Multiparty Reactive Sessions

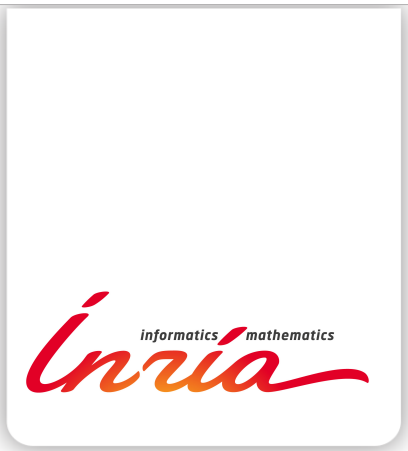
Mauricio Cano, Ilaria Castellani, Cinzia Di Giusto, Jorge A. Pérez

Multiparty Reactive Sessions

Mauricio Cano , Ilaria Castellani , Cinzia Di Giusto , Jorge A. Pérez

RESEARCH REPORT
N° 9270
April 2019
Project-Team INDES

ISSN 0249-6399 ISRN INRIA/RR--9270--FR+ENG



Multiparty Reactive Sessions

Mauricio Cano *, Ilaria Castellani †, Cinzia Di Giusto ‡, Jorge A. Pérez §

Project-Team INDES

Research Report n° 9270 — April 2019 — 62 pages

Abstract: Ensuring that communication-centric systems interact according to an intended protocol is an important but difficult problem, particularly for systems with some *reactive* or *timed* components. To rise to this challenge, we study the integration of session-based concurrency and Synchronous Reactive Programming (SRP). We propose a process calculus for multiparty sessions enriched with features from SRP. In this calculus, protocol participants may broadcast messages, suspend themselves while waiting for a message, and also react to events. Our main contribution is a session type system for this calculus, which enforces session correctness in terms of communication safety and protocol fidelity, and ensures two time-related properties that we call *output persistence* and *input timeliness*. Our type system departs significantly from existing ones, specifically as it captures the notion of logical instant typical of SRP.

Key-words: multiparty sessions, session types, global types, synchronous reactive programming

* University of Groningen, The Netherlands
† Université Côte d'Azur, INRIA, France
‡ Université Côte d'Azur, CNRS, I3S, France
§ University of Groningen, The Netherlands

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Sessions multi-parties réactives

Résumé : Assurer que les systèmes centrés sur la communication interagissent en accord avec un protocole donné est un problème important et difficile à résoudre, en particulier lorsque certains composants de ces systèmes sont *réactifs* ou *temporisés*. Pour relever ce défi, nous étudions l'intégration de primitives de la programmation réactive synchrone (PRS) dans les calculs de sessions. Nous proposons un calcul de sessions multi-parties enrichi avec des fonctionnalités typiques de la PRS. Dans ce calcul, les participants d'une session peuvent diffuser des messages, se suspendre dans l'attente de messages, et également réagir à des événements. Notre contribution principale est un système de types pour ce calcul, qui garantit deux propriétés classiques des calculs de sessions: l'absence d'erreurs de communication et la conformité au protocole. De plus, ce système de types assure deux propriétés liées au temps, que nous appelons "persistance des outputs" et "gestion sans latence des inputs". Notre système de types se démarque de façon significative des systèmes de types de session existants, en particulier en ce qu'il rend compte de la notion d'instant logique qui est caractéristique de la PRS.

Mots-clés : sessions multi-parties, types de session, types globaux, programmation réactive synchrone

Contents

1 Introduction

We study the integration of *synchronous reactive programming* (SRP) [7, 23, 22, 8] and *session-based concurrency* [18, 19]. Our goal is to devise a uniform programming model for communication-centric systems in which some components are *reactive* and/or *timed*. Synchronous reactive programming is a well-established model rooted on a few features: *broadcast signals*, *logical instants*, and *event-based preemption*. This makes it an ideal vehicle for specifying and analysing reactive systems; programming languages based on SRP include Esterel [4, 28], Céu [29], and ReactiveML [23]. On the other hand, session-based concurrency is the model induced by *session types* [18, 19], a rich typing discipline for message-passing programs. Session types specify protocols by stipulating the sequence in which messages should be sent/received by participants along a channel.

The interplay of message-passing concurrency with time- and event-based requirements is very common. In many protocols, participants are subject to time-related constraints (e.g., "the request must be answered within n seconds"). Also, protocols may depend, in various ways, on events that trigger run-time adaptations (e.g., "react to a timeout by executing an alternative protocol"). As a concrete example, consider a *buyer-seller protocol* in which a smart fridge manages groceries on behalf of a buyer, and only interacts with a supermarket in reaction to some event (say, "running out of milk"). Another example is an *electronic auction*, where an auctioneer offers a good for sale and buyers compete for this good by bidding the price upward. Here, the auctioneer supervises the bidding and decrees the knock-down price as soon as a standstill is reached. Like a physical auction, the electronic auction follows a multiparty protocol in which messages are broadcast to all participants, but they are "fetched" only by some of them. Bidders must be able to react in real time to the offers issued by other bidders and to the auctioneer's decisions. These two examples are representative of a wide class of scenarios requiring both:

- the ability of broadcasting messages that are not fetched by all participants ("orphan messages" become the norm rather than the exception) and
- a synchronous preemption mechanism, allowing participants' behaviours to be simultaneously reset in reaction to some event.

Unfortunately, existing frameworks based on session types lack these two key features—they are not expressive enough to model reactive and time-dependent interactions, essential in the two examples above. The framework in [21] handles contextual information through events, but does not support reactive behaviour nor multiparty protocols. Models such as [6, 3, 5] account for multiparty protocols with time-related conditions, but do not support reactive and event-based behaviours. The work [10] integrates SRP and session-based concurrency, but it is restricted to binary session types (protocols with two participants).

To overcome the limitations of existing approaches, we propose a new typed framework for multiparty protocols, expressive enough to support reactive, structured communications. Our framework builds on a new process language dubbed MRS (Multiparty Reactive Sessions), which combines constructs from (session) π-calculi with typical features of synchronous reactive languages, namely:

- *Logical instants*, or simply *instants*, which are periods in which all components compute until they cannot evolve anymore (instants are what make SRP "synchronous");
- Broadcast communication (instead of point-to-point communication);
- A "pause" construct, which suspends execution for the current instant;
- A "watch" construct implementing preemption, which is equipped with a standard and an alternative behaviour that is triggered in reaction to a given event. This construct generalises the exception mechanism provided by many programming languages, endowing it with a notion of time.
- Event emission, which is used here simply to control the watch construct.

The operational semantics of MRS is given in a style that is typical of synchronous languages. A process resides within a *configuration* with its memory and emitted events. There are two reduction relations on configurations: the first one formalises small-step execution *within an instant*, until the configuration *converges*, namely *suspends* or terminates. Suspension occurs when all participants have exercised their right to send/receive for the current instant, or have reached a "pause" instruction. The second reduction relation formalises how a suspended configuration evolves *across different instants*.

In more detail, during each instant, every participant can broadcast at most once and receive at most once from the same sender. This is a sensible requirement to discipline interaction in a reactive setting with valued messages. Indeed, allowing participants to broadcast multiple messages (valued events) in the same instant would amount to collect all their values at the end of the instant; then, an additional mechanism would be required to handle these "flattened" values and dispatch them in the expected order to the receivers.

Our semantics for MRS satisfies *(bounded) reactivity*, a standard soundness property of SRP which requires that small-step execution converges to a suspension or termination point at every instant [23]. This property, also called *instantaneous convergence* or *instant termination* in the SRP literature [30], is key for a reactive computation to evolve through a succession of instants and thus proceed as expected.

In session-based concurrency, protocol conformance typically follows from safety and liveness properties that stipulate how processes adhere to their session types, namely: session fidelity, communication safety, and some progress/deadlock-freedom property. In MRS, we further target the following two *time-related properties*:

P1. *Output persistence*: Every participant broadcasts exactly once during every instant;

P2. *Input timeliness*: Every unguarded input is matched by an output during the current instant, if not preceded by another input with equal source and target, or during the next instant, if not preempted.

Our main contribution is a type system, based on multiparty session types [19], that enforces session fidelity, communication safety, as well as output persistence and input timeliness for MRS processes. One crucial technical challenge consists in properly handling *explicit* and *implicit* pauses in MRS processes. Explicit (or *syntactic*) pauses correspond to occurrences of the pause construct in processes. In contrast, implicit (or *semantic*) pauses are those induced by the synchronous reactive semantics between two broadcasts by the same participant, or between two inputs by the same participant from the same source.

Our type system relies on the usual ingredients of multiparty session types: *global types* entirely describe a multiparty protocol; *local types* stipulate the protocol associated to each participant; a *projection function* relates global and local types. However, because of the interplay between sessions and SRP, these ingredients have rather different definitions in our framework. In particular, we require a new pre-processing phase over global types called *saturation*, which complements protocols with implicit pauses. Unique to our setting, saturation is essential to conduct our static analysis on MRS processes and, ultimately, to reduce the conceptual gap between SRP and session-based concurrency.

The rest of the paper is organised as follows.

- Section 2 illustrates our approach by means of the auction example discussed above;
- Section 3 introduces the syntax and semantics of MRS. We prove that our model is *reactive*, namely that every reachable configuration instantaneously converges in a number of steps that is bounded by the size of the process (Theorem 2).
- Section 4 presents our type system and proves that it ensures the correctness and time-related properties mentioned above;
- Section 6 discusses further related work and Section 7 concludes by explaining some of our design choices and sketching some directions for future work.

2 Two Motivating Examples

We illustrate our typed process model by formalising the two examples mentioned in Section 1. We first present a reactive variant of the well-known *Buyer-Seller protocol*; then, we model the *Electronic Auction* protocol.

2.1 A Reactive Buyer-Seller Protocol

Consider a scenario involving three participants: a *Smart Fridge* (F), a *Client* (C), and a *Supermarket* (S). These participants interact with the following goal: F acts on behalf of C to purchase groceries from S. Being a smart, autonomous agent, F should react whenever a low level of groceries is detected, and initiate a protocol with S and C so as to restore a predefined level of groceries. F should obtain authorization from C before issuing a purchase order to S.

Before formalizing this protocol as a global type with reactive constructs, we introduce global types informally (a formal description shall be given in Section 4):

- $\mathtt{p}\uparrow\langle S,\Pi\rangle.G$ denotes a global type in which participant p broadcasts a message of sort S[1] which will be fetched by the participants in set Π; after that, the protocol continues as specified by G.
- $\mu\mathbf{t}.G$ represents a recursive protocol given by G, which includes occurrences of variable $\mathbf{t}$.
- Given event *ev*, we introduce watch *ev* do G_1 else G_2 as a reactive, event-dependent global type. This type says that protocol G_1 will be executed until termination or suspension. When G_1 suspends there are two possibilities: if *ev* has not occurred, then the remainder of G_1 is invoked again as the governing protocol in the next instant; otherwise, as a reaction to the occurrence of *ev*, the protocol G_1 is discarded and G_2 is invoked in the next instant.
- pause.G is also peculiar to our reactive, timed setting: it says that all participants should move to the next instant to execute protocol G.
- end represents the terminated protocol, as usual.

We then have the following global type G, which describes the multiparty protocol between the fridge, the client, and the supermarket. We use two events, named *lf* (for *low food*) and *ok*, which stands for a *confirmation* event:

$$
\begin{aligned}
G = \mu\mathbf{t}_1.\mathtt{watch}\ \mathit{lf}\ \mathtt{do}\ \mu\mathbf{t}_2.&\mathtt{S}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{F}\uparrow\langle \mathit{stat},\{\mathtt{C}\}\rangle.\mathtt{C}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{pause}.\mathbf{t}_2\\
&\mathtt{else}\\
&\quad \mathtt{S}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{F}\uparrow\langle \mathit{lst},\{\mathtt{C}\}\rangle.\mathtt{C}\uparrow\langle \mathit{lst},\{\mathtt{F}\}\rangle.\\
&\quad \mathtt{watch}\ \mathit{ok}\ \mathtt{do}\ \mathtt{pause}.\mu\mathbf{t}_3.\mathtt{S}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{F}\uparrow\langle \mathit{stat},\{\mathtt{C}\}\rangle.\mathtt{C}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{pause}.\mathbf{t}_3\\
&\quad \mathtt{else}\\
&\quad\quad \mathtt{C}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{F}\uparrow\langle \mathit{lst},\{\mathtt{S}\}\rangle.\mathtt{S}\uparrow\langle \mathit{prc},\{\mathtt{F}\}\rangle.\mathtt{pause}.\\
&\quad\quad \mathtt{C}\uparrow\langle \mathit{stat},\{\mathtt{F}\}\rangle.\mathtt{F}\uparrow\langle \mathit{cc},\{\mathtt{S}\}\rangle.\mathtt{S}\uparrow\langle \mathit{iv},\{\mathtt{F}\}\rangle.\mathtt{pause}.\mathbf{t}_1
\end{aligned}
$$

To describe the protocol specified by G, we spell out the instants it involves. We assume that *lf* is emitted at time t_{lf} and that *ok* is emitted at time t_{ok}.

[1] Basic types are called "sorts" here, following the terminology introduced by Milner [24] and widely adopted in the session type literature.

Instant $t \leq t_{lf}$**:** the protocol enters the 'do' branch of the outermost watch, which is guarded by event *lf*, indicating low food levels in the fridge, and it starts executing a loop. In the body of this loop, S sends a status update (*stat*) to F indicating its availability for future purchases; next, F sends an update to C with information about the current items in the fridge (say, expiration dates), and C answer by updating his own status; finally, a pause is reached and the presence of event *lf* is checked:

1. If *lf* has not yet been emitted, then the 'do' branch is executed again in the next instant.
2. If *lf* is present, then the 'else' branch will be executed in the next instant.

Instant $t_{lf} + 1$**:** now the 'else' branch of the outermost watch is executed: once again, S updates his status; then, F sends to C a list with grocery items to buy (*lst*) and C can update the list with more items. Finally, the 'do' branch of the innermost watch is executed, guarded by event *ok*, indicating a confirmation. The pause makes the protocol move to the next instant, ensuring that the event *ok* may be immediately accounted for. If *ok* has not been emitted, then S, F and C keep issuing their status. Upon the emission of event *ok*, the innermost watch is exited and its 'else' branch is selected for the next instant.

Instant $t_{ok} + 1$**:** now the 'else' branch of the innermost watch is executed: C updates his status, F orders the groceries from S and gets back their price. Then the protocol moves to the next instant.

Instant $t_{ok} + 2$**:** once again, C updates his status, F sends to S the information required to complete the payment (e.g., credit card number), and S sends back the invoice *iv* to F.

As usual for multiparty session types, the global type G should be projected into local types for F, S, and C, which will be used to type-check against process implementations. In MRS, the local types are as follows:

- Local types $!S.T$ and $?(\mathtt{p}, S).T$, represent output and input. In the former type, a value of sort S is broadcast and then the type continues as T. In the latter, a value of sort S is received from participant p; the type then continues as T.
- Local types pause and $\langle T_1, T_2 \rangle^{ev}$ specify the reactive behaviour of participants, and they are similar to the corresponding constructs for global types.
- Type $\mu\mathbf{t}.T$ represents a recursive type, where $\mathbf{t}$ may occur in T.

We write $G\lfloor_{\mathtt{p}}$ to denote the local type obtained from the projection of global type G into participant p. This way, e.g., we have the following local type for C:

$$G\lfloor_{\mathtt{C}} = \mu\mathbf{t}_1.\langle \mu\mathbf{t}_2.?(\mathtt{F}, stat).!stat.\mathtt{pause}.\mathbf{t}_2, ?(\mathtt{F}, lst).!lst.\langle \mathtt{pause}.\mu\mathbf{t}_3?(\mathtt{F}, stat).!stat.\mathtt{pause}.\mathbf{t}_3, !stat.\mathbf{t}_1\rangle^{ok}\rangle^{lf}$$

In Fig. 1, we show a process implementation of our protocol; it allows us to introduce some salient constructs in MRS:

- Process $s[\mathtt{p}]!\langle v\rangle.P$ represents the broadcast of value v from participant p along session s; also, $s[\mathtt{p}]?(\mathtt{q}, x).P$ represents p receiving a message coming from q along session s.
- Process $\mathtt{pause}.P$ suspends for the current instant, and executes P in the next instant.
- Process $\mathtt{emit}\ ev.P$ emits an event ev, visible by all participants during the current instant, and then continues as P within the same instant.

$$
\begin{array}{l}
\mathit{System} = \mathit{Fridge} \mid \mathit{SMarket} \mid \mathit{Client} \\
\mathit{Fridge} = \mathtt{rec}\ X_1\,.\,\mathtt{watch}\ \mathit{lf}\ \mathtt{do}\ \mathtt{rec}\ X_2\,.\,s[\mathtt{F}]?(\mathtt{S}, x_1).s[\mathtt{F}]!\langle \mathit{stat}\rangle.s[\mathtt{F}]?(\mathtt{C}, x_2). \\
\qquad\qquad\qquad\qquad \mathtt{if}\ (\mathit{status}(\mathit{food}) = 0)\ \mathtt{then}\ \mathtt{emit}\ \mathit{lf}.\ \mathtt{pause}.\ X_2\ \mathtt{else}\ \mathtt{pause}.\ X_2 \\
\qquad\qquad \{s[\mathtt{F}]?(s[\mathtt{S}], x_3).s[\mathtt{F}]!\langle \widetilde{\mathit{food}}\rangle.s[\mathtt{F}]?(\mathtt{C}, x_4). \\
\qquad\qquad\quad \mathtt{watch}\ \mathit{ok}\ \mathtt{do}\ \mathtt{pause}.\ \mathtt{rec}\ X_3\,.\,s[\mathtt{F}]?(\mathtt{S}, x_5).s[\mathtt{F}]!\langle \mathit{stat}\rangle.s[\mathtt{F}]?(\mathtt{C}, x_5).X_3 \\
\qquad\qquad\quad \{s[\mathtt{F}]?(\mathtt{C}, x_6).s[\mathtt{F}]!\langle \widetilde{x_7}\rangle.s[\mathtt{F}]?(s[\mathtt{S}, x_8]).\mathtt{pause}.\ s[\mathtt{F}]?(\mathtt{C}, x_9).s[\mathtt{F}]!\langle \mathit{cc}\rangle.s[\mathtt{F}]?(\mathtt{S}, x_{10}).X_1\}\} \\
\mathit{SMarket} = \mathtt{rec}\ Y_1\,.\,\mathtt{watch}\ \mathit{lf}\ \mathtt{do}\ \mathtt{rec}\ Y_2\,.\,s[\mathtt{S}]!\langle \mathit{stat}\rangle.Y_2\{\mathtt{watch}\ \mathit{ok}\ \mathtt{do}\ \mathtt{pause}.\ \mathtt{rec}\ Y_3\,.\,s[\mathtt{S}]!\langle \mathit{stat}\rangle.Y_3 \\
\qquad\qquad \{s[\mathtt{S}]?(\mathtt{F}, y_1).s[\mathtt{S}]!\langle \mathit{price}(y_1)\rangle.\mathtt{pause}.\ s[\mathtt{S}]?(\mathtt{F}, y_2).s[\mathtt{S}]!\langle \mathit{invoice}\rangle.Y_1\}\} \\
\mathit{Client} = \mathtt{rec}\ Z_1\,.\,\mathtt{watch}\ \mathit{lf}\ \mathtt{do}\ \mathtt{rec}\ Z_2\,.\,s[\mathtt{C}]?(\mathtt{F}, z_2).s[\mathtt{C}]!\langle \mathit{stat}\rangle.Z_2 \\
\qquad\qquad \{s[\mathtt{C}]?(\mathtt{F}, z_4).s[\mathtt{C}]!\langle z_4\rangle. \\
\qquad\qquad\quad \mathtt{watch}\ \mathit{ok}\ \mathtt{do}\ \mathtt{emit}\ \mathit{ok}.\ \mathtt{pause}.\ \mathtt{rec}\ Z_3\,.\,s[\mathtt{C}]?(\mathtt{F}, z_6).s[\mathtt{C}]!\langle \mathit{stat}\rangle.Z_3\{s[\mathtt{C}]!\langle \mathit{stat}\rangle.Z_1\}\}
\end{array}
$$

Figure 1: An MRS implementation of the Reactive Buyer-Seller Protocol.

- Process $\mathtt{watch}\ \mathit{ev}\ \mathtt{do}\ P\{Q\}$ is defined in correspondence with the local type $\langle T_1, T_2\rangle^{ev}$. This process executes P up to termination or suspension. In the former case, the whole process disappears. When P evolves to P' and suspends, there are two possibilities at the end of the instant, depending on *ev*: if *ev* has not occurred then $\mathtt{watch}\ \mathit{ev}\ \mathtt{do}\ P'\{Q\}$ is executed in the next instant; otherwise, as a reaction to the occurrence of *ev*, $\mathtt{watch}\ \mathit{ev}\ \mathtt{do}\ P'\{Q\}$ is discarded and Q is executed in the next instant.

The protocol implementation in Fig. 1 is given by process *System*, which is composed of three parallel processes (*Fridge*, *SMarket*, and *Client*), implementing participants F, S, and C, respectively:

Process *Fridge* runs a recursive loop while there is enough food: it first receives a status update from S and broadcasts its own updates (to be received by *Client*); it also receives and update from *Client*. Then, it checks the current level of food: if there is enough food ($\mathit{status}(\mathit{food}) \neq 0$), then the status update loop is repeated. Otherwise, it emits *lf*, thus triggering the alternative behavior, which consists in sending the groceries list ($\widetilde{\mathit{food}}$) to *Client*, which in turn should answer with possible modifications to the list and then confirm the purchase by emitting *ok*. The status update loop will execute as long as *ok* has not occurred; when *Client* confirms, *Fridge* will first receive and status update from the client and then send the groceries list to *SMarket*, which will return the total price. Finally, another status update from the client is received and *Fridge* exchanges payment information and invoice with *SMarket*.

Process *SMarket* is engaged in the update loop with *Fridge* until *lf* is emitted. Once this event occurs, the update loop will continue until *ok* is detected. After confirmation, *SMarket* and *Fridge* interact to finalize the purchasing protocol.

Process *Client* is similar, and takes part in the status update loop until *Fridge* emits *lf*. When this occurs, and after having received the groceries list from *Fridge*, *Client* simply resends the list it received without modifications and confirms the purchase (clearly, more elaborate authorisation procedures are possible). Once *SMarket* and *Fridge* have completed the purchase, *Client* engages again into the status update loop.

$$
\begin{aligned}
&\mathit{Auction} = \mathit{Auctioneer} \mid \mathit{Bidder}_1 \mid \cdots \mid \mathit{Bidder}_n \\
&\mathit{Auctioneer} = \texttt{watch}\, \mathit{bis}\, \texttt{do}\, \Big(\texttt{rec}\, X(\widetilde{x}) \,.\, s[\mathtt{A}]?(\mathtt{B}_1, \mathit{bid}_i).\ \cdots\ .\, s[\mathtt{A}]?(\mathtt{B}_n, \mathit{bid}_n). \\
&\qquad\qquad \texttt{if}\ \bigwedge\nolimits_{i\in I}(\mathit{bid}_i = x_i)\ \texttt{then}\ \texttt{emit}\ \mathit{bis}.\, s[\mathtt{A}]!\langle\widetilde{\mathit{bid}}\rangle.X(\widetilde{\mathit{bid}}) \\
&\qquad\qquad \texttt{else}\ s[\mathtt{A}]!\langle\widetilde{\mathit{bid}}\rangle.X(\widetilde{\mathit{bid}})\Big)(\widetilde{\mathit{initPrice}}) \\
&\qquad\qquad \{s[\mathtt{A}]!\langle \mathit{max}(\tilde{\mathit{bid}})\rangle.\mathbf{0}\} \\
&\mathit{Bidder}_i = \texttt{watch}\, \mathit{bis}\, \texttt{do}\, \Big(\texttt{rec}\, Z_i(z_i) \,.\, s[\mathtt{B}_i]!\langle z_i\rangle.s[\mathtt{B}_i]?(\mathtt{A}, \tilde{w}). \\
&\qquad\qquad \texttt{if}\ (\mathit{max}(\tilde{w}) \neq z_i \ \wedge\ \mathit{max}(\tilde{w}) + \Delta_i \leq \mathit{budget}_i)\ \texttt{then}\ Z_i(\mathit{max}(\tilde{w}) + \Delta_i) \\
&\qquad\qquad \texttt{else}\ Z_i(z_i)\Big)(\mathit{initBid}_i) \\
&\qquad\qquad \{s[\mathtt{B}_i]?(\mathtt{A}, z_i').s[\mathtt{B}_i]!\langle \mathit{eog}\rangle.\mathbf{0}\}
\end{aligned}
$$

Figure 2: An MRS implementation of the Electronic Auction Protocol.

2.2 An Electronic Auction Protocol

We now formalise the electronic auction protocol sketched in Section 1. This example exhibits again the distinctive features of MRS: the slicing of computation into instants, broadcast communication, and the synchronous preemption mechanism. In addition, it illustrates the specific treatment of recursion in MRS, and it shows how parameterised recursion may be used to transmit values across instants.

