rightsquigarrow} C''$. And we can conclude by noticing that $C' \equiv C''$, as desired.

Theorem 3. Let C be a reachable collaboration. If $C \overset{cmt}{\twoheadrightarrow} C'$ then there exists no C'' such that $C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$ and $C'' \twoheadrightarrow^+ C$.

Proof. We proceed by contradiction. Suppose that there exists C'' such that $C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$ and $C'' \twoheadrightarrow^+ C$. Since C is a reachable collaboration, thanks to Theorem 8 we have that there exists an initial collaboration C_0 such that $C_0 \twoheadrightarrow^* C \overset{cmt}{\twoheadrightarrow} C'$. Since a rollback brings back the collaboration to a point before a commit, this means it has been restored a commit previous to the last one (by [M-B-RLL], indeed, only processes stored in logs can be committed). This implies that there exist at least two different commits in the trace such that

$$C_0 \twoheadrightarrow^* C_{cmt} \twoheadrightarrow^* C \overset{cmt}{\twoheadrightarrow} C' \twoheadrightarrow^* \overset{roll}{\rightsquigarrow} C''$$

with $C'' = C_{cmt}$. We have that $C'' \twoheadrightarrow^+ C \overset{cmt}{\twoheadrightarrow} C' \twoheadrightarrow^* C_{rl}$, where C_{rl} is the collaboration that perform the roll reduction. Now since the last commit has been done by C, supposing that the commit is triggered by P^c, which evolves to $P^{c\prime}$ in doing that, we have that:

$$C \equiv (\nu s : C_0)(\langle P \rangle \blacktriangleright P^c \mid \prod_{i \in I} \langle Q_i \rangle \blacktriangleright Q_i^c)$$

and

$$C' = (\nu s : C_0)(\langle P^{c\prime} \rangle \blacktriangleright P^{c\prime} \mid \prod_{i \in I} \langle Q_i^c \rangle \blacktriangleright Q_i^c)$$

Now, by hypothesis we have that $C' \twoheadrightarrow^* C_{rl}$ without any commit being present in the trace, hence:

$$C_{rl} \equiv (\nu s : C_0)(\langle P^{c\prime} \rangle \blacktriangleright P^{rl} \mid \prod_{i \in I} \langle Q_i^c \rangle \blacktriangleright Q_i^{rl})$$

By hypothesis, from C_{rl} a rollback is possible. Regardless the rollback is triggered by P_{rl} or Q_{rl}, we have that $C_{rl} \overset{roll}{\rightsquigarrow} C'$. Now, from C' we cannot reach C ($C' \not\twoheadrightarrow^{+} C$), as C' is derived from C and the rollback can only bring the collaboration back to C'. This violates the hypothesis, and hence we conclude.

Soundness results

Lemma 16. *Let* $C = (\nu s : C') \prod_{i \in I} \langle \tilde{Q_i} \rangle \blacktriangleright P_i$ *be a reachable collaboration, with* $I = \{1, .., n\}$, $C' = (\bar{a}[n](x).R_n \mid \prod_{i \in I - \{n\}} a[i](x).R_i)$*, and for all* $i \in I$ $(\varnothing; \varnothing \vdash R_i \blacktriangleright x : T_i)$, $\dashv\!\Vert\, (T_1, \ldots, T_n)$, $(\Theta_i; \Gamma_i \vdash Q_i[x/s[i]] \blacktriangleright x : V_i)$, $(\Theta'_i; \Gamma'_i \vdash P_i[x/s[i]] \blacktriangleright x : U_i)$*. If* $C \rightarrowtail (\nu s : C') \prod_{i \in I} \langle \tilde{Q'_i} \rangle \blacktriangleright P'_i$ *then there exist* V'_i *and* U'_i *such that* $(T^1, \ldots, T^n) : \prod_{h \in I} \langle \tilde{V_h} \rangle \blacktriangleright U_h \longmapsto (T^1, \ldots, T^n) : \prod_{h \in I} \langle \tilde{V'_h} \rangle \blacktriangleright U'_h$ *with* $\tilde{Q'_i}$ *in checkpoint accordance with* $\tilde{V'_h}$*, and for all* $i \in I$ $(\hat{\Theta}_i; \hat{\Gamma}_i \vdash Q'_i[x/s[i]] \blacktriangleright x : V'_i)$, $(\hat{\Theta}'_i; \hat{\Gamma}'_i \vdash P'_i[x/s[i]] \blacktriangleright x : U'_i)$.

Proof. (Skecth) We have two cases depending whether the reduction $\rightarrowtail$ has forward or backward direction.

($\rightarrowtail = \twoheadrightarrow$). From rule [M-F-Res], we have $\prod_{i \in I} \langle \tilde{Q_i} \rangle \blacktriangleright P_i \twoheadrightarrow \prod_{i \in I} \langle \tilde{Q'_i} \rangle \blacktriangleright P'_i$. We prove the result by case analysis on the last rule applied in the inference of the above reduction.

($\rightarrowtail = \rightsquigarrow$). From rule [B-Res], we have $\prod_{i \in I} \langle \tilde{Q_i} \rangle \blacktriangleright P_i \rightsquigarrow \prod_{i \in I} \langle \tilde{Q'_i} \rangle \blacktriangleright P'_i$. We prove the result by case analysis on the last rule applied in the inference of the above reduction.