Assuming n bidders ($n \geq 2$), the protocol has $n+1$ participants: participant $\mathtt{A}$, which is the *Auctioneer*, and participants $\mathtt{B}_1, \ldots, \mathtt{B}_n$, which are the *Bidders*. Bidding rounds are represented by instants: at the start of each instant, all the bidders send their new bids to the auctioneer, which responds by broadcasting the new tuple of bids, whose maximum represents the current price of the good. We suppose that the starting price of the good is the same for all bidders, i.e., $\mathit{initPrice}_i = \mathit{initPrice}_j$ for any $i, j \in \{1, \ldots, n\}$; we also suppose that each Bidder_i has a maximal budget budget_i and that her initial bid $\mathit{initBid}_i$ is such that $\mathit{initPrice}_i < \mathit{initBid}_i \leq \mathit{budget}_i$ and $\mathit{initBid}_i \neq \mathit{initBid}_j$ for $i \neq j$. Moreover, we assume that each Bidder_i bids up by a fixed amount Δ_i such that $\Delta_i \neq \Delta_j$ for $i \neq j$. The two above conditions $\mathit{initBid}_i \neq \mathit{initBid}_j$ and $\Delta_i \neq \Delta_j$ are used to prevent equal bids from different bidders, which would make the protocol more involved[2]. Then, a session s of the protocol may be described by the *Auction* process in Figure 2, where $\tilde{\sigma}$ represents the tuple $(\sigma_1, \ldots, \sigma_n)$ and for any such tuple $\tilde{\sigma}$, the function $\mathit{max}(\tilde{\sigma})$ is defined by $\mathit{max}(\tilde{\sigma}) = \mathit{max}\{\sigma_1, \ldots, \sigma_n\}$.

Note that *Auctioneer* and the Bidder_i have a similar structure: they consist of a watch statement guarded by the event *bis*, whose main branch executes a loop and whose alternative branch does an I/O action and terminates. The event *bis* (standing for "bis repetita") is emitted by *Auctioneer* to signal that the same tuple of bids has occurred twice and hence the auction is over.

Supposing event *bis* is emitted at instant t_{bis}, let us see how instants build up in our protocol.

Instant $t \leq t_{bis}$: At the beginning, all participants enter the main branch of the `watch` guarded by event *bis*, and they start executing a loop. In the body of their loop, all Bidder_i broadcast their new bids (which in the first iteration are just their initial bids $\mathit{initBid}_i$), and then wait for a new tuple of bids from *Auctioneer*. Now, *Auctioneer* inputs all the new bids from the Bidder_i and compares them with

[2] Since multiparty session protocols are usually deterministic, if equal bids were allowed there should be some predefined criterion to choose between them. Moreover, in a physical auction all bids must be different, since they are issued one after the other. The requirement $\Delta_i \neq \Delta_j$ allows this to be mimicked using simultaneous bids.

their previous bids (which in the first iteration of *Auctioneer* are just the $initPrice_i$). If the new bid from each $Bidder_i$ is equal to her previous bid, then a standstill is reached and *Auctioneer* emits the event *bis* to trigger the alternative behaviour of all participants at the next instant; then *Auctioneer* broadcasts the same tuple of bids (this broadcast is required to match the expectations of the $Bidder_i$). Otherwise, *Auctioneer* simply broadcasts the new tuple of bids. In both cases, *Auctioneer* suspends before starting the next iteration, because the semantic rule for recursion inserts a `pause` before the next occurrence of the recursive call. After the broadcast from *Auctioneer*, all $Bidder_i$ fetch the tuple of new bids and check that the maximum $max(\tilde{z})$ of this tuple (the best offer so far) is a bid different from their own, and that their budget allows them to bid up; if this is the case, they increment $max(\tilde{z})$ by Δ_i and suspend; otherwise, they issue again their previous bid z_i and suspend. The execution goes on similarly until the tuple of bids reaches a standstill (this is ensured by the fact that each $Bidder_i$ bids upwards by a fixed amount Δ_i, while not overriding $budget_i$), leading eventually to the emission of event *bis* by *Auctioneer*. At this point, all participants are deviated from their main behaviour and their alternative behaviour is triggered at the next instant.

Instant $t_{bis} + 1$: Now *Auctioneer* broadcasts the knock-down price, which is received by all the $Bidder_i$, who then react by sending an "end of game" message. Note that, because of our hypotheses, the knock-down price uniquely identifies the winner.

The global type for the protocol is as follows (see Section 4 for the local types):

$$G = \texttt{watch}\ bis\ \texttt{do}\ \mu \mathbf{t}.\mathtt{B}_1 \uparrow \langle \mathsf{int}, \{\mathtt{A}\}\rangle.\ \cdots\ .\mathtt{B}_n \uparrow \langle \mathsf{int}, \{\mathtt{A}\}\rangle.\mathtt{A} \uparrow \langle \widetilde{\mathsf{int}}, \{\mathtt{B}_1, \ldots, \mathtt{B}_n\}\rangle.\mathbf{t}$$
$$\texttt{else}\ \mathtt{A} \uparrow \langle \mathsf{int}, \{\mathtt{B}_1, \ldots, \mathtt{B}_n\}\rangle.\mathtt{B}_1 \uparrow \langle \mathsf{string}, \emptyset\rangle.\ \cdots\ .\mathtt{B}_n \uparrow \langle \mathsf{string}, \emptyset\rangle.\mathsf{end}$$

Summing Up. This example illustrates some distinctive features of our framework:

- Messages are *valued events* which are broadcast, hence they are not consumed when they are read; instead, they are consumed by the *passage of time*, since they are erased at the end of each instant;
- Our typed calculus imposes a strong common structure in protocol participants; while this may seem contrived, it is the source of the correctness properties enforced by our type system. For instance, bidders who wish to drop from the auction still have to issue their last bid until the end of the auction. This is because the bidders must match the expectations of the *Auctioneer*, who waits for a bid from all bidders at each instant since she cannot foresee at which point they will give up.
- In our calculus, as in most multiparty session frameworks, the set of participants is fixed and participants cannot dynamically enter or leave a session. It is possible however that some participants may terminate before the others, and the output persistence property is only required for nonterminated participants (although this does not appear in the above example).

Having illustrated informally MRS and its typed system, we now introduce them formally.

3 Our Process Model: MRS

We introduce MRS, our calculus of Multiparty Reactive Sessions. It integrates constructs from Synchronous Reactive Programming and from multiparty session π-calculi.

3.1 Syntax

We assume the following basic sets: *values* (booleans, integers), ranged over by v, v'; *value variables*, ranged over by x, y, z; and *expressions*, ranged over by e, e'. Expressions are built from variables and values via standard operators, and their evaluation is terminating. A set of *process variables* $X, Y, \ldots$, possibly parameterised by a tuple of parameters (written in this case as $X(\tilde{x})$ or $X(\tilde{e})$), is assumed to define recursive behaviours. We also use two sets that are specific to multiparty session calculi [19, 14]: *service names*, ranged over by a, b, each of which has an *arity* $n \geq 2$ (its number of participants), and *sessions*, denoted by s, s'. A session represents a particular activation of a service. We use $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$ to denote generic *(session) participants*. In an n-ary session, participants will often be assumed to range over the natural numbers $1, \ldots, n$ (in particular, we will use this assumption when defining the operational semantics). We denote by Π a non empty set of participants, and by Part_s the set of participants of session s. Finally, we assume a set of events *Events*, ranged over by ev, ev', which will be used for defining the reactive constructs.

Each n-ary session s has an associated set of *session channels* $\{s[1], \ldots, s[n]\}$, one per participant: channel $s[\mathtt{p}]$ is the private channel through which $\mathtt{p}$ communicates with other participants in session s. We also assume a set of *channel variables*, ranged over by α, β, γ; we use c to range over both channel variables and session channels.

The syntax of *processes*, ranged over by $P, Q \ldots$, is given in Fig. 4. A new session s on the n-ary service a is opened when the *initiator* $\bar{a}[n]$ synchronises with n processes of the form $a[i](\alpha_i).P_i$, whose channels α_i then get replaced by $s[i]$ in the body of P_i. (This synchronisation will be made precise by our operational semantics.) The initiator $\bar{a}[n]$ simply marks the presence of the service a, therefore it has no continuation behaviour. Processes of the form $a[i](\alpha_i).P_i$ are called "candidate participants" for service a.

Rather than typical point-to-point communication, we consider communication based on *broadcast* and *directed input*. The former is denoted $c!\langle e\rangle.P$: this is an "undirected output" on c, for it does not mention any intended recipient for message e. The latter, denoted $c?(\mathtt{p}, x).P$, represents the input of a message sent by $\mathtt{p}$. For simplicity we do not consider branching/selection operators here. Constructs for conditional expressions, parallel composition are standard. and have expected meanings.

With respect to usual calculi for multiparty sessions, the main novelty in MRS is the addition of three reactive constructs typical of synchronous languages, given on the bottom right of Fig. 4. They are:

- $\mathtt{pause}.P$, which postpones the execution of P to the next instant;
- $\mathtt{emit}\ ev.P$, which emits event ev in the current instant and then executes P;
- $\mathtt{watch}\ ev\ \mathtt{do}\ P\{Q\}$, a construct that we call "watch-and-replace". It executes P and, in the presence of event ev, it replaces whatever is left of P by Q at the end of the instant. P and Q are respectively the *main behaviour* and the *alternative behaviour* of the construct.

Our watching construct is slightly more general than similar constructs in synchronous languages. Without a sequential composition operator—not present in MRS (nor in most session calculi), but common in synchronous languages—this added generality is actually needed. Indeed, without sequential composition a watching statement cannot be followed by another statement; therefore, if we were to use the standard $\mathtt{watch}\ ev\ \mathtt{do}\ P$ construct, this would just lead to termination at the end of the instant in case ev is present.

For recursion we assume the standard *guardedness* condition, adapted to our language:

Definition 1 (Guardedness)**.** *A variable X is* guarded *in P if it only occurs in subprocesses $c!\langle e\rangle.Q$ or $c?(\mathtt{p}, x).Q$ or $\mathtt{emit}\ ev.Q$ or $\mathtt{pause}.Q$ of P.*

Note that the syntax of MRS is quite liberal. In particular, it allows processes with interleaved communications in different sessions, such as $s[1]!\langle e\rangle.s'[1]!\langle e'\rangle.P$. However, in the rest of this paper we focus on *processes without session interleaving*, and our technical treatment is developed only for them.

v	::=	$\mathsf{true} \mid \mathsf{false} \mid 1 \mid \ldots$		Value
e	::=	$x \mid v \mid \mathsf{not}\ e \mid e\ \mathsf{and}\ e' \mid \ldots \mid f(x_1, \ldots, x_n)$		Expression
u	::=	$a \mid s$		Service/Session Name
c	::=	$\alpha \mid s[\mathsf{p}]$		Channel variable/Session channel
Π	::=	$\{\mathsf{p}\} \mid \Pi \cup \{\mathsf{p}\}$		Set of participants
m	::=	$\epsilon \mid (v, \Pi)$		Message (with set of readers)
M	::=	$\emptyset \mid M \cup \{c : m\}$	$c \notin dom(M)$	Memory

Figure 3: MRS: Syntax of expressions, sessions, channels, messages and memories.

P	::=	$\bar{a}[n]$	Session initiator	|	`pause`$.P$	Pause
	|	$a[\mathsf{p}](\alpha).P$	Session participant	|	`emit` $ev.P$	Emit
	|	$c!\langle e\rangle.P$	Broadcast Output	|	`watch` ev `do` $P\{Q\}$	Watch & Replace
	|	$c?(\mathsf{p}, x).P$	Input			
	|	`if` e `then` P `else` Q	Conditional			
	|	$P \mid Q$	Parallel			
	|	$\mathbf{0}$	Inaction			
	|	$X(\tilde{e})$	Variable			
	|	$(\mathtt{rec}\ X(\tilde{x}).P)(\tilde{e})$	Recursion			

Figure 4: MRS: Syntax of processes.

The restriction to single sessions is standard in session calculi, as interleaving introduces inter-session dependencies that cannot be captured by session types. Moreover, interleaving would raise new issues in our setting, particularly as regards suspension of configurations involved in more than one session.

3.2 Semantics

We present now the semantics of MRS. In SRP, parallel components communicate via *broadcast events*, which may be either *valued*, if they carry some content, or *pure*. We call valued events *messages*, and pure events simply *events*. To model broadcast communication we assume that all processes share a *message set* or *memory* M, recording the messages exchanged in ongoing sessions during the current instant, and an *event set* E, recording the events emitted during the instant. Both sets are emptied at the end of each instant.

A memory M is a finite set of *named messages* $c : m$, where c is the name of the channel on which the message was sent, and m is the message content: this content may be either empty or of the form (v, Π), where v is the carried value and Π is the set of current *Readers* of the value. We will see in the next section why we need to record the set Π of readers for non-empty messages. A memory M may be viewed as a partial function from channels to messages, whose domain $dom(M) = \{c \mid \exists\, m\,.\, c : m \in M\}$ is finite.

The *session memory* M_s of session s has the form $\bigcup_{i \in \mathsf{Part}_s} \{s[i] : m_i\}$, where each $s[i] : m_i$ represents the (one-place) *output buffer* of participant i, which contains the empty message if participant i has not yet broadcast in the current instant, and a proper message otherwise. We shall use the following auxiliary

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{Q\} \equiv \mathbf{0}$$
$$P \equiv Q \;\Rightarrow\; \langle P, M, E\rangle \equiv \langle Q, M, E\rangle \qquad C \equiv C' \;\Rightarrow\; (vs)C \equiv (vs)C'$$

Figure 5: Structural congruence.

notations: $M_s^{\emptyset} = \bigcup_{i \in \mathsf{Part}_s} \{s[i] : \epsilon\}$ and $M^{\emptyset} = \{c : \epsilon \mid c \in dom(M)\}$. Fig. 3 (bottom) summarises the notation for memories.

The sets of *free names*, *bound names*, and *names* of a process P, denoted respectively by $fn(P)$, $bn(P)$, $nm(P)$, are defined as usual. Assuming Barendregt's convention, no bound name can occur free, and the same bound name cannot occur in two different bindings.

The semantics of MRS is defined on *configurations*. In its simplest form, a configuration is a triple $C = \langle P, M, E\rangle$, where P is a process, M is a memory, and E is a set of events. A configuration may also be restricted with respect to a session name s, namely have the form $(vs)\,C$. Our semantics will not be defined on arbitrary configurations, but only on those that may occur in the execution of a single session. Intuitively, these are the configurations that may be reached in zero or more steps from an initial configuration, as defined below (Def. 3). A formal definition of reachability will be given at the end of this section (Def. 4).

We first introduce the notion of sequential and session-closed process:

Definition 2 (Sequentiality and session-closedness)**.** *A process P is said to be* sequential *if it is built without the parallel construct $\mid$, and* session-closed *if it is built without the constructs $\bar{a}[n]$ and $a[\mathtt{p}](\alpha).Q$.*

An initial configuration represents a state from which a single session may start:

Definition 3 (Initial configuration)**.** *A configuration C_0 is* initial *if it is of the form*

$$C_0 = \langle a[1](\alpha_1).P_1 \mid \ldots \mid a[n](\alpha_n).P_n \mid \bar{a}[n], \emptyset, \emptyset\rangle$$

where for each $i = 1, \ldots, n$, process P_i is sequential and session-closed, and $c \in nm(P_i)$ implies $c = \alpha_i$.

We define two reduction relations on configurations, denoted $\longrightarrow$ and $\hookrightarrow_E$: while $\longrightarrow$ describes the evolution within an instant, $\hookrightarrow_E$ describes the evolution from one instant to the next one.

Reduction is defined modulo a structural congruence $\equiv$, whose rules are given in Fig. 5 and are standard [25]. The *reduction relation* $\longrightarrow$ describes the step-by-step execution of a configuration within an instant. It is defined by the rules in Fig. 6. Let us discuss some of them.

Rule [Init] describes the initiation of a new session s of service a among n processes of the required form. After the initiation, participants share a private session name s, and the channel variable $\alpha_{\mathtt{p}}$ is replaced by the session channel $s[\mathtt{p}]$ in each process $P_{\mathtt{p}}$.

Rule [Out] allows a sender $\mathtt{p}$ to broadcast a message by adding it to the memory, if $\mathtt{p}$ has not already sent a message in the current instant, namely if the output buffer of $\mathtt{p}$ has the form $s[\mathtt{p}] : \epsilon$. In the premise, $e \downarrow v$ denotes the evaluation of expression e to value v. If the message can be added, its content is set to v and its *reader set* is initialised to $\emptyset$.

Rule [In] allows a receiver $\mathtt{q}$ to fetch a message from sender $\mathtt{p}$ in the memory, if there exists one and if $\mathtt{q}$ has not already read it, namely if $\mathtt{q}$ does not belong to the reader set Π of the message. If $\mathtt{q}$ can read the message, then its value is substituted for the bound variable in the continuation process P, and the name $\mathtt{q}$ is added to the reader set Π.

Rule [Rec] inserts a `pause` before each recursive call, as usual in SRP, in order to allow at most one loop iteration at each instant and thus prevent the phenomenon known as *instantaneous loop* or *instantaneous divergence* [30]. Although parameterised recursion is used in our examples, for the sake of

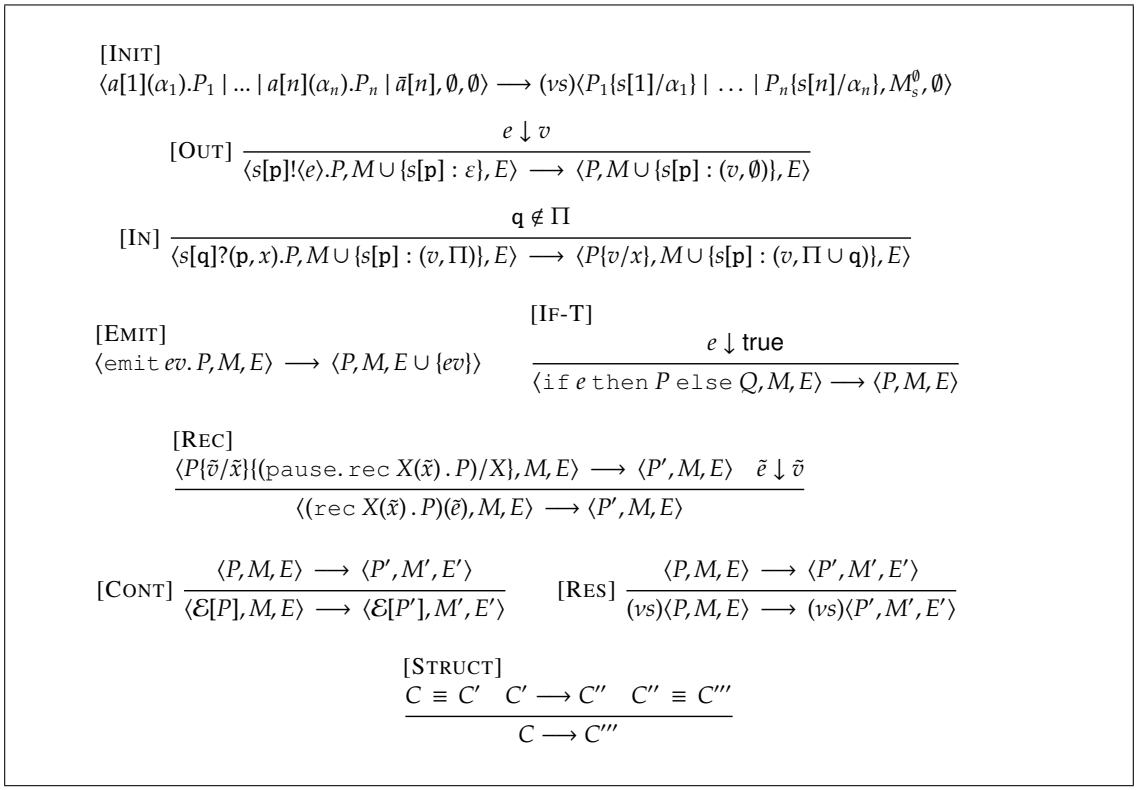


Figure 6: Reduction rules (with Rule [If-F] omitted).

simplicity we will focus on unparameterised recursion in the rest of the paper. Including parameters would have no impact on our results, but it would be tedious to carry them throughout our technical development.

The evaluation contexts $\mathcal{E}$ used in Rule [Cont] are defined by:

$$\mathcal{E} ::= [\,] \mid \mathcal{E} \mid P \mid P \mid \mathcal{E} \mid \texttt{watch}\, ev\, \texttt{do}\, \mathcal{E}\{Q\}$$

Note that the watch construct behaves as a static context as far as the reduction relation is concerned: the body of a watch process is executed up to the end of the instant, disregarding the event ev (which is relevant only for the relation $\hookrightarrow_E$ across instants).

To define the *tick transition relation* $\hookrightarrow_E$, we require two additional notions: the *suspension predicate* and the *reconditioning function*.

The suspension predicate $\langle P, M, E\rangle\ddagger$ (cf. Fig. 7) holds when all non-terminated components of P are in one of the following situations:

- wanting to release the control explicitly via a $\texttt{pause}.Q$ instruction;
- wanting to send a message after having already sent a message during the instant;
- awaiting a message from a participant who has not sent anything during the instant;
- awaiting a second message from the same participant during the same instant.

The reconditioning function (Fig. 8) "cleans-up" a process P and prepares it for the next instant: it erases all guarding pauses from $\texttt{pause}.Q$ processes, and triggers the alternative behaviour Q of all the processes $\texttt{watch}\, ev\, \texttt{do}\, P\{Q\}$ whose controlling event ev has been emitted.

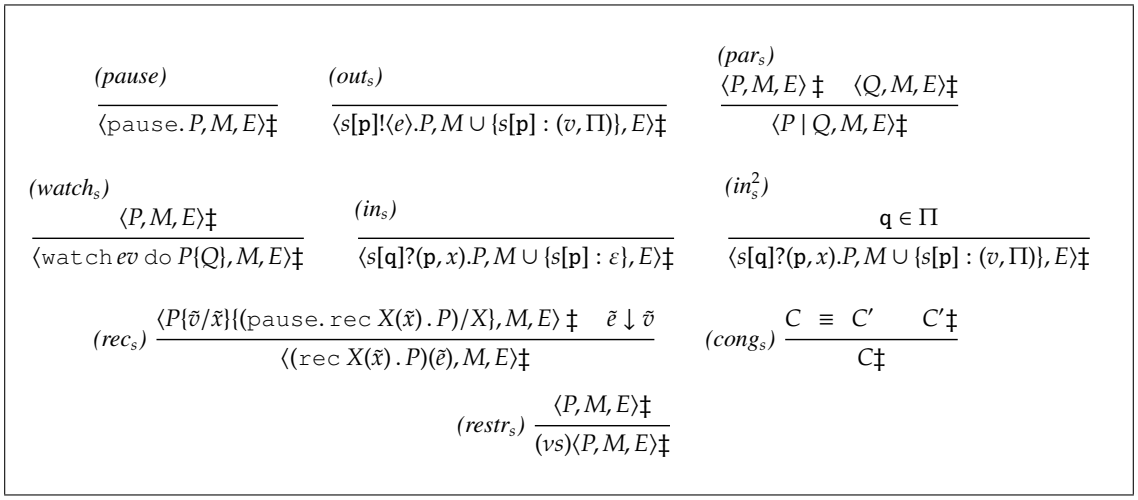


Figure 7: Suspension Predicate.

$$[P]_E = \begin{cases} R & \text{if } P = \texttt{pause}.R \\ [R]_E \mid [Q]_E & \text{if } P = R \mid Q \\ Q & \text{if } P = \texttt{watch}\, ev\, \texttt{do}\, R\{Q\}, ev \in E \text{ and } R \not\equiv \mathbf{0} \\ \texttt{watch}\, ev\, \texttt{do}\, [R]_E\{Q\} & \text{if } P = \texttt{watch}\, ev\, \texttt{do}\, R\{Q\}, ev \notin E \\ P & \text{otherwise} \end{cases}$$

Figure 8: Reconditioning Function.

$$(\text{TICK})\ \frac{(\nu s)\langle P, M, E\rangle\ddagger}{(\nu s)\langle P, M, E\rangle \hookrightarrow_E (\nu s)\langle [P]_E, M^{\emptyset}, \emptyset\rangle}$$

Figure 9: Tick transition.

The tick relation $\hookrightarrow_E$ applies only to suspended configurations: it formalises the passage of (logical) time and delimits the duration of broadcast by clearing out the memory and the event environment at the end of each instant. Formally, this relation is specified by the rule in Fig. 9, where $[P]_E$ is the reconditioning of P with respect to E.

As usual, we use $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$. We write $\rightsquigarrow$ to denote either $\longrightarrow$ or $\hookrightarrow_E$, and $\rightsquigarrow^*$ to stand for the reflexive and transitive closure of $\rightsquigarrow$.

The following definition of reachability characterises the configurations that may occur in the execution of a single session.

Definition 4 (Reachable configuration)**.** *A configuration C is* reachable *if there exists an initial configuration C_0 such that $C_0 \rightsquigarrow^* C$.*

Proposition 1. *If C is a* reachable *configuration that is not initial, then C has the form $C = (\nu s)\langle P, M, E\rangle$ and there exist an initial configuration $C_0 = \langle P_0, M_0, E_0\rangle$ and P_i, M_i, E_i for $i = 1, \ldots, n$ such that $C_0 \rightsquigarrow (\nu s)\langle P_1, M_1, E_1\rangle \rightsquigarrow \cdots \rightsquigarrow (\nu s)\langle P_n, M_n, E_n\rangle = (\nu s)\langle P, M, E\rangle$.*

From now on we will focus only on reachable configurations, without explicitly mentioning it.

3.3 Reactivity

In this section, we prove that single sessions are indeed *reactive* in our calculus, namely that every instant in their execution terminates. The single-session assumption is required, as it is well-known that interleaved sessions are subject to deadlock, and the possibility of deadlock would impair reactivity. On the other hand, it would be easy to extend our reactivity result to a pool of disjoint sessions evolving in parallel.

We now introduce some preliminary notation. First, we define the multi-step transition relation $C \Rightarrow C'$, together with its decorated variant $C \Rightarrow_n C'$ that keeps track of the number of execution steps between two configurations within an instant.

Definition 5 (Multi-step transition relation)**.** *The* decorated multi-step transition relation $C \Rightarrow_n C'$ *is defined by:*

$$C \Rightarrow_0 C \qquad\qquad (C \longrightarrow C' \ \wedge\ C' \Rightarrow_n C'') \ \Rightarrow\ C \Rightarrow_{n+1} C''$$

Then the multi-step transition relation $C \Rightarrow C'$ *is given by:*

$$C \Rightarrow C' \ \text{ if } \ \exists n \,.\, C \Rightarrow_n C'$$

Next, we define the notion of instantaneous convergence, which formalises the fact that a configuration may reach a state of termination or suspension in the current instant.

Definition 6 (Instantaneous convergence)**.** *The* immediate convergence *predicate is defined by:*

$$\begin{aligned} \langle P, M, E\rangle \overset{\ddagger}{\downarrow} \quad & \text{if } \ \langle P, M, E\rangle \ddagger \ \vee\ (P \equiv \mathbf{0}) \\ (\nu s)\langle P, M, E\rangle \overset{\ddagger}{\downarrow} \quad & \text{if } \ \langle P, M, E\rangle \overset{\ddagger}{\downarrow} \end{aligned}$$

Then the instantaneous convergence *relation and predicate are given by:*

$$C \Downarrow C' \ \text{ if } \ C \Rightarrow C' \ \wedge\ C' \overset{\ddagger}{\downarrow} \qquad\qquad C \Downarrow \ \text{ if } \ \exists C' \,.\, C \Downarrow C'$$

The annotated variants $\Downarrow_n$ may be defined in the obvious way:

$$C \Downarrow_n C' \ \text{ if } \ C \Rightarrow_n C' \ \wedge\ C' \overset{\ddagger}{\downarrow} \qquad\qquad C \Downarrow_n \ \text{ if } \ \exists C' \,.\, C \Downarrow_n C'$$

By abuse of notation, if $\sigma = C_0 \longrightarrow \cdots \longrightarrow C_n$ is a computation of C_0 and $C_n \overset{\ddagger}{\downarrow}$, we shall say that the computation σ converges (or converges to C_n).

We proceed now to prove reactivity. In fact, we shall prove a stronger property, *bounded reactivity*, which says that every configuration $\langle P, M, E\rangle$ instantaneously converges in a number of steps that is bounded by the *instantaneous size of process P in memory M*, denoted $\mathit{size}_M(P)$. Intuitively, $\mathit{size}_M(P)$ is an upper bound for the number of steps that P can execute during the first instant when run in memory M. Therefore, the idea for defining $\mathit{size}_M(P)$ is that it should not take into account the portion of P that follows a `pause` instruction (a "syntactic pause"). Moreover, $\mathit{size}_M(P)$ should span at most one iteration of recursive subprocesses, and ignore the alternative behaviour in watching subprocesses. Finally, in order to account for implicit pauses (or "semantic pauses"), $\mathit{size}_M(P)$ should stop counting when it meets an output on channel c, respectively an input on channel c from participant $\mathtt{p}$, in both process P and memory M.

Let $\mathcal{M}$ denote the set of memories.

Definition 7 (Communications and Fired channels of a memory)**.** *For any memory $M \in \mathcal{M}$, we define:*

$$\begin{aligned} \mathit{Fired}(M) &= \{(s, \mathtt{p}) \mid \exists v, \Pi.\ s[\mathtt{p}] : (v, \Pi) \in M\} \\ \mathit{Comm}(M) &= \{(s, \mathtt{p}, \mathtt{q}) \mid \exists v, \Pi.\, (s[\mathtt{p}] : (v, \Pi) \in M \ \wedge\ \mathtt{q} \in \Pi)\} \end{aligned}$$

The above functions allow us to extract useful information from the memory: $Fired(\cdot)$ identifies the participants that have sent a message in the current instant, and $Comm(\cdot)$ yields the pairs of participants that have successfully communicated in the current instant.

Definition 8 (Instantaneous size)**.** *The partial function size* : $(\mathcal{P} \times \mathcal{M}) \to \mathbf{Nat}$ *is defined inductively by:*

$size_M(\mathbf{0}) = size_M(X) = size_M(\texttt{pause}.P) = size_M(\bar{a}[n]) = 0$

$size_M(\texttt{emit}\ ev.P) = 1 + size_M(P)$

$size_\emptyset(a[\mathtt{p}](\alpha).P) = 1 + size_{M_s^\emptyset}(P\{s[\mathtt{p}]/\alpha\})$, *for any* s

$$size_M(s[\mathtt{p}]!\langle e\rangle.P) = \begin{cases} 0, & \text{if } (s,\mathtt{p}) \in Fired(M) \\ 1 + size_{M'}(P), \text{where} & \text{if } s[\mathtt{p}] : \epsilon \in M \\ M' = M\{s[\mathtt{p}] \mapsto (val(e), \emptyset)\} & \end{cases}$$

$$size_M(s[\mathtt{q}]?(\mathtt{p},x).P) = \begin{cases} 0, & \text{if either } s[\mathtt{p}] : \epsilon \in M \text{ or } (s,\mathtt{p},\mathtt{q}) \in Comm(M) \\ 1 + size_{M'}(P\{v/x\}), \text{where} & \text{if } s[\mathtt{p}] : (v,\Pi) \in M \wedge \mathtt{q} \notin \Pi \\ M' = M\{s[\mathtt{p}] \mapsto (v, \Pi \cup \mathtt{q})\} & \end{cases}$$

$size_M(\texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2) = 1 + max\{size_M(P_1), size_M(P_2)\}$

$size_M(P_1 \mid P_2) = size_M(P_1) + size_M(P_2)$

$size_M(\texttt{rec}\ X.P) = size_M(P)$

$size_M(\texttt{watch}\ ev\ \texttt{do}\ P\{Q\}) = size_M(P)$

As mentioned previously, the function $size_M(P)$ yields a bound for the number of steps that a configuration $\langle P, M, E\rangle$ may execute before reaching a state of suspension or termination. Note that the set of events E is irrelevant for this measure. Indeed, in our calculus (unlike in other SRP languages), the set of events only plays a role at the end of instants, and does not affect the reduction relation.

Definition 9 (Configuration instantaneous size)**.** *The function* $size_M(P)$ *induces a function* $Size(C)$ *on configurations, defined by:*

$$Size(\langle P, M, E\rangle) = size_M(P) \qquad Size((\nu s)C) = Size(C)$$

Although partial (because $size_M(P)$ is partial), the function $Size(C)$ is always defined for reachable configurations C, as established by the following lemma:

Lemma 1. *Let* C *be a reachable configuration. Then* $Size(C)$ *is defined.*

Proof. We distinguish two cases, depending on whether C is initial or reachable in at least one step.

1. Let $C = \langle P, \emptyset, \emptyset\rangle$ where $P = a[1](\alpha_1).P_1 \mid ... \mid a[n](\alpha_n).P_n \mid \bar{a}[n]$. Then it is immediate to see that $Size(\langle P, \emptyset, \emptyset\rangle) = size_\emptyset(P)$ is defined.
2. Let $C = (\nu s)\langle P, M, E\rangle$.There are only two possible cases where $size_M(P)$ may not be defined, namely the I/O cases $P = s[\mathtt{p}]!\langle e\rangle.Q$ and $P = s[q]?(\mathtt{p}, x).Q$, when $s[\mathtt{p}] \notin dom(M)$. However, this cannot happen, since C is derived from an initial configuration

   $$C_0 = \langle a[1](\alpha_1).P_1 \mid ... \mid a[n](\alpha_n).P_n \mid \bar{a}[n], \emptyset, \emptyset\rangle$$

   whose first reduction is necessarily of the form:

   $$C_0 \longrightarrow (\nu s)\langle P_1\{s[1]/\alpha_1\} \mid \ \ldots\ \mid P_n\{s[n]/\alpha_n\}, M_s^\emptyset, \emptyset\rangle = C_1$$

   By definition $s[i] \in dom(M_s)$ for each $i = 1, \ldots, n$. Then we may conclude, since $C_1 \longrightarrow^* C$ is deduced using rules different from [Init] and all these rules preserve $dom(M_s)$.

□

In the forthcoming proofs, we shall also use the following property:

Property 1. *For any process P and memory M, $size_M(\texttt{rec}\ X\,.\,P) = size_M(P\{(\texttt{pause}.\ \texttt{rec}\ X\,.\,P)/X\})$.*

Proof. Easy consequence of Def. 8, since $size_M(\texttt{rec}\ X\,.\,P) = size_M(P)$ and for any process Q, we have $size_M(\texttt{pause}.\,Q) = 0 = size_M(X)$. □

Before proving reactivity we will present some auxiliary results. We first prove that $size_M(P)$ decreases at each step of execution during an instant:

Lemma 2 (Size reduction during instantaneous execution)**.** *Let C be a reachable configuration. Then:*

$$C \longrightarrow C' \ \Rightarrow\ Size(C) > Size(C')$$

Proof. We distinguish two cases, depending on whether C is an initial configuration or not.

1. Let $C = \langle P, M, E\rangle$. Then $C \longrightarrow C' = (\nu s)\langle P', M', E'\rangle$ is deduced by Rule [Init]. Here $C = \langle a[1](\alpha_1).P_1 \mid ... \mid a[n](\alpha_n).P_n \mid \bar{a}[n], \emptyset, \emptyset\rangle$ and $C' = (\nu s)\langle P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\}, M_s^{\emptyset}, \emptyset\rangle$. Then we may conclude, since $size_M(P) = \sum_{i=1}^{n} size_M(a[i](\alpha_i).P_i) = n + \sum_{i=1}^{n} size_{M_s^{\emptyset}}(P_i\{s[i]/\alpha_i\}) = n + size_{M_s^{\emptyset}}(P') > size_{M_s^{\emptyset}}(P')$.

2. Let $C = (\nu s)\langle P, M, E\rangle$. In this case we have $(\nu s)\langle P, M, E\rangle \longrightarrow (\nu s)\langle P', M', E'\rangle$ if and only $\langle P, M, E\rangle \longrightarrow \langle P', M', E'\rangle$, and thus it is enough to prove the following statement:

If $(\nu s)\langle P, M, E\rangle$ is reachable then $\langle P, M, E\rangle \longrightarrow \langle P', M', E'\rangle$ implies $size_M(P) > size_{M'}(P')$

To prove this statement we proceed by induction on the inference of the transition $\langle P, M, E\rangle \longrightarrow \langle P', M', E'\rangle$, and case analysis on the last rule used in the inference. We examine the interesting cases.

- *Basic cases*
  - Rule [Out]. In this case we have $\langle P, M, E\rangle = \langle s[\mathtt{p}]!\langle e\rangle.Q, M'' \cup s[\mathtt{p}] : \epsilon, E\rangle$ and $\langle P', M', E'\rangle = \langle Q, M\{s[\mathtt{p}] \mapsto (v, \emptyset)\}, E\rangle$ where $e \downarrow v$. Then, since $(s, p) \notin Fired(M)$, we have $size_M(P) = 1 + size_{M'}(Q) = 1 + size_{M'}(P') > size_{M'}(P')$.
  - Rule [In]. Here we have $\langle P, M, E\rangle = \langle s[\mathtt{q}]?(\mathtt{p}, x).P, M'' \cup s[\mathtt{p}] : (v, \Pi), E\rangle$ for some Π such that $q \notin \Pi$, Moreover $\langle P', M', E'\rangle = \langle P\{v/x\}, M\{s[p] \mapsto (v, \Pi \cup \mathtt{q})\}, E\rangle$. In this case we have $size_M(P) = 1 + size_{M'}(P\{v/x\}) = 1 + size_{M'}(P') > size_{M'}(P')$.
- *Inductive cases*
  - Rule [Rec]. Here we have $\langle P, M, E\rangle = \langle \texttt{rec}X\,.\,Q, M, E\rangle$, and the reduction $\langle \texttt{rec}X\,.\,Q, M, E\rangle \longrightarrow \langle P', M', E'\rangle$ is deduced from the reduction

    $$\langle Q\{(\texttt{pause}.\ \texttt{rec}\ X\,.\,Q)/X\}, M, E\rangle \longrightarrow \langle P', M', E'\rangle$$

    by induction we have that $size_M(Q\{(\texttt{pause}.\ \texttt{rec}\ X\,.\,Q)/X\}) > size_{M'}(P')$. Whence also:

    $$size_M(P) = size_M(\texttt{rec}\ X\,.\,Q) = size_M(Q\{(\texttt{pause}.\ \texttt{rec}\ X\,.\,Q)/X\}) > size_{M'}(P').$$

  - Rule [Cont]. Easy induction.

□

We define now a specific notion of guardedness for recursive processes, which will be used in the proofs of the next two lemmas. In the sequel, a recursive process will often be referred to as "recursive call" or simply "call". A recursive call that is guarded by a `pause` statement is said to be *pause-guarded*. Formally:

Definition 10 (Pause-guardedness of recursive calls)**.** *A recursive call* $\texttt{rec}\, X\,.\,Q$ *is* pause-guarded *in process* P *if it appears within some subprocess* $\texttt{pause}.P'$ *of* P*. It is called* pause-unguarded *in* P *otherwise.*

The following lemma establishes that that every reachable configuration $\langle P, M, E\rangle$ immediately converges if and only if its size is equal to 0.

Lemma 3 (Immediate convergence of 0*-size* configurations)**.** *Let* C *be a reachable configuration. Then*

$$(Size(C) = 0) \;\Leftrightarrow\; C \overset{\ddagger}{\downarrow}_{\vee}$$

Proof. Note that C cannot be an initial configuration, since in this case we would have $Size(C) > 0$. Hence $C = (vs)\langle P, M, E\rangle$. Since $Size((vs)\langle P, M, E\rangle) = Size(\langle P, M, E\rangle) = size_M(P)$ and $(vs)\langle P, M, E\rangle \overset{\ddagger}{\downarrow}_{\vee} \Leftrightarrow \langle P, M, E\rangle \overset{\ddagger}{\downarrow}_{\vee}$, it is enough to prove the statement for $C = \langle P, M, E\rangle$.

We prove each side of the biconditional in turn:

(⇒) We proceed by simultaneous induction on the structure of P and on the number of pause-unguarded recursive calls in P, considering only the cases for which $size_M(P) = 0$. Note that the reachability assumption rules out the cases $P = X$, $P = \bar{a}[n]$ and $P = a[\mathtt{p}](\alpha).Q$, while the assumption $size_M(P) = 0$ rules out the cases $P = \texttt{emit}\, ev.\, Q$ and $P \equiv \texttt{if}\, e\, \texttt{then}\, P_1\, \texttt{else}\, P_2$.

Basic Cases

- $P = \mathbf{0}$. Then $P \equiv \mathbf{0}$ and thus $\langle P, M, E\rangle \overset{\ddagger}{\downarrow}_{\vee}$ by definition.
- $P = \texttt{pause}.Q$. Then $\langle P, M, E\rangle\ddagger$ by Rule (*pause*) in Fig. 7 and thus $\langle P, M, E\rangle \overset{\ddagger}{\downarrow}_{\vee}$ by definition.
- $P = s[\mathtt{p}]!\langle e\rangle.Q$. Since we assumed $size_M(P) = 0$, we have $(s, \mathtt{p}) \in Fired(M)$, i.e., there exist v, Π such that $s[\mathtt{p}] : (v, \Pi) \in M$. Then $\langle P, M, E\rangle\ddagger$ by Rule (out_s) in Fig. 7.
- $P \equiv s[\mathtt{q}]?(\mathtt{p}, x).Q$. Since $size_M(P) = 0$, then either $(s \in sn(M) \wedge (s, \mathtt{p}) \notin Fired(M))$ or $(s, \mathtt{p}, \mathtt{q}) \in Comm(M)$. In both cases we can deduce $\langle P, M, E\rangle\ddagger$, respectively by Rule ($in_s$) and by Rule ($in_s^2$) in Fig. 7.

Inductive Cases

- $P = P_1 \mid P_2$. Since $size_M(P) = 0$, we have $size_M(P_i) = 0$ for $i = 1, 2$. By induction, this implies $\langle P_i, M, E\rangle\ddagger$ for $i = 1, 2$. Whence, by Rule (par_s) in Fig. 7, we deduce $\langle P_1 \mid P_2, M, E\rangle\ddagger$.
- $P = \texttt{rec}\, X\,.\,Q$. Since $size_M(P) = 0$, also $size_M(Q\{(\texttt{pause}.\texttt{rec}\, X\,.\,Q)/X\}) = 0$, by Property 1. By induction on the number of pause-unguarded recursive calls, we have therefore $\langle Q\{(\texttt{pause}.\texttt{rec}\, X\,.\,Q)/X\}, M, E\rangle\ddagger$. Whence we may conclude that $\langle P, M, E\rangle\ddagger$, using Rule ($rec_s$) in Fig. 7.
- $\texttt{watch}\, ev\, \texttt{do}\, Q\{R\}$. Since $size_M(P) = 0$, we have $size_M(Q) = 0$. By induction $\langle Q, M, E\rangle\ddagger$. Then $\langle P, M, E\rangle\ddagger$ by Rule ($watch_s$) in Fig. 7.

($\Leftarrow$) There are two possibilities for $\langle P, M, E\rangle \overset{\ddagger}{\nu}$:

1) If $P \equiv \mathbf{0}$, we proceed by induction on the definition of $\equiv$. In essence, $P \equiv \mathbf{0}$ if and only if P is an n-ary parallel composition whose components are either $\mathbf{0}$ or of the form $\texttt{watch}\, ev\, \texttt{do}\, Q\{R\}$, where $Q \equiv \mathbf{0}$. In all cases, by Def. 8 we have $size_M(P) = 0$.

2) If $\langle P, M, E\rangle\ddagger$, we proceed by induction on the definition of the suspension predicate in Fig. 7. In each case, the reasoning is dual to that for the ($\Rightarrow$) direction above.

□

There are two reasons why reactivity could fail in our calculus: 1) a process that loops forever during an instant - what is generally called an *instantaneous loop* in SRP [30]; 2) a deadlock due to a mismatch between a participant and the memory: this happens when a participant $\mathtt{p}$ in a session s wants to broadcast a message while the output buffer $s[\mathtt{p}] : m$ is not in the memory, or to receive a message from participant $\mathtt{q}$ while the output buffer $s[\mathtt{q}] : m$ is not in the memory. Our semantic rule for recursion is designed to prevent instantaneous loops, and will be the key for proving Reactivity (Theorem 2). The absence of mismatches between participants and the memory is guaranteed by the reachability assumption. The following lemma establishes that reachable configurations are deadlock-free.

Lemma 4 (Deadlock freedom)**.** *Let C be a reachable configuration. Then either $C \overset{\ddagger}{\nu}$ or $\exists\, C' \,.\, C \longrightarrow C'$.*

Proof. We distinguish two cases, depending on whether C is an initial configuration or not.

1. Let $C = \langle P, M, E\rangle$ be an initial configuration. Then there is a reduction $C \longrightarrow C' = (\nu s)\langle P', M', E'\rangle$ deduced by Rule [INIT].

2. Let $C = (\nu s)\langle P, M, E\rangle$. Then C reduces if and only if $\langle P, M, E\rangle$ reduces and $C \overset{\ddagger}{\nu}$ if and only if $\langle P, M, E\rangle \overset{\ddagger}{\nu}$. Hence it is enough to prove the property for $\langle P, M, E\rangle$.

We proceed by induction on the structure of P. Note that the reachability assumption rules out the cases $P = X$, $P = \bar{a}[n]$ and $P = a[\mathtt{p}](\alpha).Q$.

Basic Cases

- $P = \mathbf{0}$. Then $\langle P, M, E\rangle \overset{\ddagger}{\nu}$ by definition.
- $P = \texttt{emit}\, ev.\, Q$. By Rule [EMIT], for any M, E we have $\langle \texttt{emit}\, ev.\, Q, M, E\rangle \longrightarrow \langle Q, M, E \cup \{ev\}\rangle$.
- $P = \texttt{pause}.\, Q$. By Rule (*pause*), for any M, E we have $\langle P, M, E\rangle\ddagger$.
- $P = \texttt{if}\, e\, \texttt{then}\, P_1\, \texttt{else}\, P_2$. Since the evaluation of e terminates, for any M, E a reduction may be inferred by either Rule [IF-T] or Rule [IF-F].
- $P \equiv s[\mathtt{p}]!\langle e\rangle.Q$. By the reachability condition, $s[\mathtt{p}] \in dom(M)$. There are then two possibilities:
  (i) $M = M' \cup \{s[\mathtt{p}] : \varepsilon\}$, in which case $\langle P, M, E\rangle$ can reduce by Rule [OUT];
  (ii) $M = M' \cup \{s[\mathtt{p}] : (v, \Pi)\}$, in which case $\langle P, M, E\rangle\ddagger$ by Rule (out_s).
- $P \equiv s[\mathtt{q}]?(\mathtt{p}, x).Q$. In this case, there are three possibilities:
  (i) there is a message sent by $\mathtt{p}$ that participant $\mathtt{q}$ has not read yet, in which case the configuration can reduce by Rule [IN];
  (ii) there is no message sent by $\mathtt{p}$, in which case the configuration is suspended by Rule (in_s);
  (iii) there is a message sent by $\mathtt{p}$ that participant $\mathtt{q}$ has read already, in which case the configuration is suspended by Rule (in_s^2).

Inductive cases. Straightforward, by applying the inductive hypothesis.

□

We are now ready to prove two (increasingly strong) reactivity results for reachable configurations:

1) standard *reactivity*, which amounts to the convergence of all computations of C within an instant;

2) *bounded reactivity*, which gives a bound for the number of steps of such converging computations.

Theorem 1 (Reactivity)**.** *Let C be a reachable configuration. Then $C \Downarrow$.*

Proof. Every sequence of consecutive reductions of C must be finite, because $Size(C)$ is defined by Lemma 1, and it strictly decreases along execution by Lemma 2. Moreover, no derivative of C may be deadlocked, by Lemma 4. Hence the size of the configuration eventually becomes 0 and the resulting configuration immediately converges by Lemma 3. □

We proceed now to prove bounded reactivity, namely that every reachable configuration C instantaneously converges in a number of steps that is bounded by $Size(C)$.

Theorem 2 (Bounded reactivity)**.** *Let C be a reachable configuration. Then*

$$\exists\, n \leq Size(C)\,.\; C \Downarrow_n$$

Proof. We distinguish two cases, depending on whether C is an initial configuration or not. Since the latter case depends on the former, we start by considering non initial configurations.

1. C is not initial. In this case, C has the form $C = (\nu s)\langle P, M, E\rangle$, where P is a parallel composition of sequential session-closed processes. Therefore we have:

$$\begin{array}{rcl} Size((\nu s)\langle P, M, E\rangle) & = & Size(\langle P, M, E\rangle) \;=\; size_M(P) \\ (\nu s)\langle P, M, E\rangle \overset{\ddagger}{\downarrow} & \Leftrightarrow & \langle P, M, E\rangle \overset{\ddagger}{\downarrow} \\ (\nu s)\langle P, M, E\rangle \longrightarrow (\nu s)\langle P', M', E'\rangle & \Leftrightarrow & \langle P, M, E\rangle \longrightarrow \langle P', M', E'\rangle \end{array}$$

Therefore it is enough to prove the statement:

$$\exists\, n \leq size_M(P)\,.\;\; \langle P, M, E\rangle \Downarrow_n$$

We proceed by simultaneous induction on the structure of P, on the size of P and on the number of pause-unguarded recursive calls in P.

- *Basic case: $size_M(P) = 0$ and P has no pause-unguarded recursive calls.*

  By Lemma 3, if $size_M(P) = 0$ then either $P \equiv \mathbf{0}$ or $\langle P, M, E\rangle\ddagger$. By definition $\langle P, M, E\rangle \overset{\ddagger}{\downarrow} \Rightarrow (\langle P, M, E\rangle \Downarrow_0 \langle P, M, E\rangle)$, Then we may conclude, since $n = 0 = size_M(P)$.

- *Inductive cases: $size_M(P) \geq 1$ or P has pause-unguarded calls.*

  – $P = \texttt{emit}\; ev.\, P'$. In this case, by Rule [Emit] we have the reduction:

  $$\langle \texttt{emit}\; ev.\, P', M, E\rangle \longrightarrow \langle P', M, E \cup \{ev\}\rangle$$

  By induction (on the size or on the structure), there exists $n \leq size_M(P')$ such that $\langle P', M, E \cup \{ev\}\rangle \Downarrow_n$. Then $\langle \texttt{emit}\; ev.\, P', M, E\rangle \Downarrow_{n+1}$, where $n + 1 \leq size_M(P') + 1 = size_M(\texttt{emit}\; ev.\, P')$.

– $P = s[\mathtt{q}]?(\mathtt{p}, x).Q$. By Lemma 4, there are two possibilities:

  – $\langle s[\mathtt{q}]?(\mathtt{p}, x).Q, M, E\rangle\ddagger$. This is inferred either using Rule (in_s), if $s \in sn(M) \wedge (s, p) \notin Fired(M)$, or using Rule ($in_s^2$), if $(s, p, q) \in Comm(M)$. Then $\langle s[\mathtt{q}]?(\mathtt{p}, x).Q, M, E\rangle \Downarrow_0 \langle Q, M, E\rangle$ and we may conclude, since $n = 0 \leq size_M(P)$.