Lemma 17. *Let* $C = \bar{a}[n](x).P_n \mid \prod_{i \in I - \{n\}} a[i](x).P_i$*, with* $I = \{1, .., n\}$*, such that* $C \blacktriangleright \{\bar{a}[\mathtt{n}] : V_n, \ldots, a[\mathtt{1}] : V_1\}$, $T_i = V_i \cdot \mathtt{i}$ *for all* $i \in I$*, and* $\dashv\!\Vert\, (T_n, \ldots, T_1)$*. If* $C \rightarrowtail^{*} (\nu s : C)(\langle \underline{R_1} \rangle \blacktriangleright Q_1 \mid \prod_{i \in \{2, .., n\}} \langle \tilde{R_i} \rangle \blacktriangleright Q_i)$ *and* $Q_1 \xrightarrow{roll} Q'_1$*, then there exist* $\{U_i\}_{i \in I}$, $\{U'_i\}_{i \in I}$ *and* U''_1 *such that* $(T_1, \ldots, T_n) : \prod_{i \in I} \langle T_i \rangle \blacktriangleright T_i \longmapsto^{*} (T_1, \ldots, T_n) : \langle \underline{U_1} \rangle \blacktriangleright U'_1 \parallel \prod_{i \in \{2, .., n\}} \langle \tilde{U_i} \rangle \blacktriangleright U'_i$ *and* $U'_1 \xrightarrow{\mathtt{roll}} U''_1$.

Proof. (Sketch) As in Lemma 8's proof, this result can be obtained by repeatedly applying Lemma 16.

Lemma 18. *Let Let* $C = \bar{a}[n](x).P_n \mid \prod_{i \in I - \{n\}} a[i](x).P_i$*, with* $I = \{1, .., n\}$*, such that* $C \blacktriangleright \{\bar{a}[\mathtt{n}] : V_n, \ldots, a[\mathtt{1}] : V_1\}$, $T_i = V_i \cdot \mathtt{i}$ *for all* $i \in I$*, and* $\dashv\!\Vert\, (T_n, \ldots, T_1)$*. If* $C \rightarrowtail^{*} (\nu s : C) \prod_{i \in I} \langle \tilde{Q_i} \rangle \blacktriangleright P_i$, $P_1 \xrightarrow{\ell} P'_1$ *and there is no* P_j *with* $j \in \{2, .., n\}$ *such that* $P_j \Downarrow_{\bar{\ell}}$ *with* ℓ *of the form* $s[\mathtt{p}][\mathtt{q}]!\langle v \rangle$, $s[\mathtt{p}][\mathtt{q}]?(x)$, $s[\mathtt{p}][\mathtt{q}] \lhd l$ *or* $s[\mathtt{p}][\mathtt{q}] \rhd l$*, then there exist* $\{U_i\}_{i \in I}$, $\{U'_i\}_{i \in I}$ *and* U''_1 *such that* $(T_1, \ldots, T_n) : \prod_{i \in I} \langle T_i \rangle \blacktriangleright T_i \longmapsto^{*} (T_1, \ldots, T_n) : \prod_{i \in I} \langle \tilde{U_i} \rangle \blacktriangleright U'_i$ *with* $\tilde{Q}_i$ *in checkpoint accordance with* $\tilde{U}_i$, $U'_1 \xrightarrow{tl_{\Gamma}(\ell)} U''_1$*, with* Γ *sorting for typing* P'_1*, and for all* U''_i *such that* $U'_i \xrightarrow{\tau}^{*} U''_i$ *we have* $U''_i \overset{\overline{tl_{\Gamma}(\ell)}}{\not\longrightarrow}$.

Proof. (Skecth) As in Lemma 9's proof, this result can be obtained by repeatedly applying Lemma 16 and reasoning by case analysis on the rule for deriving the transition $P_1 \xrightarrow{\ell} P'_1$.

Theorem 9. *If C is a rollback safe collaboration, then $C \not\rightarrowtail^* \mathbb{C}[\mathit{roll\_error}]$.*

Proof. (Skecth) The proof proceeds by contradiction and relies on Lemma 17.

Theorem 10. *If C is a rollback safe collaboration, then $C \not\rightarrowtail^* \mathbb{C}[\mathit{com\_error}]$.*

Proof. (Skecth) The proof proceeds by contradiction and relies on Lemma 18.

Theorem 7. Let $C = (\bar{a}[n](x).P_n \mid \prod_{i \in \{1,..,n-1\}} a[i](x).P_i)$ be a r-safe collaboration. If $C \rightarrowtail^* C'$ then either $C' \rightarrowtail C''$ for some C'' or $C' \equiv (\nu s : C) \prod_{i \in \{1,..,n\}} \langle \tilde{Q}_i \rangle \blacktriangleright \mathbf{0}$ for some $\tilde{Q}_1, \ldots, \tilde{Q}_n$.

Proof. The proof proceeds by contradiction. Suppose that C is rollback safe and $C \rightarrowtail^* C'$ with $C' \not\rightarrowtail$ and $C' \not\equiv (\nu s : C) \prod_{i \in \{1,..,n\}} \langle \tilde{Q}_i \rangle \blacktriangleright \mathbf{0}$ for any $Q_1, \ldots, Q_n$. The only situations that prevents C' from progressing are $C' = \mathbb{C}[\texttt{roll\_error}]$ and $C' = \mathbb{C}[\texttt{com\_error}]$. However, from Theorems 9 and 10, respectively, we have $C' \neq \mathbb{C}[\texttt{roll\_error}]$ and $C' \neq \mathbb{C}[\texttt{com\_error}]$, which is a contradiction.