  – $\langle P, M, E\rangle = \langle s[\mathtt{q}]?(\mathtt{p}, x).Q, M'' \cup s[\mathtt{p}] : (v, \Pi), E\rangle$ for some M'', v and Π such that $q \notin \Pi$. Then Rule [In] can be applied, yielding $\langle P', M', E'\rangle = \langle Q\{v/x\}, M\{s[p] \mapsto (v, \Pi \cup \mathtt{q})\}, E\rangle$. Then by induction there exists $n \leq size_M(P')$ such that $\langle P', M', E'\rangle \Downarrow_n$. Hence $\langle P, M, E\rangle \Downarrow_{n+1}$, where $n + 1 \leq size_M(P') + 1 = size_M(s[\mathtt{q}]?(\mathtt{p}, x).Q)$.

– $P = s[\mathtt{p}]!\langle e\rangle.Q$. This case is similar to the previous one, and slightly simpler.

– $P = \mathtt{rec}\, X\,.\, Q$. By Lemma 4, there are two possibilities:

  – $\langle \mathtt{rec}\, X\,.\, Q, M, E\rangle\ddagger$. Then $\langle \mathtt{rec}\, X\,.\, Q, M, E\rangle \Downarrow_0 \langle \mathtt{rec}\, X\,.\, Q, M, E\rangle$ and we may conclude, since $n = 0 = size_M(P)$.

  – There exist P', M', E' such that $\langle P, M, E\rangle \longrightarrow \langle P', M', E'\rangle$. Then the reduction is inferred by Rule [Rec], namely:

$$\frac{\langle Q\{(\mathtt{pause}.\, \mathtt{rec}\, X\,.\, Q)/X\}, M, E\rangle \longrightarrow \langle P', M', E'\rangle}{\langle \mathtt{rec}\, X\,.\, Q, M, E\rangle \longrightarrow \langle P', M', E'\rangle} \quad \text{[Rec]}$$

    Since the call $\mathtt{rec}\, X\,.\, Q$ is pause-guarded in P', the number of pause-unguarded calls in P' is strictly less than in P. Then by induction there exists $n \leq size_M(P')$ such that $\langle P', M', E'\rangle \Downarrow_n$. Whence $\langle P, M, E\rangle \Downarrow_{n+1}$. By Lemma 2, we know that $size_M(P') < size_M(Q\{(\mathtt{pause}.\, \mathtt{rec}\, X\,.\, Q)/X\})$. By Property 1, $size_M(Q\{(\mathtt{pause}.\, \mathtt{rec}\, X\,.\, Q)/X\}) = size_M(P)$. We may thus conclude that $n < size_M(P)$, that is to say, $n + 1 \leq size_M(P)$.

– Conditional, parallel and watching constructs: these cases are straightforward, by induction on the structure of the process.

2. $C = \langle P, M, E\rangle$ is initial. Thus we have $C = \langle a[1](\alpha_1).P_1 \mid ... \mid a[k](\alpha_k).P_k \mid \bar{a}[k], \emptyset, \emptyset\rangle$. Then by Rule [Init] we have a reduction $C \longrightarrow C' = (\nu s)\langle P', M_s^{\emptyset}, \emptyset\rangle$, where $P' = (\nu s)P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_k\{s[k]/\alpha_k\}$. By Definition 8 $Size(C) = \sum_{i=1}^{k} size_{\emptyset}(a[i](\alpha_i).P_i) = k + \sum_{i=1}^{k} size_{M_s^{\emptyset}}(P_i\{s[i]/\alpha_i\}) = k + size_{M_s^{\emptyset}}(P') > size_{M_s^{\emptyset}}(P') = Size(C')$. By Point 1. there exists $m \leq Size(C')$ such that $C' \Downarrow_m$. Letting $n = m + 1$, we conclude that $C \Downarrow_n$ and $n \leq Size(C)$.

□

Sorts	S	::=	$\mathsf{bool} \mid \mathsf{int} \mid \ldots$	
Global types	G	::=	$\mathtt{p}\uparrow\langle S, \Pi\rangle.G$	*communication*
		$\mid$	$\mathtt{pause}.G$	*explicit pause*
		$\mid$	$\mathtt{tick}.G$	*implicit pause*
		$\mid$	$\mathtt{watch}\ ev\ \mathtt{do}\ G\ \mathtt{else}\ G$	*global watch*
		$\mid$	$\mathbf{t}$	*recursive variable*
		$\mid$	$\mu\mathbf{t}.G$	*recursion*
		$\mid$	$\mathtt{end}$	*end*
Local Types	T	::=	$!S.T$	*send*
		$\mid$	$?(\mathtt{p}, S).T$	*receive*
		$\mid$	$\mathtt{pause}.T$	*explicit pause*
		$\mid$	$\mathtt{tick}.T$	*implicit pause*
		$\mid$	$\langle T, T'\rangle^{ev}$	*local watch*
		$\mid$	$\mu\mathbf{t}.T$	*recursion*
		$\mid$	$\mathbf{t}$	*recursive variable*
		$\mid$	$\mathtt{end}$	*end*
Message Types	ϑ	::=	$\mathtt{void} \mid (S, \Pi)$	

Figure 10: Sorts, Global types, Local types and Message types.

4 Types for MRS

In this section we present the session type system for MRS. We show that typability implies the classical properties of session calculi, namely the absence of communication errors (communication safety) and the conformance to the session protocol (session fidelity). Furthermore, our type system enforces the following properties, which are specific to our synchronous reactive setting and will be discussed in more detail later:

P1. *Output persistence*: Every participant broadcasts exactly one message during every time instant.

P2. *Input timeliness*: Every unguarded input is matched by an output during the current instant, if not preceded by another input with equal source and target, or during the next instant, if not preempted.

4.1 Global and Local Types

Our calculus allows multiparty communication [19, 14]. Hence, typing relies on *global types* to describe communication protocols and on *local types* to describe the contributions of protocol participants.

Our type syntax, given in Fig. 10, uses sorts, ranged over by $S, S', \ldots$, global types ranged over by $G, G', \ldots$, local types, ranged over by $T, T', \ldots$, and type variables, ranged over by $\mathbf{t}, \mathbf{t}', \ldots$. Sorts denote basic types such as int and bool and type variables are used when defining recursive types. We recall that participants are denoted by $\mathtt{p}, \mathtt{q}, \ldots$ or by natural numbers. Similarly, sets of participants are denoted by Π, Π'. A peculiarity of our typing system is that for every sort S we assume a default value d_S, representative of the particular basic type S. We present the syntax of both global and local types below.

Global types:

- Type $\mathtt{p} \uparrow \langle S, \Pi \rangle.G$ represents a participant $\mathtt{p}$ broadcasting a message of sort S, with participants in Π as intended receivers; subsequently, the interaction continues as specified by G. As a well-formedness condition for the type above, we require $\mathtt{p} \notin \Pi$.
- Type $\mathtt{pause}.G$ stipulates that all participants should jointly move to the next time instant in order to execute protocol G.
- Type $\mathtt{watch}\ ev\ \mathtt{do}\ G\ \mathtt{else}\ G'$ represents a protocol which has an alternative behaviour. Intuitively, protocol G is executed until the end of the current instant; then, if event ev has been emitted, the governing protocol at the next instant will be G', otherwise it will be the continuation of G. Note that if G' becomes the governing protocol, protocol G is pre-empted.
- Type $\mathtt{end}$ represents the terminated protocol.

Local types:

- The *send* type $!S.T$ indicates the broadcast of a value of sort S, followed by the behaviour described by T.
- The *receive* type $?(\mathtt{p}, S).T$ describes the reception of a value of sort S sent by participant $\mathtt{p}$, followed by the behaviour described by T.
- The *pause* type $\mathtt{pause}.T$ signals the change of time instant, followed by the behaviour described by T.
- The *local watch* type $\langle T, T' \rangle^{ev}$ is meant to be assigned to a participant that behaves as specified by type T until the end of instant; then, if event ev has appeared, it will behave according to type T', otherwise according to the continuation of T.

We assume recursive types $\mu\mathbf{t}.G$ and $\mu\mathbf{t}.T$ to be contractive (i.e., type variables only appear under the prefixes). Moreover, we take an *equi-recursive* view of types, meaning that we do not distinguish between $\mu\mathbf{t}.G$ (resp. $\mu\mathbf{t}.T$) and its unfolding $G\{\mu\mathbf{t}.G/\mathbf{t}\}$ (resp. $T\{\mu\mathbf{t}.T/\mathbf{t}\}$). As a consequence, we never consider types of the form $\mu\mathbf{t}.T$ in typing rules. Indeed, whenever we find a type $\mu\mathbf{t}.T$ in a typing rule, we pick another type equal to it (i.e., its unfolding $T\{\mu\mathbf{t}.T/\mathbf{t}\}$).

Finally, types $\mathtt{tick}.G$ and $\mathtt{tick}.T$ are introduced to represent the implicit pauses that arise from the semantics of processes. In a sense, it can be said that $\mathtt{tick}$ is "runtime" type, which is added dynamically when obtaining the local types of the process.

Example 1. *We now present some examples of global and local types that can be written using the syntax in Fig. 10. Assume participants* $\mathtt{p}, \mathtt{q}, \mathtt{r}$*:*

$\mathtt{p} \uparrow \langle S, \{\mathtt{q}, \mathtt{r}\} \rangle.\mathtt{pause}.\mathtt{r} \uparrow \langle S, \{\mathtt{p}, \mathtt{q}\} \rangle.\mathtt{end}$	$\mathtt{pause}.\mathtt{r} \uparrow \langle S, \{\mathtt{p}, \mathtt{q}\} \rangle.\mathtt{end}$
$\mu t.\mathtt{r} \uparrow \langle S, \{\mathtt{p}, \mathtt{q}\} \rangle.t$	$\mu t.\mathtt{pause}.t$
$!S.?(\mathtt{p}, S').\mathtt{end}$	$\mu t.!S.!S'.t$

$$
\begin{array}{ll}
\mathsf{S}_{\mathcal{R}}(\mathtt{p}\!\uparrow\!\langle S,\Pi\rangle.G',\mathcal{P}) = \mathtt{p}\!\uparrow\!\langle S,\Pi\rangle.\mathsf{S}_{\mathcal{R}}(G',\mathcal{P}\cup\{\mathtt{p}\}) & \text{if } \mathtt{p}\notin\mathcal{P}\\
\mathsf{S}_{\mathcal{R}}(\mathtt{p}\!\uparrow\!\langle S,\Pi\rangle.G',\mathcal{P}) = \mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\emptyset\rangle.\mathtt{tick}.\mathtt{p}\!\uparrow\!\langle S,\Pi\rangle.\mathsf{S}_{\mathsf{Part}(\mathsf{G}')}(G',\{\mathtt{p}\}) & \text{if } \mathtt{p}\in\mathcal{P} \text{ and } \mathcal{R}\setminus\mathcal{P}=\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}\\
\\
\mathsf{S}_{\mathcal{R}}(\mathtt{pause}.G',\mathcal{P}) = \mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\emptyset\rangle.\mathtt{pause}.\mathsf{S}_{\mathsf{Part}(\mathsf{G}')}(G',\emptyset) & \text{with } \mathcal{R}\setminus\mathcal{P}=\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}\\
\mathsf{S}_{\mathcal{R}}(\mathtt{tick}.G',\mathcal{P}) = \mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\emptyset\rangle.\mathtt{pause}.\mathsf{S}_{\mathsf{Part}(\mathsf{G}')}(G',\emptyset) & \text{with } \mathcal{R}\setminus\mathcal{P}=\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}\\
\\
\mathsf{S}_{\mathcal{R}}(\mathtt{end},\mathcal{P}) = \mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\emptyset\rangle.\mathtt{end} & \text{with } \mathcal{R}\setminus\mathcal{P}=\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}\\
\\
\mathsf{S}_{\mathcal{R}}(\mathtt{watch}\ ev\ \mathtt{do}\ G'\ \mathtt{else}\ G'',\mathcal{P}) = \mathtt{watch}\ ev\ \mathtt{do}\ \mathsf{S}_{\mathcal{R}}(G',\mathcal{P})\ \mathtt{else}\ \mathsf{S}_{\mathsf{Part}(\mathsf{G}'')}(G'',\emptyset) & \\
\\
\mathsf{S}_{\mathcal{R}}(\mu\mathbf{t}.G',\emptyset) = \mu\mathbf{t}.\ \mathsf{S}_{\mathsf{Part}(\mathsf{G}')}(G',\emptyset) & \\
\mathsf{S}_{\mathcal{R}}(\mu\mathbf{t}.G',\mathcal{P}) = \mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\emptyset\rangle.\mathtt{tick}.\mu\mathbf{t}.\ \mathsf{S}_{\mathsf{Part}(\mathsf{G}')}(G',\emptyset) & \text{with } \mathcal{R}\setminus\mathcal{P}=\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}\\
\\
\mathsf{S}_{\mathcal{R}}(\mathbf{t},\mathcal{P}) = \mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\emptyset\rangle.\mathtt{pause}.\mathbf{t} & \text{with } \mathcal{R}\setminus\mathcal{P}=\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}
\end{array}
$$

Figure 11: Saturation of global types.

4.2 Projection and Saturation

As usual in multiparty session calculi, global and local types are related by the notion of *projection*. Intuitively, the projection of a global type G onto its participants generates the local types for every participant using the information given by G.

In our reactive setting, however, projection requires a pre-processing phase in which the global type is modified by adding the necessary implicit pauses. Indeed, as hinted before, `tick` represents implicit pauses, which are induced by the semantics, rather than by a pause explicitly written in the protocol specification. This pre-processing phase is called *saturation* and besides adding the necessary implicit pauses, it adds outputs that may be missing to ensure output persistence.

Before formally introducing saturation and projection, we define some auxiliary notation. We write $\mathsf{Part}(G)$ to represent the set of participants declared in G. We also find it useful to write $\mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\Pi\rangle.G$ to abbreviate the global type $\mathtt{p}_1\!\uparrow\!\langle S,\Pi\rangle.\cdots.\mathtt{p}_n\!\uparrow\!\langle S,\Pi\rangle.G$, describing n consecutive broadcasts of messages of the same sort S from each of the participants $\mathtt{p}_1,\ldots,\mathtt{p}_n$ to the set of participants in Π. Moreover, whenever $n=0$, $\mathtt{p}_{\{1,n\}}\!\uparrow\!\langle S,\Pi\rangle.G$ denotes the global type G.

We now introduce saturation (cf. Fig. 11) and projection (cf. Fig. 12). As hinted above, the saturation function makes the global type reflect precisely the slicing into instants of the protocol's behaviour. Intuitively, a global type is saturated as follows: first, the saturation function identifies the correct instant slicing in the protocol adding the necessary `tick`. Next, the function saturates the global type with outputs, guaranteeing that in each instant all participants broadcast a message (even if it is not received by anyone). Formally, the function $\mathsf{S}_{\mathcal{R}}(G,\mathcal{P})$, takes a global type G, the set of currently active participants, written $\mathcal{R}$, and a set that collects the participants of $\mathcal{R}$ that have already sent a message in the current instant, denoted $\mathcal{P}\subseteq\mathcal{R}$.

Our typing system will then require *saturated projections*, which simply stand for the projection of the saturated global type. Intuitively, the *saturated projection* of G onto $\mathtt{q}$, denoted $G\lfloor_{\mathtt{q}}$, yields a local type representing $\mathtt{q}$'s involvement in the protocol described by G. It is obtained in two steps: first, G is

$$(\mathtt{p}\!\uparrow\!\langle S,\Pi\rangle.G')\upharpoonright \mathtt{q} = \begin{cases} !S.(G'\upharpoonright \mathtt{q}) & \text{if } \mathtt{q}=\mathtt{p}\\ ?(\mathtt{p},S).(G'\upharpoonright \mathtt{q}) & \text{if } \mathtt{q}\in\Pi,\\ G'\upharpoonright \mathtt{q} & \text{otherwise.}\end{cases}$$

$$(\mathtt{pause}.G')\upharpoonright \mathtt{q} = \begin{cases} \mathtt{end} & \text{if } \mathtt{q}\notin \mathsf{Part}(G')\wedge \mathbf{t} \text{ does not occur in } G'\\ \mathtt{pause}.(G'\upharpoonright \mathtt{q}) & \text{otherwise.}\end{cases}$$

$$(\mathtt{tick}.G')\upharpoonright \mathtt{q} = \begin{cases} \mathtt{end} & \text{if } \mathtt{q}\notin \mathsf{Part}(G')\wedge \mathbf{t} \text{ does not occur in } G'\\ \mathtt{tick}.(G'\upharpoonright \mathtt{q}) & \text{otherwise.}\end{cases}$$

$$(\mathtt{watch}\ ev\ \mathtt{do}\ G_1\ \mathtt{else}\ G_2)\upharpoonright \mathtt{q} = \langle G_1\upharpoonright \mathtt{q}, G_2\upharpoonright \mathtt{q}\rangle^{ev}$$

$$(\mu\mathbf{t}.G')\upharpoonright \mathtt{q} = \begin{cases} \mu\mathbf{t}.(G'\upharpoonright \mathtt{q}) & \text{if } \mathtt{q} \text{ occurs in } G'\wedge \mathbf{t} \text{ does not occur in } G',\\ \mathtt{end} & \text{otherwise.}\end{cases} \qquad \mathbf{t}\upharpoonright \mathtt{q}=\mathbf{t} \qquad \mathtt{end}\upharpoonright \mathtt{q}=\mathtt{end}$$

Figure 12: Projection of global types onto participants.

saturated as described in Fig. 11; then the resulting global type is projected onto $\mathtt{q}$ as described in Fig. 12:

$$G\lfloor_{\mathtt{q}} = (\mathsf{S}_{\mathsf{Part}(\mathsf{G})}(G,\emptyset)\upharpoonright \mathtt{q}) \tag{1}$$

To illustrate our definitions of projection and saturation, let us look back at the auction example of Section 2.

Example 2 (Types for the *Auction* process)**.** *Recall that the set of participants of the Auction process is* $\{\mathtt{A},\mathtt{B}_1,\ldots,\mathtt{B}_n\}$*, where* $\mathtt{A}$ *represents the Auctioneer and* $\mathtt{B}_i$ *the process Bidder*$_i$ *(*$1\le i\le n$*).*

The global type G is as follows:

$$\begin{aligned} G = \mathtt{watch}\ \mathit{bis}\ \mathtt{do}\ \mu\boldsymbol{t}.\mathtt{B}_1\!\uparrow\!\langle \mathsf{int},\{\mathtt{A}\}\rangle.\ \cdots\ .\mathtt{B}_n\!\uparrow\!\langle \mathsf{int},\{\mathtt{A}\}\rangle.\mathtt{A}\!\uparrow\!\langle \widetilde{\mathsf{int}},\{\mathtt{B}_1,\ldots,\mathtt{B}_n\}\rangle.\boldsymbol{t}\\ \mathtt{else}\ \mathtt{A}\!\uparrow\!\langle \mathsf{int},\{\mathtt{B}_1,\ldots,\mathtt{B}_n\}\rangle.\mathtt{B}_1\!\uparrow\!\langle \mathsf{string},\emptyset\rangle.\ \cdots\ .\mathtt{B}_n\!\uparrow\!\langle \mathsf{string},\emptyset\rangle.\mathtt{end}\end{aligned}$$

Then, the saturated global type $\mathcal{G}=\mathsf{S}(G,\emptyset)$ *is built according to Fig. 11, as follows:*

$$\begin{aligned} \mathsf{S}(G,\emptyset) = \mathtt{watch}\ \mathit{bis}\ \mathtt{do}\ \mu\boldsymbol{t}.\mathtt{B}_1\!\uparrow\!\langle \mathsf{int},\{\mathtt{A}\}\rangle.\ \cdots\ .\mathtt{B}_n\!\uparrow\!\langle \mathsf{int},\{\mathtt{A}\}\rangle.\mathtt{A}\!\uparrow\!\langle \widetilde{\mathsf{int}},\{\mathtt{B}_1,\ldots,\mathtt{B}_n\}\rangle.\mathtt{pause}.\boldsymbol{t}\\ \mathtt{else}\ \mathtt{A}\!\uparrow\!\langle \mathsf{int},\{\mathtt{B}_1,\ldots,\mathtt{B}_n\}\rangle.\mathtt{B}_1\!\uparrow\!\langle \mathsf{string},\emptyset\rangle.\ \cdots\ .\mathtt{B}_n\!\uparrow\!\langle \mathsf{string},\emptyset\rangle.\mathtt{end}\end{aligned}$$

The only difference with respect to G is the addition of a `pause` *before the recursion variable* $\boldsymbol{t}$*.*

Finally, the saturated projections of G (i.e., the projections of $\mathcal{G}$*) onto participants are as follow:*

$$\begin{aligned} G\lfloor_{\mathtt{A}} &= \langle \mu\boldsymbol{t}_1.?(\mathtt{B}_1,\mathsf{int}).\ \cdots\ .?(\mathtt{B}_n,\mathsf{int}).!\widetilde{\mathsf{int}}.\mathtt{pause}.\boldsymbol{t}_1, !\mathsf{int}.\mathit{end}\rangle^{\mathit{bis}}\\ G\lfloor_{\mathtt{B}_i} &= \langle \mu\boldsymbol{t}_2.!\mathsf{int}.?(\mathtt{A},\widetilde{\mathsf{int}}).\mathtt{pause}.\boldsymbol{t}_2, ?(\mathtt{A},\mathsf{int}).!\mathsf{string}.\mathit{end}\rangle^{\mathit{bis}}\end{aligned}$$

$$\lfloor\textsc{Service}\rfloor \quad \Gamma, a : G \vdash a : G \qquad\qquad \lfloor\textsc{ProcVar}\rfloor \quad \Gamma, X : T \vdash X : S\,T$$

Figure 13: Typing rules for services and process variables.

4.3 Type System

Typing judgements for our type system rely on three kinds of typing environments: *standard environments* ranged over by $\Gamma, \Gamma', \ldots$, *session environments* denoted by $\Delta, \Delta', \ldots$, and *message environments* denoted by Θ. Standard environments, map variables to sort types, service names to global types, and process variables to local types. Formally:

$$\Gamma ::= \emptyset \mid \Gamma, x : S \mid \Gamma, a : G \mid \Gamma, X : T$$

We write $\Gamma, x : S$ only if x does not occur in $dom(\Gamma)$, where $dom(\Gamma)$ denotes the domain of Γ, i.e., the set of identifiers occurring in Γ. We adopt the same convention for $\Gamma, a : G$ and $\Gamma, X : T$.

Session environments assign local types to channels occurring in processes, while message environments, assign message types (cf. Fig. 10) to channels occurring in memories. More specifically, a message environment assigns to a channel c in the memory the type ϑ of the message carried by c, namely the type `void` if no message has been sent on c, and the type (S, Π) if a message of type S has been sent on c and has been read by the participants in Π.

The syntax of message types ϑ is given in Fig. 10. Then, session and message environments are formally defined by the following grammars:

$$\Delta ::= \emptyset \mid \Delta, c : T \qquad\qquad \Theta ::= \emptyset \mid \Theta, c : \vartheta$$

For $\Delta, c : T$ and $\Theta, c : \vartheta$ we use the same conventions as for Γ, meaning that a session environment $\Delta, c : T$ (resp. a message environment $\Theta, c : \vartheta$) is only well-defined if $c \notin dom(\Delta)$ (rep. $c \notin dom(\Theta)$). Thus, a session environment Δ_1, Δ_2 is only well-defined if $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$.

Following the same notation introduced for memories, we write $\Theta^\emptyset$ to represent the message environment obtained from Θ by turning all message types to `void`. Intuitively, if Θ types memory M then $\Theta^\emptyset$ types memory $M^\emptyset$. Formally, $\Theta^\emptyset =_{\mathsf{def}} \bigcup_{c \in dom(\Theta)} \{c : \mathtt{void}\}$.

Below, we give two auxiliary definitions. The first one introduces notations for referring to session environments that contain only ended behaviours and environments that contains at least one "live" behaviour. The second one defines a way to extract the "active" participants in the current instant from the memory M and memory environment Θ.

Definition 11 (Live and terminated session environments)**.** *A session environment Δ is said to be* live *if $c : T \in \Delta$ implies $T \neq \mathtt{end}$, and* terminated *if $c : T \in \Delta$ implies $T = \mathtt{end}$. Any session environment Δ may be partitioned in two session environments Δ^{live} and Δ^{end} defined by:*

$$\begin{array}{lll} \Delta^{live} & =_{\mathsf{def}} & \{c : T \in \Delta \mid T \neq \mathtt{end} \wedge T \neq \langle \mathtt{end}, T \rangle^{ev}\} \\ \Delta^{end} & =_{\mathsf{def}} & \{c : T \in \Delta \mid T = \mathtt{end} \vee T = \langle \mathtt{end}, T \rangle^{ev}\} \end{array}$$

Definition 12 (Visible domain of memories and message environments)**.**
The visible domain *of memories and message environments is defined by:*

$$\begin{array}{lll} vdom(M) & = & \{c \mid c : m \in M \wedge m \neq \epsilon\} \\ vdom(\Theta) & = & \{c \mid c : \vartheta \in \Theta \wedge \vartheta \neq \mathtt{void}\} \end{array}$$

$$\lfloor \textsc{BoolVal} \rfloor \quad \Gamma \vdash \mathsf{true}, \mathsf{false} : \mathsf{bool} \qquad \lfloor \textsc{IntVal} \rfloor \quad \Gamma \vdash 1, 2, \ldots : \mathsf{int} \qquad \lfloor \textsc{DVal} \rfloor \quad \Gamma \vdash d_S : S$$

$$\lfloor \textsc{Var} \rfloor \quad \Gamma, x : S \vdash x : S \qquad \lfloor \textsc{And} \rfloor \frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \mathsf{bool}}{\Gamma \vdash e_1 \ \mathsf{and} \ e_2 : \mathsf{bool}} \qquad \lfloor \textsc{Sum} \rfloor \frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}$$

Figure 14: Typing rules for expressions.

$$\lfloor \textsc{VoidMsg} \rfloor \ \Gamma \vdash c : \epsilon \triangleright c : \mathtt{void} \qquad \lfloor \textsc{FullMsg} \rfloor \frac{\Gamma \vdash v : S}{\Gamma \vdash c : (v, \Pi) \triangleright c : (S, \Pi)}$$

$$\lfloor \textsc{EmptyMem} \rfloor \ \Gamma \vdash \emptyset \triangleright \emptyset \qquad \lfloor \textsc{MergeMem} \rfloor \frac{\Gamma \vdash M \triangleright \Theta \quad \Gamma \vdash c : m \triangleright c : \vartheta}{\Gamma \vdash M \cup \{c : m\} \triangleright \Theta, \{c : \vartheta\}}$$

Figure 15: Typing rules for memories.

Our type system uses three kinds of type judgements. The first one, used for typing expressions, is defined by the rules in Fig. 14 and has the form:

$$\Gamma \vdash e : S$$

where Γ represents a standard environment, e and expression and S a sort. The second one, used for typing memories, is defined by the rules in Fig. 15 and has the form:

$$\Gamma \vdash M \triangleright \Theta$$

where Θ is a message environment associating a message type with each channel in M. Finally, the third kind of judgement is used for typing configurations and has the form:

$$\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta \rangle$$

where $\langle \Delta \diamond \Theta \rangle$ is called a *configuration environment*. The reason for using configuration environments is because the state of the memory directly impact the behaviour of the process. Thus, it becomes necessary for types to have knowledge about the memory at every point of the typing derivation.

Intuitively, if $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$, then Δ is a session environment typing the channels of P in memory M, and Θ is a message environment typing the messages of M.

Before presenting the typing rules for configurations, we introduce some auxiliary notions and results. We say that a local type is *output granting* if the first `pause` in it (if any) is preceded by an output.

Definition 13 (Output-granting type)**.** *A local type T is* output-granting *if it satisfies the predicate* $\mathsf{OG}(T)$ *defined by:*

$$\mathsf{OG}(T) = \begin{cases} \mathsf{true} & \text{if } T = !S.T' \\ \mathsf{OG}(T') & \text{if } T = ?(\mathtt{p}, S).T' \vee T = \langle T', T'' \rangle^{ev} \vee T = \mu \mathbf{t}.T' \\ \mathsf{false} & \text{otherwise} \end{cases}$$

The predicate is extended to session environments by letting $\mathsf{OG}(\Delta)$ *if* $\mathsf{OG}(T)$ *for all c:T in* Δ.

We can then show that local types that come from the projection of a saturated global type are output granting.

Lemma 5 (Correctness of saturation with respect to $\mathsf{OG}(T)$)**.** *Let G be a global type with $|\mathsf{Part}(G)| \geq 2$, $\mathtt{p}$ be a participant, and $\mathcal{P}$ be a set of participants such that (1) $\mathcal{P} \subseteq \mathsf{Part}(G)$, and (2) $\mathsf{Part}(G) \setminus \mathcal{P} \neq \emptyset$. Then, $\mathsf{OG}(\mathsf{S}_{\mathsf{Part(G)}}(G, \mathcal{P}) \upharpoonright \mathtt{p})$ holds for every $\mathtt{p} \in \mathsf{Part}(G) \setminus \mathcal{P}$.*

Proof. By induction on the structure of G. The base cases are $G = \mathsf{end}$ and $G = \mathbf{t}$, which are vacuously true since $|\mathsf{Part}(\mathsf{end})| = |\mathsf{Part}(\mathbf{t})| = 0$. For the inductive step we assume that the property holds for a global type G' and prove the statement for every type that contains G' as a sub-expression. There are 5 inductive cases.

Case $G = \mathtt{pause}.G'$**:**

(1)	$\mathsf{S}_{\mathsf{Part(G)}}(G, \mathcal{P}) \upharpoonright \mathtt{p} = \mathsf{S}_{\mathsf{Part(G)}}(\mathtt{pause}.G', \mathcal{P}) \upharpoonright \mathtt{p}$ $= \mathtt{p}_{\{1,n\}} \uparrow \langle S, \emptyset \rangle.\mathtt{pause}.\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{p}$	(Assumption, Fig. 11)
(2)	$\mathsf{Part}(G) \setminus \mathcal{P} = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$	(Assumption, Fig. 11)
(3)	$\forall \mathtt{p}_i \in \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}.(\mathtt{p}_{\{1,n\}} \uparrow \langle S, \emptyset \rangle.\mathtt{pause}.\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{p}_i$ $= !S_i.\mathtt{pause}.(\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{p}))$	((1), (2), Fig. 12)
(4)	$\mathsf{OG}(!S_i.\mathtt{pause}.(\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{p}))$ is true	(Def. 13, (3))

Case $G = \mathtt{tick}.G'$**:** Analogous to the case above.

Case $G = \mathtt{r} \uparrow \langle S, \Pi \rangle.G'$**:** There are two cases depending on whether $\mathtt{r} \in \mathcal{P}$ or not:

Case $\mathtt{r} \notin \mathcal{P}$**:**

(1)	$\mathsf{S}_{\mathsf{Part(G)}}(\mathtt{r} \uparrow \langle S, \Pi \rangle.G', \mathcal{P}) = \mathtt{r} \uparrow \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \mathcal{P} \cup \{\mathtt{r}\})$	(Def. 11)
(2)	$\forall \mathtt{p} \in \mathcal{P}.(\mathsf{OG}(\mathsf{S}_{\mathsf{Part(G)}}(G', \mathcal{P}) \upharpoonright \mathtt{p}))$	(IH)
(3)	$\mathsf{OG}(\mathtt{r} \uparrow \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \mathcal{P} \cup \{\mathtt{r}\}) \upharpoonright \mathtt{r})$ is true	(Def. 12, (2), (1))

Case $\mathtt{r} \in \mathcal{P}$**:** By applying Fig. 11 to G, which adds an output for every $\mathtt{r} \in \mathcal{P}$.

Case $G = \mathtt{watch}\ ev\ \mathtt{do}\ G'\ \mathtt{else}\ G''$**:** Follows directly from the IH. Notice that by Def. 13, $\mathsf{OG}(\langle T_1, T_2 \rangle^{ev}) = \mathsf{OG}(T_1)$.

□

Corollary 1 (Projection implies *OG*)**.** *For any global type G such that $\mathsf{Part}(G) \geq 2$ and any participant $\mathtt{p} \in \mathsf{Part}(G)$, $\mathsf{OG}(G\lfloor_{\mathtt{p}})$ holds.*

Proof. The proof follows from Lemma 5. Notice that in this case $\mathcal{P} = \emptyset$, hence $\mathsf{Part}(G) \setminus \mathcal{P} = \mathsf{Part}(G)$.
□

Next, we define the *pair composition* of session types, which intuitively generates end or $\langle T_1, T_2 \rangle^{ev}$, depending on the value of T_1.

Definition 14. *The* pair composition *of session types T_1 and T_2 under event ev, written $T_1 \star_{ev} T_2$, is defined by:*

$$T_1 \star_{ev} T_2 := \begin{cases} T_1 & if\ \ T_1 = \mathsf{end} \\ \langle T_1, T_2 \rangle^{ev} & otherwise. \end{cases}$$

Abusing notation, we extend this definition to session environments by writing $\Delta_1 \star_{ev} \Delta_2$.

Given that our typing rules are defined over session environments and that our semantics induce suspension points for all processes in parallel inside a configuration, we define functions $\mathsf{pause}(\cdot)$ and $\mathsf{tick}(\cdot)$ to extract all the suspended sessions of a session environment.

Definition 15. *Given a session environment* Δ*, we write* $\mathsf{pause}(\Delta)$ *to denote the environment defined as* $\mathsf{pause}(\Delta) = \{c : \mathtt{pause}.T \mid c : T \in \Delta\}$*. Analogously* $\mathsf{tick}(\Delta)$ *denotes the environment defined as* $\mathsf{tick}(\Delta) = \{c : \mathtt{tick}.T \mid c : T \in \Delta\}$.

We next introduce generalised types for session channels occurring in configurations. Note that in a configuration, channels may occur on the process side, on the memory side, or on both sides. Since our semantics allows at most one message to be sent on each channel during an instant, both our message types and our generalised types are simpler than those of standard asynchronous multiparty session calculi [14]. In particular, in our calculus the syntax of generalised types coincides with that of local types.

Definition 16 (Generalised Types)**.** *A* generalised type $\mathcal{T}$ *is a local type* T *or a send type (extracted from a message type* ϑ*) followed by a local type* T*.*
Let $\langle\Delta \diamond \Theta\rangle$ *be a configuration environment and* $s[\mathtt{p}] \in dom(\langle\Delta \diamond \Theta\rangle)$*. Then the generalised type of* $s[\mathtt{p}]$ *in* $\langle\Delta \diamond \Theta\rangle$ *is given by:*

$$\langle\Delta \diamond \Theta\rangle(s[\mathtt{p}]) = \begin{cases} T & \text{if } s[\mathtt{p}] : T \in \Delta \ \wedge \ s[\mathtt{p}] \notin vdom(\Theta), \\ !S.\,T & \text{if } s[\mathtt{p}] : (S, \Pi) \in \Theta \ \wedge \ (s[\mathtt{p}] : T \in \Delta \ \vee \ (s[\mathtt{p}] \notin dom(\Delta) \ \wedge \ T = \mathtt{end})\,), \\ \mathtt{end} & \text{if } s[\mathtt{p}] \notin dom(\Delta) \ \wedge \ s[\mathtt{p}] : \mathtt{void} \in dom(\Theta) \\ \textit{undefined} & \textit{otherwise}. \end{cases}$$

The generalised type $\langle\Delta\diamond\Theta\rangle(s[\mathtt{p}])$ represents the usage of channel $s[\mathtt{p}]$ in the configuration environment $\langle\Delta \diamond \Theta\rangle$: it is the concatenation of the send type extracted from the message sent by p in the current instant, if any, with the local type describing the remaining behaviour of p.

Example 3 (Generalised types)**.**
Let $\Delta = s[1] : \mathtt{end}$ *and* $\Theta = \emptyset$*. Then* $\langle\Delta \diamond \Theta\rangle(s[1]) = \mathtt{end}$.
Let $\Delta = \emptyset$ *and* $\Theta = s[1] : \mathtt{void}$*. Then* $\langle\Delta \diamond \Theta\rangle(s[1]) = \mathtt{end}$.
Let $\Delta = s[1] : \mathtt{end}$ *and* $\Theta = s[1] : \mathtt{void}$*. Then* $\langle\Delta \diamond \Theta\rangle(s[1]) = \mathtt{end}$.
Let $\Delta = s[1] : !S.\mathtt{end}$ *and* $\Theta = s[1] : \mathtt{void}$*. Then* $\langle\Delta \diamond \Theta\rangle(s[1]) = !S.\mathtt{end}$.
Let $\Delta = s[1] : \mathtt{end}$ *and* $\Theta = s[1] : (S, \Pi)$*. Then* $\langle\Delta \diamond \Theta\rangle(s[1]) = !S.\mathtt{end}$.
Let $\Delta = s[1] : !S.\mathtt{end}$ *and* $\Theta = s[1] : (S', \Pi)$*. Then* $\langle\Delta \diamond \Theta\rangle(s[1]) = !S'.!S.\mathtt{end}$.

Generalised types may be projected on participants (notation $\mathcal{T} \upharpoonright \mathtt{q}$), as described in Fig. 16. These projections are *anonymous local types* (local types where sender names are omitted), defined as follows:

$$\tau ::= \mathtt{end} \mid !S.T \mid ?S.T \mid \mathtt{pause}.T \mid \mathtt{tick}.T \mid \langle T, T'\rangle^{ev} \mid \mu\mathbf{t}.T \mid \mathbf{t}$$

Intuitively, the projection of the generalised type $\langle\Delta\diamond\Theta\rangle(s[\mathtt{p}])$ on $\mathtt{q}$, namely $\langle\Delta\diamond\Theta\rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q}$, describes the part of $\mathtt{p}$'s contribution to $\langle\Delta \diamond \Theta\rangle$ that concerns $\mathtt{q}$.

We may now define a duality relation $\bowtie$ between projections of generalised types. Informally, duality holds when the inputs offered by one side are matched by the outputs offered by the other side. The requirement is weaker in the other direction, since outputs do not need to be matched by inputs in a broadcast setting (the actual receivers may range from none to all participants except the sender). Hence, two types may be dual although not completely symmetric. In this respect, we depart from standard session calculi, where the requirement is symmetric for inputs and outputs. Dual types are expected to have matching explicit and implicit pauses, as well as matching watching statements, whose types are required to be pairwise dual.

$$(!S.T)\upharpoonright \mathtt{q} = \ !S.\,(T\upharpoonright \mathtt{q}) \qquad (?(\mathtt{p},S).T)\upharpoonright \mathtt{q} = \begin{cases} ?S.(T\upharpoonright \mathtt{q}) & \text{if } \mathtt{q}=\mathtt{p},\\ T\upharpoonright \mathtt{q} & \text{otherwise.}\end{cases}$$

$$(\mathtt{pause}.T)\upharpoonright \mathtt{q} = \mathtt{pause}.(T\upharpoonright \mathtt{q}) \qquad (\mathtt{tick}.T)\upharpoonright \mathtt{q} = \mathtt{tick}.(T\upharpoonright \mathtt{q}) \qquad (\langle T_1,T_2\rangle^{ev})\upharpoonright \mathtt{q} = \langle T_1\upharpoonright \mathtt{q}, T_2\upharpoonright \mathtt{q}\rangle^{ev}$$

$$(\mu \mathbf{t}.T)\upharpoonright \mathtt{q} = \begin{cases} \mu \mathbf{t}.(T\upharpoonright \mathtt{q}) & \text{if } \mathtt{q} \text{ occurs in } T \wedge \mathbf{t} \text{ does not occur in } G'\\ \mathtt{end} & \text{otherwise.}\end{cases} \qquad \mathbf{t}\upharpoonright \mathtt{q} = \mathbf{t} \qquad \mathtt{end}\upharpoonright \mathtt{q} = \mathtt{end}$$

Figure 16: Projection of generalised types on participants.

Definition 17. *The* duality relation *between projections of generalised types is the minimal symmetric relation which satisfies:*

$$\mathtt{end}\bowtie\mathtt{end} \qquad \boldsymbol{t}\bowtie\boldsymbol{t} \qquad T\bowtie T' \Rightarrow \mu\boldsymbol{t}.T\bowtie\mu\boldsymbol{t}.T' \qquad T\bowtie T' \Rightarrow\ !S.T\bowtie ?S.T'$$
$$T\bowtie T' \Rightarrow\ !S.T\bowtie T' \qquad T\bowtie T' \Rightarrow \mathtt{pause}.T\bowtie\mathtt{pause}.T' \qquad T\bowtie T' \Rightarrow \mathtt{tick}.T\bowtie\mathtt{tick}.T'$$
$$\langle \mathtt{end},T\rangle^{ev}\bowtie\mathtt{end} \qquad T_1\bowtie T_3 \textit{ and } T_2\bowtie T_4 \Rightarrow \langle T_1,T_2\rangle^{ev}\bowtie\langle T_3,T_4\rangle^{ev}$$
$$\mathtt{end}\bowtie T \Rightarrow \mathtt{end}\bowtie\mathtt{pause}.T \qquad \mathtt{end}\bowtie T \Rightarrow \mathtt{end}\bowtie\mathtt{tick}.T$$

Notice that terminated types may be dual to non terminated ones, due to the clause $T\bowtie T' \Rightarrow\ !S.T\bowtie T'$ and to the last two clauses of the definition. However, such non terminated types can only be sequences of send types or explicit/implicit pauses followed by send types, as for instance in $\mathtt{end}\bowtie\mathtt{pause}.!S.!S'.\mathtt{end}$.

Example 4 (Dual projections of generalised types)**.**

1. *Let* $\Delta = s[1] : !S.\,\mathtt{end}, s[2] : ?(1,S).\mathtt{end}$ *and* $\Theta = s[1] : \mathtt{void}, s[2] : \mathtt{void}$.
   Then $\langle\Delta\diamond\Theta\rangle(s[1])\upharpoonright 2 = !S.\mathtt{end} \ \bowtie\ ?S.\mathtt{end} = \langle\Delta\diamond\Theta\rangle(s[2])\upharpoonright 1$

2. *Let* $\Delta = s[1] : \mathtt{end}, s[2] : ?(1,S).\mathtt{end}$ *and* $\Theta = s[1] : (S,\emptyset), s[2] : \mathtt{void}$.
   Then $\langle\Delta\diamond\Theta\rangle(s[1])\upharpoonright 2 = !S.\mathtt{end} \ \bowtie\ ?S.\mathtt{end} = \langle\Delta\diamond\Theta\rangle(s[2])\upharpoonright 1$

3. *Let* $\Delta = s[1] : \mathtt{end}, s[2] : ?(1,S).\mathtt{end}$ *and* $\Theta = s[1] : (S,\{2\}), s[2] : \mathtt{void}$.
   Then $\langle\Delta\diamond\Theta\rangle(s[1])\upharpoonright 2 = !S.\mathtt{end} \ \bowtie\ ?S.\mathtt{end} = \langle\Delta\diamond\Theta\rangle(s[2])\upharpoonright 1$

4. *Let* $\Delta = s[1] : !S.\,\mathtt{end}, s[2] : ?(1,S).!S'.\,\mathtt{end}$ *and* $\Theta = s[1] : \mathtt{void}, s[2] : \mathtt{void}$.
   Then $\langle\Delta\diamond\Theta\rangle(s[1])\upharpoonright 2 = !S.\mathtt{end} \ \bowtie\ ?S.!S'.\mathtt{end} = \langle\Delta\diamond\Theta\rangle(s[2])\upharpoonright 1$
   Note that here the mutual projections are dual although not symmetric.

5. *Let* $\Delta = s[1] : \mathtt{pause}.!S_1.\,\mathtt{end}, s[2] : \mathtt{pause}.?(1,S_1).?(3,S_3).!S_2.\,\mathtt{end}, s[3] : ?(1,S_1).\mathtt{pause}.!S_3.\,\mathtt{end}$,
   and $\Theta = s[1] : (S_1,\{2\}), s[2] : (S_2,\emptyset), s[3] : (S_3,\emptyset)$. *Then:*
   $\langle\Delta\diamond\Theta\rangle(s[1])\upharpoonright 2 = !S_1.\mathtt{pause}.!S_1.\mathtt{end}$ *and* $\langle\Delta\diamond\Theta\rangle(s[2])\upharpoonright 1 = !S_2.\,\mathtt{pause}.?S_1.!S_2.\,\mathtt{end}$
   $\langle\Delta\diamond\Theta\rangle(s[1])\upharpoonright 3 = !S_1.\mathtt{pause}.!S_1.\mathtt{end}$ *and* $\langle\Delta\diamond\Theta\rangle(s[3])\upharpoonright 1 = !S_3.?S_1.\mathtt{pause}.!S_3.\,\mathtt{end}$
   $\langle\Delta\diamond\Theta\rangle(s[2])\upharpoonright 3 = !S_2.\mathtt{pause}.?S_3.!S_2.\,\mathtt{end}$ *and* $\langle\Delta\diamond\Theta\rangle(s[3])\upharpoonright 2 = !S_3.\,\mathtt{pause}.!S_3.\,\mathtt{end}$

We are now ready to define *coherence* of configuration environments. Besides the usual compatibility condition between the types of participants, our notion of coherence also requires output persistence:

Definition 18 (Coherence)**.** *A configuration environment* $\langle \Delta \diamond \Theta \rangle$ *is* coherent*, written* $Co\,\langle \Delta \diamond \Theta \rangle$*, if:*

1. *For any* $s[\mathtt{p}] \in dom(\langle \Delta \diamond \Theta \rangle)$*, if* $s[\mathtt{p}] \notin vdom(\Theta)$ *then* $s[\mathtt{p}] : T \in \Delta^{live}$ *implies* $\mathsf{OG}(T)$*;*
2. *For any* $\mathtt{p}, \mathtt{q}$*, if* $s[\mathtt{p}] \in dom(\Delta) \cup vdom(\Theta)$ *and* $s[\mathtt{q}] \in dom(\Delta) \cup vdom(\Theta)$ *then:*

$$\langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q} \bowtie \langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) \upharpoonright \mathtt{p}$$

In Definition 18, Condition 2. is the standard duality requirement for any pair of present participants in $\langle \Delta \diamond \Theta \rangle$ (i.e., participants whose session channel in s has some type in Δ or a non-void message type in Θ): it essentially requires that $\mathtt{p}$ and $\mathtt{q}$ make dual communications offers to each other. Note that any configuration environment whose domain is a singleton $\{s[\mathtt{p}]\}$ trivially satisfies this condition. Condition 1. is specific to our calculus and is meant to ensure output persistence: it says that if a participant has not yet sent a message in the current instant, then it better do so before the next suspension point is reached. The fact that the type of $\mathtt{p}$ is saturated, with all suspension points represented by explicit pauses, plays an essential role here.

It is easy to see that the first three configuration environments in Example 4 are not coherent, because they violate Condition 1. (while they satisfy Condition 2.). On the other hand, the last two configuration environments are coherent.

We prove now that any two projections of the same global type have dual mutual projections:

Lemma 6. *If* $(\mathsf{S}_{\mathcal{R}}(G, \mathcal{P}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathcal{R}}(G, \mathcal{P}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$*,* $\mathcal{P} \subseteq \mathcal{P}'$*,* $\mathsf{Part}(G) \subseteq \mathcal{R} \subseteq \mathcal{R}'$*,* $\mathcal{P} \subseteq \mathcal{R}$ *and* $\mathcal{P}' \subseteq \mathcal{R}'$ *then* $(\mathsf{S}_{\mathcal{R}'}(G, \mathcal{P}') \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathcal{R}'}(G, \mathcal{P}') \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$*.*

Proof. By induction on the structure of G. There are two base cases and five inductive cases.

Base Cases: The cases are $G = \mathtt{end}$ and $G = \mathbf{t}$. We only show $G = \mathtt{end}$, as the other is similar. Let $\mathcal{P}, \mathcal{P}'$, $\mathcal{R}$, and $\mathcal{R}'$ be sets of participants satisfying satisfying the following assumptions:

Case $G = \mathtt{end}$**:**

(1)	$\mathsf{Part}(G) \subseteq \mathcal{R} \subseteq \mathcal{R}'$	(Assumption)
(2)	$\mathcal{P} \subseteq \mathcal{R}$	(Assumption)
(3)	$\mathcal{P}' \subseteq \mathcal{R}'$	(Assumption)
(4)	$(\mathsf{S}_{\mathcal{R}}(\mathtt{end}, \mathcal{P}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathcal{R}}(\mathtt{end}, \mathcal{P}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$	(Assumption)
(5)	$\mathcal{P} \subseteq \mathcal{P}'$	(Assumption)

Then, we distinguish cases depending on whether $\mathtt{p}, \mathtt{q}$ belong to $\mathcal{P}'$ or not.

(a) $\mathtt{p} \in \mathcal{P}'$ and $\mathtt{q} \in \mathcal{P}'$.
(b) $\mathtt{p} \notin \mathcal{P}'$ and $\mathtt{q} \notin \mathcal{P}'$.
(c) $\mathtt{p} \in \mathcal{P}'$ and $\mathtt{q} \notin \mathcal{P}'$.
(d) $\mathtt{p} \notin \mathcal{P}'$ and $\mathtt{q} \in \mathcal{P}'$.

All the cases proceed similarly, hence we only show (a):

Sub-Case (a):

(i)	$\forall \mathtt{r} \in \mathcal{P}.(\mathsf{S}_{\mathcal{R}'}(\mathtt{end}, \mathcal{P}') = \mathsf{S}_{\mathcal{R}'}(\mathtt{end}, \mathcal{P}') = \mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset \rangle.\mathtt{end}$	(Fig. 11)
(ii)	$(\mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset \rangle.\mathtt{end} \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = (!S_d.\mathtt{end}) \upharpoonright \mathtt{q} = !S_d.\mathtt{end}$	(Fig. 12, Fig. 16, (i))
(iii)	$(\mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset \rangle.\mathtt{end} \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p} = (!S_d) \upharpoonright \mathtt{p} = !S_d.\mathtt{end}$	(Fig. 12, Fig. 16, (i))
(iv)	$!S_d.\mathtt{end} \bowtie !S_d.\mathtt{end}$	(Def. 17, (ii), (iii))

Inductive Cases: There are five cases. As in the base cases, let $\mathcal{P}, \mathcal{P}'$, $\mathcal{R}$, and $\mathcal{R}'$ be sets of participants satisfying satisfying the following assumptions:

(1) $\mathsf{Part}(G) \subseteq \mathcal{R} \subseteq \mathcal{R}'$ (Assumption)
(2) $\mathcal{P} \subseteq \mathcal{R}$ (Assumption)
(3) $\mathcal{P}' \subseteq \mathcal{R}'$ (Assumption)
(4) $(\mathsf{S}_{\mathcal{R}}(G, \mathcal{P}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathcal{R}}(G, \mathcal{P}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ (Assumption)
(5) $\mathcal{P} \subseteq \mathcal{P}'$ (Assumption)

Then, we distinguish cases depending on the shape of G. The IH states that the property holds for all the subterms G' of type G.

Case $G = \mathtt{r} \uparrow \langle S, \Pi \rangle . G'$**: IH:** $(\mathsf{S}_{\mathcal{R}}(\mathtt{r} \uparrow \langle S, \Pi \rangle . G', \mathcal{P}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathcal{R}}(\mathtt{r} \uparrow \langle S, \Pi \rangle . G', \mathcal{P}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ implies that
$(\mathsf{S}_{\mathcal{R}'}(\mathtt{r} \uparrow \langle S, \Pi \rangle . G', \mathcal{P}') \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathcal{R}'}(\mathtt{r} \uparrow \langle S, \Pi \rangle . G', \mathcal{P}') \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ for every $\mathcal{R}', \mathcal{P}'$ such that $\mathcal{R} \subseteq \mathcal{R}' \wedge \mathcal{P} \subseteq \mathcal{P}'$.

We then further distinguish two cases depending on whether $\mathtt{r} \in \mathcal{P}$ or $\mathtt{r} \notin \mathcal{P}$:

Case $\mathtt{r} \in \mathcal{P}$**:** By Fig. 11, $\mathsf{S}_{\mathcal{R}}(\mathtt{r} \uparrow \langle S, \Pi \rangle . G', \mathcal{P}') = \mathtt{r} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{r}\})$. We then further distinguish eight sub-cases depending on the conditions satisfied by $\mathtt{p}$ and $\mathtt{q}$:

Sub-case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \in \Pi$**:** Then $G = \mathtt{p} \uparrow \langle S, \Pi \rangle . G'$:

(i) $\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\}) \upharpoonright \mathtt{p} = \, !S.(\mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\})) \upharpoonright \mathtt{p}$ (Fig. 12)

(ii) $\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\}) \upharpoonright \mathtt{p} \upharpoonright \mathtt{q} = \, !S.((\mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\})) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}$ (Fig. 16)

(iii) $\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\}) \upharpoonright \mathtt{q} \upharpoonright \mathtt{p} = \, ?(\mathtt{q}, S).((\mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\})) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ (Fig. 12 and Fig. 16)

By (1) and (5), $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{P} \subseteq \mathcal{P}'$. Thus, we distinguish two more sub-cases: $\mathtt{p} \notin \mathcal{P}'$ and $\mathtt{p} \in \mathcal{P}$:

Sub-case $\mathtt{p} \notin \mathcal{P}'$**:** Notice that (i)$\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathcal{R}'}(G', \mathcal{P}' \cup \{\mathtt{p}\}) \upharpoonright \mathtt{p} \upharpoonright \mathtt{q} = !S.((\mathsf{S}_{\mathcal{R}'}(G', \mathcal{P}' \cup \{\mathtt{p}\})) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}$ by Fig. 12 and Fig. 16. Also, by assumption, Fig. 12 and Fig. 16 we have (ii) $\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathcal{R}'}(G', \mathcal{P} \cup \{\mathtt{p}\}) \upharpoonright \mathtt{q} \upharpoonright \mathtt{p} = ?(\mathtt{q}, S).((\mathsf{S}_{\mathcal{R}}(G', \mathcal{P}' \cup \{\mathtt{p}\})) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$. Since by assumption, $((\mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\})) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie ((\mathsf{S}_{\mathcal{R}}(G', \mathcal{P} \cup \{\mathtt{p}\})) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ and that $\mathcal{R}' \supseteq \mathcal{R} \wedge \mathcal{P}' \supseteq \mathcal{P}$, we can apply the inductive hypothesis to obtain (iii) $((\mathsf{S}_{\mathcal{R}'}(G', \mathcal{P}' \cup \{\mathtt{p}\})) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie ((\mathsf{S}_{\mathcal{R}'}(G', \mathcal{P}' \cup \{\mathtt{p}\})) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ and then we can conclude by Def. 17.

Sub-case $\mathtt{p} \in \mathcal{P}'$**:** Notice that (i) $\mathsf{S}_{\mathcal{R}}(\mathtt{p} \uparrow \langle S, \Pi \rangle . G', \mathcal{P}') = \mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset \rangle . \mathtt{tick}.\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathsf{Part}(G')}(G', \{\mathtt{r}\})$ by Fig. 11. Then, we have that (ii) $\mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset \rangle . \mathtt{tick}.\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathsf{Part}(G')}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p} = \mathtt{tick}.!S.(\mathsf{S}_{\mathsf{Part}(G')}(G, \{\mathtt{p}\}) \upharpoonright \mathtt{p})$ by Fig. 12 and (iii) $\mathtt{tick}.!S.(\mathsf{S}_{\mathsf{Part}(G')}(G, \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = \mathtt{tick}.!S.((\mathsf{S}_{\mathsf{Part}(G')}(G, \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}$ by Fig. 16 and Assumption. Also, by assumption, Fig. 12 and Fig. 16 we have (iv) $(\mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset \rangle . \mathtt{tick}.\mathtt{p} \uparrow \langle S, \Pi \rangle . \mathsf{S}_{\mathsf{Part}(G')}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = \mathtt{tick}.?(\mathtt{p}, S).((\mathsf{S}_{\mathsf{Part}(G')}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$. Finally, since $\{\mathtt{p}\} \subseteq \mathcal{P}'$ and the length of G' has decreased we can apply the IH and Def. 17 to show that duality holds.

Sub-case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \notin \Pi$**:** This proof proceeds similarly as above, while considering that $\mathtt{q}$ is not a receptor from the broadcast done by $\mathtt{p}$. Then, the case will conclude by Def. 17 and IH, since an output can be dual to any type.

Sub-case $\mathtt{r} = \mathtt{q} \wedge \mathtt{p} \in \Pi$**:** Symmetrical to Case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \in \Pi$.

Sub-case $\mathtt{r} = \mathtt{q} \wedge \mathtt{p} \notin \Pi$**:** Symmetrical to Case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \notin \Pi$.

Sub-case $\mathtt{p} \in \Pi \wedge \mathtt{q} \in \Pi$**:** In this case $\mathtt{r} \neq \mathtt{p} \wedge \mathtt{r} \neq \mathtt{q}$, by Definition. Notice also, that in this case both participants are receptors, and hence, by Fig. 16, the input prefix obtained by Fig. 12 will disappear, hence the case will conclude by the IH.

Sub-case $\mathtt{p} \notin \Pi \wedge \mathtt{q} \in \Pi$**:** Analogous to the case above.

Sub-case $\mathtt{p} \in \Pi \wedge \mathtt{q} \notin \Pi$**:** Analogous to the case above.

Sub-case $\mathtt{p} \notin \Pi \wedge \mathtt{q} \notin \Pi$**:** Analogous to the case above.

Case $\mathtt{r} \in \mathcal{P}$**:** Then $\mathsf{S}_{\mathcal{R}}(\mathtt{r} \uparrow \langle S, \Pi\rangle.G', \mathcal{P}') = \mathtt{r}_{\{1,n\}} \uparrow \langle S_d, \emptyset\rangle.\mathtt{tick}.\mathtt{r} \uparrow \langle S, \Pi\rangle.\mathsf{S}_{\mathsf{Part}(\mathtt{G}')}(G', \{\mathtt{r}\})$ by Fig. 11. The proof proceeds as shown above, the only difference being the fact that we are adding a tick to the global type by projection. Hence, our IH will consider $\mathsf{Part}(G')$ and $\{\mathtt{r}\}$ as parameters.

Case $G = \mathtt{pause}.G'$**:** This case is straightforward by applying the IH, since the size of the parameters of the saturation function will only affect the projections up to the number of outputs, which are dual with any type.

Case $G = \mu\mathbf{t}.G'$**:** There are two cases, but they are straightforward by applying the IH, notice that projections do not affect the recursion, so they only work internally in the body of the recursive type.

Case $G = \mathtt{watch}\ ev\ \mathtt{do}\ G'\ \mathtt{else}\ G''$**:** This case again is straightforward by applying the inductive hypothesis. Similarly to the recursive type, we have to consider that the IH is applied to both the main body and alternative body. In the alternative body, it is necessary to consider that the size of the parameters set will not affect duality.

□

Proposition 2. *Let G be a global type and $\mathtt{p} \neq \mathtt{q}$. Then $(G\lfloor_{\mathtt{p}}) \upharpoonright \mathtt{q} \bowtie (G\lfloor_{\mathtt{q}}) \upharpoonright \mathtt{p}$.*

Proof. By induction on G. Let $\mathtt{p}, \mathtt{q} \in \mathsf{Part}(G)$ and let $\mathsf{Part}(G) = \mathcal{R}$.

Case $G = \mathsf{end}$**:**

(1)	$(\mathsf{end}\lfloor_{\mathtt{p}}) \upharpoonright \mathtt{q} = (\mathsf{S}_{\mathcal{R}}(\mathsf{end}, \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}$	(Fig. 12, Fig. 11)
(2)	$(\mathsf{end}\lfloor_{\mathtt{q}}) \upharpoonright \mathtt{p} = (\mathsf{S}_{\mathcal{R}}(\mathsf{end}, \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$	(Fig. 12 and Fig. 11)
(3)	$\mathsf{Part}(\mathsf{end}) = \mathcal{R} = \emptyset$	(Definition of $\mathsf{Part}(G)$)
(4)	$\mathcal{R} \setminus \mathcal{P} = \emptyset$	(algebra of sets)
(5)	$\mathsf{S}_{\mathcal{R}}(\mathsf{end}, \emptyset) = \mathsf{S}_{\emptyset}(\mathsf{end}, \emptyset) = \mathsf{end}$	(Fig. 11)
(6)	$(\mathsf{S}_{\emptyset}(\mathsf{end}, \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = (\mathsf{end} \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$	((5), Fig. 12, Fig. 16)
(7)	$(\mathsf{S}_{\emptyset}(\mathsf{end}, \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p} = (\mathsf{end} \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p} = \mathsf{end} \upharpoonright \mathtt{p} = \mathsf{end}$	((5), Fig. 12, Fig. 16)
(8)	$\mathsf{end} \bowtie \mathsf{end}$	(Def. 17)

Case $G = \mathbf{t}$**:** As above.

Case $G = \mathtt{r} \uparrow \langle S, \Pi\rangle.G'$**:** We have that $(G'\lfloor_{\mathtt{p}}) \upharpoonright \mathtt{q} \bowtie (G'\lfloor_{\mathtt{q}}) \upharpoonright \mathtt{p}$ by IH. Moreover, $(\mathsf{S}_{\mathsf{Part}(\mathtt{G}')}(G', \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathsf{Part}(\mathtt{G}')}(G', \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ by Fig. 12 and Fig. 11. Then, We distinguish cases depending on $\mathtt{r}$ and the memberships of $\mathtt{p}, \mathtt{q}$ in Π. There are eight cases:

Case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \in \Pi$**:** Then $G = \mathtt{p} \uparrow \langle S, \Pi\rangle.G'$:

(i)	$\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(\mathtt{p} \uparrow \langle S, \Pi\rangle.G', \emptyset) = \mathtt{p} \uparrow \langle S, \Pi\rangle.\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\})$	(Fig. 11)
(ii)	$\mathtt{p} \uparrow \langle S, \Pi\rangle.\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p} = !S.(\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p})$	(Fig. 12)
(iii)	$!S.(\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = !S.((\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q})$	(Fig. 16)
(iv)	$\mathtt{p} \uparrow \langle S, \Pi\rangle.\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q} = ?(\mathtt{p}, S).(\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q})$	(Fig. 12)
(v)	$?(\mathtt{p}, S).(\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p} = ?S.((\mathsf{S}_{\mathsf{Part}(\mathtt{G})}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p})$	(Fig. 16)

We conclude by Lemma 6 and IH that $(\mathsf{S}_{\mathsf{Part}(\mathtt{G}')}(G', \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathsf{Part}(\mathtt{G}')}(G', \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$

implies $!S.((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}) \bowtie ?S.((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p})$ which is the same as saying $\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \bowtie \mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\})$.

Case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \notin \Pi$**:** Then $G = \mathtt{p} \upharpoonright \langle S, \Pi \rangle.G'$:

(i) $\mathsf{S}_{\mathsf{Part(G)}}(\mathtt{p} \upharpoonright \langle S, \Pi \rangle.G', \emptyset) = \mathtt{p} \upharpoonright \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\})$ (Fig. 11)
(ii) $\mathtt{p} \upharpoonright \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p} = !S.(\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p})$ (Fig. 12)
(iii) $!S.(\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = !S.((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q})$ (Fig. 16)
(iv) $\mathtt{p} \upharpoonright \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q} = \mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}$ (Fig. 12 and Assumption)
(v) $(\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p} = (\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ (Fig. 16)

We then conclude by Lemma 6 since $(\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ implies $!S.((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}) \bowtie ((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p})$ which means that $\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \bowtie \mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\})$.

Case $\mathtt{r} = \mathtt{q} \wedge \mathtt{p} \in \Pi$**:** Symmetrical to Case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \in \Pi$.

Case $\mathtt{r} = \mathtt{q} \wedge \mathtt{p} \notin \Pi$**:** Symmetrical to Case $\mathtt{r} = \mathtt{p} \wedge \mathtt{q} \notin \Pi$.

Case $\mathtt{p} \in \Pi \wedge \mathtt{q} \in \Pi$**:** In this case $\mathtt{r} \neq \mathtt{p} \wedge \mathtt{r} \neq \mathtt{q}$, hence:

(i) $\mathsf{S}_{\mathsf{Part(G)}}(\mathtt{r} \upharpoonright \langle S, \Pi \rangle.G', \emptyset) = \mathtt{r} \upharpoonright \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{r}\})$ (Fig. 11)
(ii) $\mathtt{r} \upharpoonright \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p} = ?(\mathtt{r}, S).\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}$ (Fig. 12 and Assumption)
(iii) $?(\mathtt{r}, S).(\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} = (\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}$ (Fig. 16)
(iv) $\mathtt{p} \upharpoonright \langle S, \Pi \rangle.\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q} = ?(\mathtt{r}, S).(\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q})$ (Fig. 12)
(v) $?(\mathtt{p}, S).(\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p} = (\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ (Fig. 16)

We then conclude by Lemma 6, since $(\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ implies $((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{r}\}) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}) \bowtie ((\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{p}\}) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p})$ which means that $\mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{r}\}) \bowtie \mathsf{S}_{\mathsf{Part(G)}}(G', \{\mathtt{r}\})$.

Case $\mathtt{p} \notin \Pi \wedge \mathtt{q} \in \Pi$**:** Analogous to the case above – concludes by Lemma 6.

Case $\mathtt{p} \in \Pi \wedge \mathtt{q} \notin \Pi$**:** Analogous to the case above – concludes by Lemma 6.

Case $\mathtt{p} \notin \Pi \wedge \mathtt{q} \notin \Pi$**:** Analogous to the case above – concludes by Lemma 6.

Case $G = \mathtt{pause}.G'$**:**

(1) $(\mathtt{pause}.G' \lfloor_{\mathtt{p}}) \upharpoonright \mathtt{q} = (\mathsf{S}_{\mathcal{R}}(\mathtt{pause}.G', \emptyset) \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q}$ (Fig. 12 and Fig. 11)
(2) $(\mathtt{pause}.G' \lfloor_{\mathtt{q}}) \upharpoonright \mathtt{p} = (\mathsf{S}_{\mathcal{R}}(\mathtt{pause}.G', \emptyset) \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$ (Fig. 12 and Fig. 11)
(3) $\mathsf{S}_{\mathsf{Part(G)}}(\mathtt{pause}.G', \emptyset) = \mathtt{r}_{\{1,n\}} \upharpoonright \langle S_d, \emptyset \rangle.\mathtt{pause}.\mathsf{S}_{\mathsf{Part(G')}}(G', \emptyset)$ (Fig. 11)

We then distinguish cases depending on the presence of $\mathtt{p}, \mathtt{q}$ in $\{\mathtt{r}_1, \ldots \mathtt{r}_n\}$. There are four cases:

Case $\mathtt{p}, \mathtt{q} \in \{\mathtt{r}_1, \ldots \mathtt{r}_n\}$**:** This case follows from the Lemma 6.

Case $\mathtt{p} \in \{\mathtt{r}_1, \ldots \mathtt{r}_n\} \wedge \mathtt{q} \notin \{\mathtt{r}_1, \ldots \mathtt{r}_n\}$**:** This case follows from the Lemma 6.

Case $\mathtt{p} \notin \{\mathtt{r}_1, \ldots \mathtt{r}_n\} \wedge \mathtt{q} \in \{\mathtt{r}_1, \ldots \mathtt{r}_n\}$**:** This case follows from the Lemma 6.

Case $\mathtt{p}, \mathtt{q} \notin \{\mathtt{r}_1, \ldots \mathtt{r}_n\}$**:**

Case $G = \mu \mathbf{t}.G'$**:** There are two cases, but they proceed straightforward by the IH. Note that $\mathsf{Part}(\mu \mathbf{t}.G') = \mathsf{Part}(G')$.

Case $G = \mathtt{watch}\ ev\ \mathtt{do}\ G'\ \mathtt{else}\ G''$**:** This case follows by Lemma 6. This is because the duality of a watch operator depends on the duality of its components.

□

We are now finally ready to introduce the typing rules for configurations. First of all, we mention that our type system admits the following weakening and contraction rules:

$$\lfloor \textsc{Weak} \rfloor \ \frac{\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta \rangle}{\Gamma \vdash C \triangleright \langle \Delta, c : \mathsf{end} \diamond \Theta \rangle} \qquad \lfloor \textsc{Contr} \rfloor \ \frac{\Gamma \vdash C \triangleright \langle \Delta, c : \mathsf{end} \diamond \Theta \rangle}{\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta \rangle}$$

$$\lfloor \textsc{RVar} \rfloor \frac{\Gamma \vdash X : \mathsf{pause}.T \quad \Gamma \vdash M \triangleright \Theta}{\Gamma \vdash \langle X, M, E\rangle \triangleright \langle c : \mathsf{pause}.T \diamond \Theta\rangle} \qquad \lfloor \textsc{Inact} \rfloor \frac{\Delta = \Delta^{end} \quad \Gamma \vdash M \triangleright \Theta}{\Gamma \vdash \langle \mathbf{0}, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle}$$

$$\lfloor \textsc{MInit} \rfloor \frac{\Gamma \vdash a \triangleright G \quad |\mathsf{Part}(G)| = n}{\Gamma \vdash \langle \bar{a}[n], \emptyset, \emptyset\rangle \triangleright \langle \emptyset \diamond \emptyset\rangle} \qquad \lfloor \textsc{MAcc} \rfloor \frac{\Gamma \vdash a \triangleright G \quad \Gamma \vdash \langle P\{s[\mathtt{p}]/\alpha\}, M_s^{\emptyset}, \emptyset\rangle \triangleright \langle s[\mathtt{p}] : G\lfloor_{\mathtt{p}} \diamond \Theta_s^{\emptyset}\rangle}{\Gamma \vdash \langle a[\mathtt{p}](\alpha).P, \emptyset, \emptyset\rangle \triangleright \langle \emptyset \diamond \emptyset\rangle}$$

$$\lfloor \textsc{Conc} \rfloor \frac{\Gamma \vdash \langle P_i, M, E\rangle \triangleright \langle \Delta_i \diamond \Theta\rangle,\ i = 1,2}{\Gamma \vdash \langle P_1 \mid P_2, M, E\rangle \triangleright \langle \Delta_1, \Delta_2 \diamond \Theta\rangle} \qquad \lfloor \textsc{CRes} \rfloor \frac{\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta\rangle \quad Co\,\langle \Delta \diamond \Theta\rangle}{\Gamma \vdash (\nu s)C \triangleright \langle \emptyset \diamond \emptyset\rangle}$$

$$\lfloor \textsc{Emit} \rfloor \frac{\Gamma \vdash \langle P, M, E \cup ev\rangle \triangleright \langle \Delta \diamond \Theta\rangle}{\Gamma \vdash \langle \mathtt{emit}\ ev.P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle} \qquad \lfloor \textsc{Pause} \rfloor \frac{\Gamma \vdash M \triangleright \Theta \quad \Gamma \vdash \langle P, M^{\emptyset}, \emptyset\rangle \triangleright \langle c : T \diamond \Theta^{\emptyset}\rangle \quad \mathsf{OG}(T)}{\Gamma \vdash \langle \mathtt{pause}.P, M, E\rangle \triangleright \langle c : \mathsf{pause}.T \diamond \Theta\rangle}$$

$$\lfloor \textsc{Rec} \rfloor \frac{\begin{array}{c}\Gamma, X : \mathsf{pause}.T \vdash \langle P, M^{\emptyset}, \emptyset\rangle \triangleright \langle c : T \diamond \Theta^{\emptyset}\rangle \\ \Gamma, X : \mathsf{pause}.T \vdash \langle P, M, E\rangle \triangleright \langle c : T \diamond \Theta\rangle\end{array}}{\Gamma \vdash \langle (\mathtt{rec}\ X.P), M, E\rangle \triangleright \langle c : T \diamond \Theta\rangle}$$

$$\lfloor \textsc{SendFirst} \rfloor \frac{\Gamma \vdash e : S \quad \Gamma \vdash \langle P, M \cup s[\mathtt{p}] : (d_S, \emptyset), E\rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta, s[\mathtt{p}] : (S, \emptyset)\rangle}{\Gamma \vdash \langle s[\mathtt{p}]!\langle e\rangle.P, M \cup s[\mathtt{p}] : \epsilon, E\rangle \triangleright \langle s[\mathtt{p}] :!S.T \diamond \Theta, s[\mathtt{p}] : \mathtt{void}\rangle}$$

$$\lfloor \textsc{SendMore} \rfloor \frac{\begin{array}{c}\Gamma \vdash M \triangleright \Theta \quad \Gamma \vdash e : S \quad \Gamma \vdash v : S' \quad \mathsf{OG}(T) \\ \Gamma \vdash \langle P, M^{\emptyset} \cup s[\mathtt{p}] : (d_S, \emptyset), \emptyset\rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta^{\emptyset}, s[\mathtt{p}] : (S, \emptyset)\rangle\end{array}}{\Gamma \vdash \langle s[\mathtt{p}]!\langle e\rangle.P, M \cup s[\mathtt{p}] : (v, \Pi), E\rangle \triangleright \langle s[\mathtt{p}] : \mathtt{tick}.!S.T \diamond \Theta, s[\mathtt{p}] : (S', \Pi)\rangle}$$

$$\lfloor \textsc{RcvFirst} \rfloor \frac{\Gamma, x : S \vdash \langle P, M \cup s[\mathtt{p}] : (v, \Pi \cup \mathtt{q}), E\rangle \triangleright \langle s[\mathtt{q}] : T \diamond \Theta, s[\mathtt{p}] : (S, \Pi \cup \mathtt{q})\rangle}{\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P, M \cup s[\mathtt{p}] : (v, \Pi), E\rangle \triangleright \langle s[\mathtt{q}] :?(\mathtt{p}, S).T \diamond \Theta, s[\mathtt{p}] : (S, \Pi)\rangle}$$

$$\lfloor \textsc{RcvMore} \rfloor \frac{\begin{array}{c}\Gamma \vdash M \triangleright \Theta \quad \mathtt{q} \in \Pi \quad \Gamma \vdash v : S' \quad \mathsf{OG}(T) \\ \Gamma, x : S \vdash \langle P, M^{\emptyset} \cup s[\mathtt{p}] : (d_S, \{\mathtt{q}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : T \diamond \Theta^{\emptyset}, s[\mathtt{p}] : (S, \{\mathtt{q}\})\rangle\end{array}}{\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P, M \cup s[\mathtt{p}] : (v, \Pi), E\rangle \triangleright \langle s[\mathtt{q}] : \mathtt{tick}.?(\mathtt{p}, S).T \diamond \Theta, s[\mathtt{p}] : (S', \Pi)\rangle}$$

$$\lfloor \textsc{RcvNext} \rfloor \frac{\Gamma, x : S \vdash \langle P, M \cup s[\mathtt{p}] : (v, \{\mathtt{q}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : T \diamond \Theta, s[\mathtt{p}] : (S, \{\mathtt{q}\})\rangle}{\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P, M \cup s[\mathtt{p}] : \epsilon, E\rangle \triangleright \langle s[\mathtt{q}] :?(\mathtt{p}, S).T \diamond \Theta, s[\mathtt{p}] : \mathtt{void}\rangle}$$

$$\lfloor \textsc{Watch} \rfloor \frac{\begin{array}{c}\Gamma \vdash \langle P, M, E\rangle \triangleright \langle s[\mathtt{p}] : T_P \diamond \Theta\rangle \\ T_P = \mathsf{end} \vee (\Gamma \vdash \langle Q, M^{\emptyset}, \emptyset\rangle \triangleright \langle s[\mathtt{p}] : T_Q \diamond \Theta^{\emptyset}\rangle \wedge \mathsf{OG}(T_Q))\end{array}}{\Gamma \vdash \langle \mathtt{watch}\ ev\ \mathtt{do}\ P\{Q\}, M, E\rangle \triangleright \langle s[\mathtt{p}] : T_P \star_{ev} T_Q \diamond \Theta\rangle}$$

$$\lfloor \textsc{If} \rfloor \frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle \quad \Gamma \vdash \langle Q, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle}{\Gamma \vdash \langle \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle}$$

Figure 17: Typing rules for configurations.

These structural rules will allow us to add and remove fresh channels with empty behaviors in the session environment. They are included to allow simplifications on some of the typing rules in Fig. 17, which we briefly describe below:

- Rule ⌊RVAR⌋ types a process variable inside a configuration. It asks that environment Γ contains the required process variable and that the current memory is typable.
- Rule ⌊INACT⌋ types a terminated configuration with any session environment containing only elements of the form $c : \mathtt{end}$ or $c : \langle \mathtt{end}, T \rangle^{ev}$ (condition $\Delta = \Delta^{end}$).
- Rule ⌊EMIT⌋ has no effect on types, as event emission only affects the communication behaviour via the watching construct, and the event set is not typed.
- Rules ⌊MINIT⌋ and ⌊MACC⌋ require the standard environment to associate a global type G with the service identifier a. Moreover, the last premise of Rule ⌊MACC⌋ guarantees that the type of $\mathtt{p}$'s channel in P is obtained as the $\mathtt{p}$-th saturated projection of G.
- Rule ⌊PAUSE⌋ types a paused configuration, checking that the reconditioned configuration is well-typed, and requiring that the reconditioned session environment is output-granting. The latter condition will be required in all rules for suspended configurations.
- Rule ⌊SENDFIRST⌋ types the first broadcast by participant $\mathtt{p}$ in the current instant (the fact that it is the first is indicated by the presence of $s[\mathtt{p}] : \epsilon$ in the memory). If S is the sort of the broadcast value, the continuation P is required to be typable in the memory obtained by replacing $s[\mathtt{p}] : \epsilon$ with $s[\mathtt{p}] : (d_S, \emptyset)$, where d_S is the default value of sort S. This amounts to say that the continuation P must be typable in the memory just after the broadcast.
- Rule ⌊SENDMORE⌋ types the broadcast of the value of an expression e when the sender $\mathtt{p}$ has already sent some value v in the current instant (as witnessed by the presence of $s[\mathtt{p}] : (v, \Pi)$ in the memory). In this case, we insert a `tick` in front of the send type of $s[\mathtt{p}]$, to indicate that the new broadcast will take place at the next instant. The continuation P must be typable in the refreshed memory $M^{\emptyset}$ updated with the new message sent by $\mathtt{p}$, and the reconditioned environment must be output-granting.
- Rule ⌊RCVFIRST⌋ is analogous to Rule ⌊SENDFIRST⌋: it types an input by receiver $\mathtt{q}$ from sender $\mathtt{p}$ if a message sent by $\mathtt{p}$ is present in the memory, and this message has not been read yet by $\mathtt{q}$ (namely, $\mathtt{q}$ is not in its set of Readers Π). The continuation P needs to be typable in the updated memory.
- Rule ⌊RCVMORE⌋ is analogous to Rule ⌊SENDMORE⌋: it types an input by receiver $\mathtt{q}$ from sender $\mathtt{p}$ in case $\mathtt{q}$ has already read a message v from $\mathtt{p}$ in the current instant. In this case, a `tick` is inserted in front of the receive type of channel $s[\mathtt{q}]$. Again, the reconditioned environment must be output-granting.
- Rule ⌊RCVNEXT⌋ is used to type an input by receiver $\mathtt{q}$ from sender $\mathtt{p}$ at the start of a new instant, when no participant has sent any message yet. This rule is very similar to rule ⌊RCVFIRST⌋, except that the output buffer of the sender is empty, and therefore it is typed with `void`.
- Rule ⌊CONC⌋ types the parallel composition of two processes with a given memory and set of events, provided both components are typable with such memory and set of events and the obtained session environments have disjoint domains.
- Rule ⌊CRES⌋ types a restricted configuration with the empty configuration environment provided the session environment of its body is coherent. This is the only typing rule that requires coherence.

- Rule ⌊IF⌋ requires, as usual in session type systems, that the two branches of the conditional be typed with the same session environment Δ.
- Rule ⌊WATCH⌋ types the watching construct by creating a pair of types $\langle T, T' \rangle^{ev}$ for each participant. Since the alternative process Q may only be launched at the start of a new instant, its session environment is required to be output-granting.
- Rule ⌊REC⌋ types recursion, requiring each call to be isolated in its own time instant. To this end, the type declaration $X : \texttt{pause}.T$ is added to Γ.

4.4 Properties

The semantic properties P1 and P2 defined previously both rely on subject reduction (SR). To prove SR, we need some preliminary definitions and results. The first property we prove ensures that the typing of configurations ensures the typing of memories:

Lemma 7. *If* $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ *then* $\Gamma \vdash M \triangleright \Theta$.

Proof. By induction on the height of the typing derivation $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$.

Base Cases:

Case 1. *Rule* ⌊RVAR⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ with Rule ⌊RVAR⌋. Then, by inversion, we have $\Gamma \vdash M \triangleright \Theta$, concluding the proof.

Case 2. *Rule* ⌊INACT⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ with Rule ⌊INACT⌋. Then, by inversion, we have $\Gamma \vdash M \triangleright \Theta$, concluding the proof.

Case 3. *Rule* ⌊MINIT⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ with Rule ⌊MINIT⌋. Notice that the memories are empty. Hence, the judgement $\Gamma \vdash M \triangleright \Theta$ holds by Rule ⌊EMPTYMEM⌋ in Fig. 15.

Case 4. *Rule* ⌊MACC⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ with Rule ⌊MACC⌋. Notice that the memories are empty. Hence, the judgement $\Gamma \vdash M \triangleright \Theta$ holds by Rule ⌊EMPTYMEM⌋ in Fig. 15.

Inductive Cases:

Case 1. *Rule* ⌊WEAK⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ with Rule ⌊WEAK⌋. By inversion, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta, c : \mathsf{end} \diamond \Theta \rangle$. Then, by IH, $\Gamma \vdash M \triangleright \Theta$.

Case 2. *Rule* ⌊CONTR⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ with Rule ⌊CONTR⌋. By inversion, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \setminus c : \mathsf{end} \diamond \Theta \rangle$. Then, by IH, $\Gamma \vdash M \triangleright \Theta$.

Case 3. *Rule* ⌊CONC⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊CONC⌋. By inversion, $P = P_1 \mid P_2$, $\Gamma \vdash \langle P_1, M, E\rangle \triangleright \langle \Delta_1, c : \mathsf{end} \diamond \Theta\rangle$, and $\Gamma \vdash \langle P_1, M, E\rangle \triangleright \langle \Delta_1, c : \mathsf{end} \diamond \Theta\rangle$, with $\Delta = \Delta_1, \Delta_2$. Then, by IH, $\Gamma \vdash M \triangleright \Theta$, concluding the proof.

Case 4. *Rule* ⌊EMIT⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊EMIT⌋. As above, this case concludes by applying inversion and the IH.

Case 5. *Rule* ⌊PAUSE⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊PAUSE⌋. By inversion, we have that $\Gamma \vdash M \triangleright \Theta$, which is what we wanted to prove.

Case 6. *Rule* ⌊REC⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊REC⌋. By inversion, (1) $\Gamma, X : \mathtt{pause}.T \vdash \langle P, M^{\emptyset}, \emptyset\rangle \triangleright \langle c : T \diamond \Theta^{\emptyset}\rangle$ and (2) $\Gamma, X : \mathtt{pause}.T \vdash \langle P, M, E\rangle \triangleright \langle c : T \diamond \Theta\rangle$. Then, by applying the IH on (2), we conclude that $\Gamma, X : \mathtt{pause}.T \vdash M \triangleright \Theta$. Since $X$is not used in typing M it implies that $\Gamma \vdash M \triangleright \Theta$, which allows us to conclude this case.

Case 7. *Rule* ⌊SENDFIRST⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊SENDFIRST⌋. From the rule it can be deduced that $M = M' \cup s[p] : \epsilon$, $\Theta = \Theta', s[\mathtt{p}] : \mathtt{void}$. Then, by inversion on the Rule, (1) $\Gamma \vdash \langle P, M' \cup s[\mathtt{p}] : (d_S, \emptyset), E\rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta', s[\mathtt{p}] : (S, \emptyset)\rangle$. From IH applied to (1), we have that (2) $\Gamma \vdash M' \cup s[\mathtt{p}] : (d_S, \emptyset) \triangleright \Theta', s[\mathtt{p}] : (S, \emptyset)$. Notice that the judgement in (2) can only be deduced from Rule ⌊MERGEMEM⌋ in Fig. 15, thus, by inversion we have that (3) $\Gamma \vdash M' \triangleright \Theta'$. Then, applying Rule ⌊VOIDMSG⌋, we have that (4) $\Gamma \vdash s[\mathtt{p}] : \epsilon \triangleright s[\mathtt{p}] : \mathtt{void}$. Thus, applying Rule ⌊MERGEMEM⌋, we can conclude that $\Gamma \vdash M' \cup s[\mathtt{p}] : \epsilon \triangleright \Theta', s[\mathtt{p}] : \mathtt{void}$, which concludes the proof.

Case 8. *Rules* ⌊SENDMORE⌋*,* ⌊RCVFIRST⌋*,* ⌊RCVMORE⌋*, and* ⌊RCVNEXT⌋*:*
The proof for each of these cases proceeds similarly as above, by using inversion, applying the IH and using the rules in Fig. 15, to deduce tha the memory is typable.

Case 9. *Rule* ⌊WATCH⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊WATCH⌋. By inversion, we have that $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle s[\mathtt{p}] : T_P \diamond \Theta\rangle$. Then, we conclude by applying the IH.

Case 10. *Rule* ⌊IF⌋*:*
By assumption, $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ with Rule ⌊IF⌋. The proof proceeds by inversion and applying the IH.

□

Before proving subject reduction and since the configuration environment changes along execution, we now formalise a notion of reduction for configuration environments.

Definition 19 (Reduction of configuration environments)**.** *Let $\Rightarrow$ be the reflexive and transitive relation on configuration environments generated by:*

1. $\langle \Delta, s[\mathtt{p}] : !S.T \diamond \Theta, s[\mathtt{p}] : \mathtt{void} \rangle \;\Rightarrow\; \langle \Delta, s[\mathtt{p}] : T \diamond \Theta, s[\mathtt{p}] : (S, \emptyset) \rangle$
2. $\langle \Delta, s[\mathtt{q}] : ?(\mathtt{p}, S).T \diamond \Theta, s[\mathtt{p}] : (S, \Pi) \rangle \;\Rightarrow\; \langle \Delta, s[\mathtt{q}] : T \diamond \Theta, s[\mathtt{p}] : (S, \Pi \cup \{\mathtt{q}\}) \rangle$ *with* $\mathtt{q} \notin \Pi$
3. $\langle \Delta \diamond \Theta \rangle \;\Rightarrow\; \langle \Delta' \diamond \Theta' \rangle$ *implies* $\langle \Delta \star_{ev} \Delta'' \diamond \Theta \rangle \;\Rightarrow\; \langle \Delta' \star_{ev} \Delta'' \diamond \Theta' \rangle$
4. $\langle \Delta \diamond \Theta \rangle \;\Rightarrow\; \langle \Delta' \diamond \Theta' \rangle$ *implies* $\langle \Delta, \Delta'' \diamond \Theta \rangle \;\Rightarrow\; \langle \Delta', \Delta'' \diamond \Theta' \rangle$

Similarly, we introduce a tick reduction for the configuration environments of suspended configurations. This reduction relation is parameterised by the set of events E which is used to recondition the watch types.

We now argue that it is important for us to distinguish between implicit and explicit pauses in our typing, specifically, when aiming to prove a subject reduction result. In particular, consider

$$C = \langle \mathcal{E}[s[\mathtt{p}]?(\mathtt{q}, x).s[\mathtt{p}]?(\mathtt{r}, y).P], M, E \rangle$$

with $s[\mathtt{r}] : (v, \Pi) \in M$, $\mathtt{p} \in \Pi$ and $(vs)C\ddagger$. Using the typing rules in Fig. 17, we can see that:

$$\Gamma \vdash C \triangleright \langle \Delta, s[\mathtt{p}] : ?(\mathtt{q}, S).\mathtt{tick}.?(\mathtt{r}, S).T \diamond \Theta \rangle$$

Now, since C is suspended, the configuration reduces, via a tick transition (cf. Fig. 9), to $(vs)D$ with

$$D = \langle \mathcal{E}[s[\mathtt{p}]?(\mathtt{q}, x).s[\mathtt{p}]?(\mathtt{r}, y).P], M^{\emptyset}, \emptyset \rangle.$$

whose typing is:

$$\Gamma \vdash D \triangleright \langle \Delta', s[\mathtt{p}] : ?(\mathtt{q}, S).?(\mathtt{r}, S).T \diamond \emptyset \rangle$$

It is clear that in the above typing judgement, the `tick` has been deleted. Thus, it is necessary for us to distinguish between implicit and explicit pauses when defining a reduction for typing environments, as some of the implicit pauses may be removed. Formally, this situation happens whenever a suspension occurs because of rule (in_s) (cf. Fig. 7). We address the deletion of `tick`, using the $\mathtt{trm}(\cdot)$ function defined in Fig. 18. Intuitively, $\mathtt{trm}(\cdot)$ parses the local type and recalculates where to put implicit pauses.

Definition 20 (Tick reduction of configuration environments)**.** *Let $\curvearrowright_E$ be the parameterised relation on configuration environments generated by:*

1. *(Pause)* $\langle \mathsf{pause}(\Delta) \diamond \Theta \rangle \curvearrowright_{\emptyset} \langle \Delta \diamond \Theta^{\emptyset} \rangle$
2. *(Tick)* $\langle \mathsf{tick}(\Delta) \diamond \Theta \rangle \curvearrowright_{\emptyset} \langle \Delta \diamond \Theta^{\emptyset} \rangle$
3. *(In)* $\Theta = \Theta', s[\mathtt{p}] : \mathtt{void}$ *implies : for any* E $\langle s[\mathtt{q}] : ?(\mathtt{p}, S).T \diamond \Theta \rangle \curvearrowright_E \langle s[\mathtt{q}] : ?(\mathtt{p}, S).\mathtt{trm}(T)^0_{\{\mathtt{p}\}} \diamond \Theta^{\emptyset} \rangle$ *where* $\mathtt{trm}(\cdot)$ *is as in Fig. 18*
4. *(Par)* $\langle \Delta_i \diamond \Theta \rangle \curvearrowright_E \langle \Delta_i' \diamond \Theta' \rangle$, $i = 1, 2$ *implies* $\langle \Delta_1, \Delta_2 \diamond \Theta \rangle \curvearrowright_E \langle \Delta_1', \Delta_2' \diamond \Theta' \rangle$
5. *(Restr)* $\langle \Delta \diamond \Theta \rangle \curvearrowright_E \langle \Delta' \diamond \Theta' \rangle$ *implies* $\langle \Delta \setminus s \diamond \Theta \setminus s \rangle \curvearrowright_E \langle \Delta' \setminus s \diamond \Theta' \setminus s \rangle$
6. *(Watch)* $\langle \Delta_1 \diamond \Theta \rangle \curvearrowright_E \langle \Delta_1' \diamond \Theta' \rangle$ *implies* $\begin{cases} \langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle \curvearrowright_{E \cup \{ev\}} \langle \Delta_2 \diamond \Theta' \rangle & \text{if } ev \in E \\ \langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle \curvearrowright_{E \cup \{ev\}} \langle \Delta_1' \star_{ev} \Delta_2 \diamond \Theta' \rangle & \text{if } ev \notin E \end{cases}$

$$\mathtt{trm}(T)^0_\Pi = \begin{cases} !S.T' & \text{if } T = \mathtt{tick}.!S.T' \\ !S.\mathtt{trm}(T')^1_\Pi & \text{if } T = !S.T' \\ \mathtt{tick}.?(\mathtt{p},S).T' & \text{if } T = \mathtt{tick}.?(\mathtt{p},S).T' \text{ and } \mathtt{p} \in \Pi \\ ?(\mathtt{p},S).T' & \text{if } T = \mathtt{tick}.?(\mathtt{p},S).T' \text{ and } \mathtt{p} \notin \Pi \\ ?(\mathtt{p},S).\mathtt{trm}(T')^0_{\Pi\cup\{\mathtt{p}\}} & \text{if } T = ?(\mathtt{p},S).T' \text{ and } \mathtt{p} \notin \Pi \\ \langle \mathtt{trm}(T_1)^0_\Pi, T_2\rangle^{ev} & \text{if } T = \langle T_1, T_2\rangle^{ev} \\ \mu\mathbf{t}.T' & \text{if } T = \mu\mathbf{t}.T' \\ \mathtt{pause}.T' & \text{if } T = \mathtt{pause}.T' \\ \mathtt{end} & \text{if } T = \mathtt{end} \end{cases}$$

$$\mathtt{trm}(T)^1_\Pi = \begin{cases} \mathtt{tick}.!S.T' & \text{if } T = \mathtt{tick}.!S.T' \\ \mathtt{tick}.?(\mathtt{p},S).T' & \text{if } T = \mathtt{tick}.?(\mathtt{p},S).T' \text{ and } \mathtt{p} \in \Pi \\ ?(\mathtt{p},S).T' & \text{if } T = \mathtt{tick}.?(\mathtt{p},S).T' \text{ and } \mathtt{p} \notin \Pi \\ ?(\mathtt{p},S).\mathtt{trm}(T')^1_{\Pi\cup\{\mathtt{p}\}} & \text{if } T = ?(\mathtt{p},S).T' \text{ and } \mathtt{p} \notin \Pi \\ \langle \mathtt{trm}(T_1)^1_\Pi, T_2\rangle^{ev} & \text{if } T = \langle T_1, T_2\rangle^{ev} \\ \mu\mathbf{t}.T' & \text{if } T = \mu\mathbf{t}.T' \\ \mathtt{pause}.T' & \text{if } T = \mathtt{pause}.T' \\ \mathtt{end} & \text{if } T = \mathtt{end} \end{cases}$$

Figure 18: trm() function

We also let $\langle \Delta \diamond \Theta\rangle \curvearrowright \langle \Delta' \diamond \Theta'\rangle$ *if there exists* E *such that* $\langle \Delta \diamond \Theta\rangle \curvearrowright_E \langle \Delta' \diamond \Theta'\rangle$.
The predicate $\langle \Delta \diamond \Theta\rangle \curvearrowright_E$ *is defined by* $\langle \Delta \diamond \Theta\rangle \curvearrowright_E$ *if there exist* Δ', Θ *such that* $\langle \Delta \diamond \Theta\rangle \curvearrowright_E \langle \Delta' \diamond \Theta'\rangle$, *and the predicate* $\langle \Delta \diamond \Theta\rangle \curvearrowright$ *is defined similarly.*

Note that $\curvearrowright$ is defined also on the environments of suspended subconfigurations, which cannot perform a tick transition.

Just like processes, local types and session environments should be reconditioned to properly reflect the reconditioning of processes during tick reductions:

Definition 21 (Reconditioning of local types and session environments)**.** *Given a set of events* E*, the* reconditioning *of a local type* H *(under* E*), written* $[T]_E$*, is defined as:*

$$[T]_E = \begin{cases} T' & \text{if } T = \mathtt{pause}.T' \text{ or } T = \mathtt{tick}.T' \\ T & \text{if } T = \mathtt{end} \text{ or } T = \boldsymbol{t} \\ ?(\mathtt{p},S).\mathtt{trm}(T')^0_{\{\mathtt{p}\}} & \text{if } T = ?(\mathtt{p},S).T' \\ T_2 & \text{if } T = \langle T_1, T_2\rangle^{ev} \text{ and } ev \in E \\ \langle [T_1]_E, T_2\rangle^{ev} & \text{if } T = \langle T_1, T_2\rangle^{ev} \text{ and } ev \notin E \end{cases}$$

where $\mathtt{trm}(\cdot)$ *is defined in Fig. 18. Given* Δ*, its reconditioning under* E *is defined as* $[\Delta]_E = \{c : [T]_E \mid c : T \in \Delta\}$.

The following proposition is easy to show, by induction on Definition 20:

Proposition 3. *If* $\langle \Delta \diamond \Theta\rangle \curvearrowright_E \langle \Delta' \diamond \Theta'\rangle$ *then* $\Delta' = [\Delta]_E$ *and* $\Theta' = \Theta^\emptyset$.

Definition 22 (Mixed reduction of configuration environments)**.** *Let* $\approx\!\!>$ *stand for either* $\Rightarrow$ *or* $\curvearrowright$*, and let* $\approx\!\!>^*$ *be its reflexive and transitive closure.*

Hence $\langle \Delta \diamond \Theta \rangle \approx\!\!> \langle \Delta' \diamond \Theta' \rangle$ when $\langle \Delta \diamond \Theta \rangle$ can reduce to $\langle \Delta' \diamond \Theta' \rangle$ via a mixed sequence of reductions. We prove now that types are preserved under $\equiv$ (i.e., *subject congruence*) and substitution. Notice that Rule ⌊WEAK⌋ is necessary for subject congruence, as it allows to add ended sessions to Δ.

Lemma 8 (Subject Congruence)**.** *If* $\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta \rangle$ *and* $C \equiv C'$ *then* $\Gamma \vdash C' \triangleright \langle \Delta \diamond \Theta \rangle$.

Proof. By induction on the definition of structural congruence (cf. Fig. 5). There are two cases:

Case 1. *Rule* $P \equiv Q \Rightarrow \langle P, M, E \rangle \equiv \langle Q, M, E \rangle$. Assume $C = \langle P, M, E \rangle$ and $C' = \langle Q, M, E \rangle$. We proceed by case analysis on the hypothesis $P \equiv Q$. There are 4 rules by which it may be deduced:

Case $R \mid \mathbf{0} \equiv R$**.** Suppose $P = R \mid \mathbf{0}$ and $Q = R$. By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$. This judgement is necessarily deduced using Rule ⌊CONC⌋. By inversion, (i) $\Gamma \vdash \langle R, M, E \rangle \triangleright \langle \Delta_1 \diamond \Theta \rangle$, and (ii) $\Gamma \vdash \langle \mathbf{0}, M, E \rangle \triangleright \langle \Delta_2 \diamond \Theta \rangle$. Furthermore, the judgement in (ii) has to be necessarily deduced using Rule ⌊INACT⌋. Thus, applying inversion, (iv) $\Delta_2 = \Delta^{end}$. Thus, we conclude by applying Rule ⌊WEAK⌋ to add Δ_2 in (i), obtaining $\Gamma \vdash \langle R, M, E \rangle \triangleright \langle \Delta_1, \Delta_2 \diamond \Theta \rangle$.
Let us now consider the inverse case, i.e. $P = R$ and $Q = R \mid \mathbf{0}$. By assumption, we have $\Gamma \vdash \langle R, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$, and we have to prove that $\Gamma \vdash \langle R \mid \mathbf{0}, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$. By Lem. 7, (i) $\Gamma \vdash M \triangleright \Theta$. Then, observe that $\Delta_0 = \emptyset$ implies that $\Delta_0 = \Delta^{end}$ by Def. 11. We may then apply Rule ⌊INACT⌋, using $\Delta_0 = \emptyset$ and (i) to derive $\Gamma \vdash \langle \mathbf{0}, M, E \rangle \triangleright \langle \emptyset \diamond \Theta \rangle$, and subsequently apply Rule ⌊CONC⌋ to obtain $\Gamma \vdash \langle R \mid 0, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$.

Case $\texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\} \equiv \mathbf{0}$**.** Suppose $P = \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\}$ and $Q = \mathbf{0}$. By assumption, $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$. This judgement is deduced with Rule ⌊WATCH⌋, hence $\Delta = \{s[\mathtt{p}] : T_{\mathbf{0}} \star_{ev} T_R\}$. By inversion on Rule ⌊WATCH⌋, we have (i) $\Gamma \vdash M \triangleright \Theta$ and (ii) $\Gamma \vdash \langle \mathbf{0}, M, E \rangle \triangleright \langle s[\mathtt{p}] : T_{\mathbf{0}} \diamond \Theta \rangle$. Since (ii) is deduced by Rule ⌊INACT⌋, it must be $T_{\mathbf{0}} = \mathsf{end}$. Hence also $T_{\mathbf{0}} \star_{ev} T_R = \mathsf{end}$ (cf. Def. 14), and we may conclude that $\Gamma \vdash \langle Q, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$.
Conversely, suppose $P = \mathbf{0}$ and $Q = \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\}$. By assumption, $\Gamma \vdash \langle \mathbf{0}, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$. This judgement is deduced by Rule ⌊INACT⌋. By inversion we have (i) $\Delta = \Delta^{end}$. Hence, we distinguish two cases: (1) $\Delta = \emptyset$ and (2) $\Delta \neq \emptyset$. In Case (1), we apply Rule ⌊WEAK⌋ to obtain $\Gamma \vdash \langle \mathbf{0}, M, E \rangle \triangleright \langle c : \mathsf{end} \diamond \Theta \rangle$. Then, we apply Rule ⌊WATCH⌋ to deduce $\Gamma \vdash \langle \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\}, M, E \rangle \triangleright \langle s[\mathtt{p}] : \mathsf{end} \diamond \Theta \rangle$. Finally, we apply Rule ⌊CONTR⌋ to conclude $\Gamma \vdash \langle \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\}, M, E \rangle \triangleright \langle \emptyset \diamond \Theta \rangle$.
For Case (2), consider an arbitrary $s[\mathtt{p}] \in dom(\Delta)$. By (i), $s[\mathtt{p}] : \mathsf{end} \in \Delta$. Then, using the contraction rule ⌊CONTR⌋, we may deduce (ii) $\Gamma \vdash \langle \mathbf{0}, M, E \rangle \triangleright \langle s[\mathtt{p}] : \mathsf{end} \diamond \Theta \rangle$. We may now apply Rule ⌊WATCH⌋ to the judgement (ii) to obtain $\Gamma \vdash \langle \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\}, M, E \rangle \triangleright \langle s[\mathtt{p}] : \mathsf{end} \diamond \Theta \rangle$. Then, by iteratively applying Rule ⌊WEAK⌋ to add $\Delta \setminus s[\mathtt{p}] : \mathsf{end}$ to the session environment, we finally obtain $\Gamma \vdash \langle \texttt{watch}\, ev\, \texttt{do}\, \mathbf{0}\{R\}, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$.

Case $R_1 \mid R_2 \equiv R_2 \mid R_1$**.** The proof of this case is straightforward by inversion, since the hypotheses $\Gamma \vdash \langle R_1, M, E \rangle \triangleright \langle \Delta_1 \diamond \Theta \rangle$ and $\Gamma \vdash \langle R_2, M, E \rangle \triangleright \langle \Delta_2 \diamond \Theta \rangle$ are not ordered in Rule ⌊CONC⌋.

Case $(R_1 \mid R_2) \mid R_3 \equiv R_1 \mid (R_2 \mid R_3)$**.** Again, the proof is straightforward by inversion, considering that $(dom(\Delta_1) \cap dom(\Delta_2)) \cap dom(\Delta_3) = \emptyset$ if and only if $dom(\Delta_1) \cap (dom(\Delta_2) \cap dom(\Delta_3)) = \emptyset$.

Case 2. *Rule* $C_1 \equiv C_2 \Rightarrow (\nu s)C_1 \equiv (\nu s)C_2$. Suppose that $C = (\nu s)C_1$ and $C' = (\nu s)C_2$. By assumption, $\Gamma \vdash C \triangleright \langle \emptyset \diamond \emptyset \rangle$. This judgement is derived with Rule $\lfloor$CRES$\rfloor$. Then, by inversion, $\Gamma \vdash C_1 \triangleright \langle \Delta \diamond \Theta \rangle$ and $Co\langle \Delta \diamond \Theta \rangle$. By induction $\Gamma \vdash C_2 \triangleright \langle \Delta_1 \diamond \Theta_1 \rangle$, and we can apply Rule $\lfloor$CRES$\rfloor$ to this statement and to $Co\langle \Delta_1 \diamond \Theta_1 \rangle$ to obtain $\Gamma \vdash (\nu s)C_2 \triangleright \langle \Delta_1 \diamond \Theta_1 \rangle$, namely $\Gamma \vdash C' \triangleright \langle \Delta \diamond \Theta \rangle$.

□

Lemma 9 (Substitution lemma)**.** *If* $\Gamma, x : S \vdash C \triangleright \langle \Delta \diamond \Theta \rangle$ *and* $\Gamma \vdash v : S$ *then* $\Gamma \vdash C\{v/x\} \triangleright \langle \Delta \diamond \Theta \rangle$.

Proof. By induction on the height of the type derivation. The base cases are rules $\lfloor$INACT$\rfloor$ and $\lfloor$MINIT$\rfloor$. The thesis is easily derived observing that x does not occur free neither in **0** nor in the initiator.

For the inductive cases, the interesting ones are those of rules $\lfloor$SENDFIRST$\rfloor$ and $\lfloor$SENDMORE$\rfloor$. All other cases follow by the fact that the process under consideration either do not contain free occurrences of x or the conclusion follows by a direct application of the inductive hypothesis.

Let us consider the case of rule $\lfloor$SENDFIRST$\rfloor$ (the treatment for rule $\lfloor$SENDMORE$\rfloor$ will be the same). We know that:

$$\Gamma, x : S \vdash \langle s[\mathtt{p}]!\langle e \rangle.P, M \cup s[\mathtt{p}] : \epsilon, E \rangle \triangleright \langle s[\mathtt{p}] :!S.T \diamond \Theta, s[\mathtt{p}] : \mathtt{void} \rangle$$

By inversion we have: (1) $\Gamma, x : S \vdash e : S$ and (2) $\Gamma, x : S \vdash \langle P, M \cup s[\mathtt{p}] : (d_S, \emptyset), E \rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta, s[\mathtt{p}] : (S, \emptyset) \rangle$.

By an easy induction on the height of the type derivation of expressions, Fig. 14, and from (1) we deduce $\Gamma, v : S \vdash e\{v/x\} : S$. From (2), by inductive hypothesis we conclude $\Gamma, v : S \vdash \langle P\{v/x\}, M \cup s[\mathtt{p}] : (d_S, \emptyset), E \rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta, s[\mathtt{p}] : (S, \emptyset) \rangle$.

Finally using rule $\lfloor$SENDFIRST$\rfloor$ we conclude this case obtaining:

$$\Gamma, v : S \vdash \langle (s[\mathtt{p}]!\langle e \rangle.P)\{v/x\}, M \cup s[\mathtt{p}] : \epsilon, E \rangle \triangleright \langle s[\mathtt{p}] :!S.T \diamond \Theta, s[\mathtt{p}] : \mathtt{void} \rangle$$

□

Lemma 10 (Reduction lemma)**.** *Let* $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$ *and* $\langle P, M, E \rangle \longrightarrow \langle P', M', E' \rangle$ *via some reduction rule different from* [Cont] *and* [Struct]. *Then* $\langle \Delta \diamond \Theta \rangle \Rightarrow \langle \Delta' \diamond \Theta' \rangle$ *and* $\Gamma \vdash \langle P', M', E' \rangle \triangleright \langle \Delta' \diamond \Theta' \rangle$. *Moreover, if* $\langle \Delta, \Delta_0 \diamond \Theta, \Theta_0 \rangle$ *is coherent, then* $\langle \Delta', \Delta_0 \diamond \Theta', \Theta_0 \rangle$ *is coherent.*

Proof. By induction on length of the inference of the reduction $\langle P, M, E \rangle \longrightarrow \langle P', M', E' \rangle$, with a case analysis on the last applied rule. Notice that Case [Init] is not considered here, as we are only interested in reductions that affect the processes inside a reachable non initial configuration.

Case 1. [Out]*:*

Let $P = s[\mathtt{p}]!\langle e \rangle.P'$, and $M = M'' \cup \{s[\mathtt{p}] : \varepsilon\}$ for some M''. P is typed with rule $\lfloor$SENDFIRST$\rfloor$ and by hypothesis we have: $e \downarrow v$, $M' = M'' \cup \{s[\mathtt{p}] : (v, \emptyset)\}$, $\Delta = s[\mathtt{p}] :!S.T$ and $\Theta = \Theta'', s[\mathtt{p}] : \mathtt{void}$.

By inversion on rule $\lfloor$SENDFIRST$\rfloor$ we know that $\Gamma \vdash e : S$ and $\Gamma \vdash \langle P', M'' \cup \{s[\mathtt{p}] : (d_S, \emptyset)\}, E \rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta'', s[\mathtt{p}] : (S, \emptyset) \rangle$.

Now since $\Gamma \vdash e : S$ and $e \downarrow v$ we know that $\Gamma \vdash v : S$. Hence we can conclude that

$$\Gamma \vdash \langle P', M'' \cup \{s[\mathtt{p}] : (v, \emptyset)\}, E \rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta'', s[\mathtt{p}] : (S, \emptyset) \rangle$$

and by using item 1 of Def. (Def. 19) we have that

$$\langle s[\mathtt{p}] :!S.T \diamond \Theta'', s[\mathtt{p}] : \mathtt{void} \rangle \Rightarrow \langle s[\mathtt{p}] : T \diamond \Theta'', s[\mathtt{p}] : (S, \emptyset) \rangle$$

Finally by assumption we have, $Co\,\langle s[\mathtt{p}] :! S.T, \Delta_0 \diamond \Theta'', s[\mathtt{p}] : \mathtt{void}, \Theta_0\rangle$. We know that $Co\,\langle s[\mathtt{p}] : T, \Delta_0 \diamond \Theta'', \{s[\mathtt{p}] : (S, \emptyset), \Theta_0\}\rangle$. We only have to consider the impact of $s[\mathtt{p}]$as the other members of the environment are unchanged. Since $s[\mathtt{p}] \in vdom(\Theta'', s[\mathtt{p}] : (S, \emptyset), \Theta_0)$ Condition 1 of Def. 18 holds. For Condition 2, duality is preserved, as the generalized type of $s[\mathtt{p}]$ in Δ will lift the necessary output from Θ to re-construct the correct type.

Case 2. [In]*:*

Let $P = s[\mathtt{q}]?(\mathtt{p}, x).P''$ for some P'' and $M = M'' \cup \{s[\mathtt{p}] : (v, \Pi)\}$ for some M'', v, Π such that $\mathtt{q} \notin \Pi$.

P is typed with rule $\lfloor$RcvFirst$\rfloor$. By hypothesis we have: $P' = P''\{v/x\}$, $M' = M'' \cup \{s[\mathtt{p}] : (v, \Pi \cup \mathtt{q})\}$, $\Delta = s[\mathtt{q}] :?(\mathtt{p}, S).T$ and $\Theta = \Theta'', s[\mathtt{p}] : (S, \Pi)$.

By inversion on $\lfloor$RcvFirst$\rfloor$ we know that

$$\Gamma, x : S \vdash \langle P'', M'' \cup s[\mathtt{p}] : (v, \Pi \cup \{\mathtt{q}\}), E\rangle \triangleright \langle s[\mathtt{q}] : T \diamond \Theta'', \{s[\mathtt{p}] : (S, \Pi \cup \{\mathtt{q}\})\}\rangle$$

Now by Lemma Lem. 9 we can conclude that

$$\Gamma \vdash \langle P''\{v/x\}, M'' \cup s[\mathtt{p}] : (v, \Pi \cup \{\mathtt{q}\}), E\rangle \triangleright \langle s[\mathtt{q}] : T \diamond \Theta'', \{s[\mathtt{p}] : (S, \Pi \cup \{\mathtt{q}\})\}\rangle$$

and by item 2 in Def. 19

$$\langle s[\mathtt{q}] :?(\mathtt{p}, S).T \diamond \Theta'', s[\mathtt{p}] : (S, \Pi)\rangle \;\Rightarrow\; \langle s[\mathtt{q}] : T \diamond \Theta'', s[\mathtt{p}] : (S, \Pi \cup \{\mathtt{q}\})\rangle$$

Finally, we prove $Co\,\langle s[\mathtt{q}] : T, \Delta_0 \diamond \Theta'', s[\mathtt{p}] : (S, \Pi), \Theta_0\rangle$. Let us check the first condition for coherence; notice that $Co\,\langle \Delta, \Delta_0 \diamond \Theta, \Theta_0\rangle$ implies that either $s[\mathtt{q}] \in vdom(\Theta, \Theta_0)$ or $\mathsf{OG}(?(\mathtt{p}, S).T)$. In both cases coherence holds by assumption. Now, to check duality, let $\Delta_0 = \Delta_0', s[\mathtt{p}] : T'$ and we know that $s[\mathtt{p}] : (S, \Pi) \in \Theta$. Hence, by assumption $\langle \Delta, \Delta_0 \diamond \Theta, \Theta_0\rangle(s[\mathtt{q}]) = ?(\mathtt{p}, S).T$ and $\langle \Delta, \Delta_0 \diamond \Theta, \Theta_0\rangle(s[\mathtt{p}]) = !S.T'$. Furthermore, $?(\mathtt{p}, S).T \bowtie !S.T'$ implies $T \bowtie T'$ by Def. 17 and therefore, $T \bowtie !s.T'$, which in turn, ensures that duality is preserved in $\langle \Delta', \Delta_0 \diamond \Theta', \Theta_0\rangle$.

Case 3. [Emit]*:*

Immediate, since this rule does not change the memory nor the configuration environment.

Case 4. [Watch]*:*

Let $P = \mathtt{watch}\, ev\, \mathtt{do}\, P'\{Q\}$. By assumption we know $\langle P, M, E\rangle \longrightarrow \langle \mathtt{watch}\, ev\, \mathtt{do}\, P'\{Q\}, M', E'\rangle$ and $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle T'_P \star_{ev} T_Q \diamond \Theta\rangle$. By inversion on rule $\lfloor$Watch$\rfloor$ we have $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle T_{P'} \diamond \Theta\rangle$, $\Gamma \vdash \langle Q, M^{\emptyset}, E\rangle \triangleright \langle T_Q \diamond \Theta^{\emptyset}\rangle$ and $\mathsf{OG}(T_Q)$. By inductive hypothesis we have $\Gamma \vdash \langle P', M', E'\rangle \triangleright \langle T_P \diamond \Theta'\rangle$ and $\langle T_{P'} \diamond \Theta\rangle \;\Rightarrow\; \langle T'_{P'} \diamond \Theta'\rangle$. Hence, by item 3 in Def. 19 we conclude
$\langle T_{P'} \star_{ev} T_Q \diamond \Theta\rangle \;\Rightarrow\; \langle T'_{P'} \star_{ev} T_Q \diamond \Theta'\rangle$.

Finally, assume $Co\,\langle T_{P'} \star_{ev} T_Q, \Delta_0 \diamond \Theta, \Theta_0\rangle$. This implies $Co\,\langle T_{P'}, \Delta_0 \diamond \Theta, \Theta_0\rangle$ and hence, by inductive hypothesis we have $Co\,\langle T'_{P'}, \Delta_0 \diamond \Theta', \Theta_0\rangle$, which in turn implies $Co\,\langle T'_{P'} \star_{ev} T_Q, \Delta_0 \diamond \Theta', \Theta_0\rangle$.

Case 5. [Rec]*:*

The proof proceeds as above by using the inductive hypothesis.

□

To prove subject reduction, we first need to prove that tick reduction for configuration environments preserves duality and that suspension preserves typing. This is useful for ensuring that the semantics given to environment configurations works correctly.

Lemma 11 (Tick reduction of configuration environments preserves duality)**.** *If* $\langle \Delta \diamond \Theta \rangle \curvearrowright_E \langle \Delta' \diamond \Theta' \rangle$ *then* $\Delta' = [\Delta]_E$*,* $dom(\Delta') \subseteq dom(\Delta)$ *and* $\Theta' = \Theta^{\emptyset}$*. Moreover, if* $\langle \Delta \diamond \Theta \rangle$ *satisfies duality then also* $\langle \Delta' \diamond \Theta' \rangle$ *satisfies duality and for any* $s[\mathtt{p}], s[\mathtt{q}] \in dom(\Delta')$*, if* $s[\mathtt{p}] : T_{\mathtt{p}} \in \Delta$ *and* $s[\mathtt{q}] : T_{\mathtt{q}} \in \Delta$*, then* $\langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q} \bowtie \langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) \upharpoonright \mathtt{p}$ *if and only if* $[T_{\mathtt{p}}]_E \upharpoonright \mathtt{q} \bowtie [T_{\mathtt{q}}]_E \upharpoonright \mathtt{p}$.

Proof. By induction on the definition of $\curvearrowright$. There is only one basic case, corresponding to Rule (Pause).

Case 1. *(Pause):*

In this case $\langle \Delta \diamond \Theta \rangle \curvearrowright_E \langle \Delta' \diamond \Theta' \rangle$ is deduced by Rule 1. of Def. 20 and we have $\Delta = \mathsf{pause}(\Delta')$, $E = \emptyset$, $\Delta' = [\Delta]_E$ and $\Theta' = \Theta^{\emptyset}$. Moreover $dom(\Delta') = dom(\Delta)$.

Assume now $s[\mathtt{p}] : T_{\mathtt{p}} \in \Delta$ and $s[\mathtt{q}] : T_{\mathtt{q}} \in \Delta$. Then, letting $T'_{\mathtt{p}} = [T_{\mathtt{p}}]_E$ and $T'_{\mathtt{q}} = [T_{\mathtt{q}}]_E$, we have:

$$T_{\mathtt{p}} = \mathtt{pause}.T'_{\mathtt{p}} \qquad T_{\mathtt{q}} = \mathtt{pause}.T'_{\mathtt{q}} \qquad s[\mathtt{p}] : T'_{\mathtt{p}} \in \Delta' \qquad s[\mathtt{q}] : T'_{\mathtt{q}} \in \Delta'$$

We want to prove the duality of $\langle \Delta' \diamond \Theta^{\emptyset} \rangle$ assuming the duality of $\langle \Delta \diamond \Theta \rangle$. Note that $\langle \Delta' \diamond \Theta^{\emptyset} \rangle(s[\mathtt{p}]) = T'_{\mathtt{p}}$ and $\langle \Delta' \diamond \Theta^{\emptyset} \rangle(s[\mathtt{q}]) = T'_{\mathtt{q}}$. On the other hand, there are two possibilities for $\langle \Delta \diamond \Theta \rangle(s[\mathtt{p}])$: either $\langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) = T_{\mathtt{p}}$, in case $s[\mathtt{p}] \notin vdom(\Theta)$, or $\langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) = !S_{\mathtt{p}}.T_{\mathtt{p}}$, in case $s[\mathtt{p}] : (S_{\mathtt{p}}, \Pi_{\mathtt{p}}) \in \Theta$. Similarly, either $\langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) = T_{\mathtt{q}}$ or $\langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) = !S_{\mathtt{q}}.T_{\mathtt{q}}$.

Let now $\tau_{\mathtt{p}} = T_p \upharpoonright \mathtt{q}$, $\tau_{\mathtt{q}} = T_q \upharpoonright \mathtt{p}$, and $\tau'_{\mathtt{p}} = T'_p \upharpoonright \mathtt{q}$, $\tau'_{\mathtt{q}} = T'_q \upharpoonright \mathtt{p}$. Since projection preserves $\mathtt{pause}$, we have $\tau_{\mathtt{p}} = \mathtt{pause}.\tau'_{\mathtt{p}}$ and $\tau_{\mathtt{q}} = \mathtt{pause}.\tau'_{\mathtt{q}}$. Then:

$$\langle \Delta' \diamond \Theta^{\emptyset} \rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q} = T'_p \upharpoonright \mathtt{q} = \tau'_{\mathtt{p}} \qquad \langle \Delta' \diamond \Theta^{\emptyset} \rangle(s[\mathtt{q}]) \upharpoonright \mathtt{p} = T'_q \upharpoonright \mathtt{p} = \tau'_{\mathtt{q}}$$

Moreover, we have:

$$\begin{array}{lll} \text{either} & \langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q} = \tau_{\mathtt{p}} & \text{or} \quad \langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q} = !S_{\mathtt{p}}.\tau_{\mathtt{p}} \\ \text{either} & \langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) \upharpoonright \mathtt{p} = \tau_{\mathtt{q}} & \text{or} \quad \langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) \upharpoonright \mathtt{p} = !S_{\mathtt{q}}.\tau_{\mathtt{q}} \end{array}$$

We want to show $\tau'_{\mathtt{p}} \bowtie \tau'_{\mathtt{q}}$. By assumption duality holds for $\langle \Delta \diamond \Theta \rangle$, so we are in one of the four cases:

$$\begin{array}{rcll} \tau_{\mathtt{p}} & \bowtie & \tau_{\mathtt{q}} & \text{i.e.} \quad \mathtt{pause}.\tau'_{\mathtt{p}} \bowtie \mathtt{pause}.\tau'_{\mathtt{q}} \\ !S_{\mathtt{p}}.\tau_{\mathtt{p}} & \bowtie & \tau_{\mathtt{q}} & \text{i.e.} \quad !S_{\mathtt{p}}.\mathtt{pause}.\tau'_{\mathtt{p}} \bowtie \mathtt{pause}.\tau'_{\mathtt{q}} \\ \tau_{\mathtt{p}} & \bowtie & !S_{\mathtt{q}}.\tau_{\mathtt{q}} & \text{i.e.} \quad \mathtt{pause}.\tau'_{\mathtt{p}} \bowtie !S_{\mathtt{q}}.\mathtt{pause}.\tau'_{\mathtt{q}} \\ !S_{\mathtt{p}}.\tau_{\mathtt{p}} & \bowtie & !S_{\mathtt{q}}.\tau_{\mathtt{q}} & \text{i.e.} \quad !S_{\mathtt{p}}.\mathtt{pause}.\tau'_{\mathtt{p}} \bowtie !S_{\mathtt{q}}.\mathtt{pause}.\tau'_{\mathtt{q}} \end{array}$$

Now, it is easy to see that any of the four statements on the right-hand side implies $\tau'_{\mathtt{p}} \bowtie \tau'_{\mathtt{q}}$. Hence we have shown duality of $\langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ and more specifically that $\langle \Delta \diamond \Theta \rangle(s[\mathtt{p}]) \upharpoonright \mathtt{q} \bowtie \langle \Delta \diamond \Theta \rangle(s[\mathtt{q}]) \upharpoonright \mathtt{p}$ if and only if $[T_{\mathtt{p}}]_E \upharpoonright \mathtt{q} \bowtie [T_{\mathtt{q}}]_E \upharpoonright \mathtt{p}$.

Case 2. *(Tick):*

Analogous to the previous case.

Case 3. *(In):*

This case is vacuously true, since $\Delta = s[\mathtt{q}]\ :?(\mathtt{p}, S).T$ and hence it does not have any other type to be compared to.

Case 4. *(Par):*

In this case $\langle \Delta \diamond \Theta \rangle \curvearrowright_E \langle \Delta' \diamond \Theta' \rangle$ is deduced by Rule 4. of Definition 20. Therefore $\Delta = (\Delta_1, \Delta_2)$ and $\langle \Delta_i \diamond \Theta \rangle \curvearrowright_{E_i} \langle \Delta_i' \diamond \Theta' \rangle$, for $i = 1, 2$. By induction $\Delta_i' = [\Delta_i]_E$, $dom(\Delta_i') \subseteq dom(\Delta_i)$ and $\Theta' = \Theta^{\emptyset}$. Now, suppose $\langle \Delta_1, \Delta_2 \diamond \Theta \rangle$ satisfies duality. We want to show that also $\langle [\Delta_1, \Delta_2]_E \diamond \Theta^{\emptyset} \rangle = \langle [\Delta_1]_{E_1}, [\Delta_2]_E \diamond \Theta^{\emptyset} \rangle$ satisfies duality. Note that since the Δ_i have disjoint domains, the duality of $\langle \Delta_1, \Delta_2 \diamond \Theta \rangle$ entails the duality of each $\langle \Delta_i \diamond \Theta \rangle$. Then by induction also each $\langle [\Delta_i]_E \diamond \Theta^{\emptyset} \rangle$ satisfies duality. Hence, to check duality of $\langle [\Delta_1]_E, [\Delta_2]_E \diamond \Theta^{\emptyset} \rangle$ we only have to look at pairs $s[\mathtt{p}_1], s[\mathtt{p}_2]$ such that $s[\mathtt{p}_1] \in dom([\Delta_1]_E)$ and $s[\mathtt{p}_2] \in dom([\Delta_2]_E)$, because the other cases (i.e., when pairs belong to only one single Δ_i) conclude by IH. Let $s[\mathtt{p}_1], s[\mathtt{p}_2]$ be such a pair. Recalling that $dom([\Delta_i]_E) \subseteq dom(\Delta_i)$, this means that for each $i = 1, 2$, there exists T_i such that $s[\mathtt{p}_i] : T_i \in \Delta_i$. By definition this implies $s[\mathtt{p}_i] : [T_i]_E \in [\Delta_i]_E$. Let $T_i' = [T_i]_{E_i}$. Thus we have:

$$\langle [\Delta_1, \Delta_2]_E \diamond \Theta^{\emptyset} \rangle (s[\mathtt{p}_1]) = T_1' \qquad \langle [\Delta_1, \Delta_2]_E \diamond \Theta^{\emptyset} \rangle (s[\mathtt{p}_2]) = T_2'$$

As regards $\langle \Delta_1, \Delta_2 \diamond \Theta \rangle (s[\mathtt{p}_i])$, we have:

$$\begin{array}{ll} \text{either} \quad \langle \Delta_1, \Delta_2 \diamond \Theta \rangle (s[\mathtt{p}_i]) = T_i & \text{if} \quad s[\mathtt{p}_i] \notin vdom(\Theta) \\ \text{or} \quad \langle \Delta_1, \Delta_2 \diamond \Theta \rangle (s[\mathtt{p}_i]) = !S_i.T_i & \text{if} \quad s[\mathtt{p}_i] : (S_i, \Pi_i) \in \Theta \end{array}$$

Let now $\tau_1 = T_1 \upharpoonright \mathtt{p}_2$ and $\tau_2 = T_2 \upharpoonright \mathtt{p}_1$ and $\tau_1' = T_1' \upharpoonright \mathtt{p}_2$ and $\tau_2' = T_2' \upharpoonright \mathtt{p}_1$. We need to prove that $\tau_1 \bowtie \tau_2$ implies $\tau_1' \bowtie \tau_2'$. There are then 4 cases to analyze:

$$\begin{array}{rcl} \tau_1 & \bowtie & \tau_2 \\ !S_1.\tau_1 & \bowtie & \tau_2 \\ \tau_1 & \bowtie & !S_2.\tau_2 \\ !S_1.\tau_1 & \bowtie & !S_2.\tau_2 \end{array}$$

this proceeds as follows:

Case $\tau_1 \bowtie \tau_2$**:** We need to apply induction on the structure of τ_1, base case is $\tau_1 = \mathtt{end}$:

- **Case** $\tau_1 = \mathtt{end} \vee \tau_1 = \mathbf{t} \vee \tau_1 = !S.\tau_1'' \vee \tau_1'' = ?S.\tau_1''$**:** In this case, for $\tau_2 = \mu\mathbf{t}.\tau_2''$ and $\tau_2 = \mathtt{pause}.\tau_2''$ the statement is vacuously true, since duality dos not hold, and for the other case, duality proceeds simply by assumption, since the types do not change.
- **Case** $\tau_1 = \mathtt{pause}.\tau_1''$**:** Now, we apply a case analysis on τ_2:
  - **Case** $\tau_2 = \mathtt{pause}.\tau_2''$**:** In this case we have that $\mathtt{pause}.\tau_1'' \bowtie \mathtt{pause}.\tau_2''$, and then $[\mathtt{pause}.\tau_1'']_E = \tau_1''$, as well as $[\mathtt{pause}.\tau_2'']_E = \tau_2''$, and since we know that $T_i' = [T_i]_E$, then we know that $\tau_1'' = \tau_1'$ and $\tau_2'' = \tau_2'$, and hence it is clear that $\tau_1' \bowtie \tau_2'$ by Def. 17. All the other cases are vacuously true.
- **Case** $\tau_1 = \mu\mathbf{t}.\tau_1''$ **:** Again, we can only proceed for the case when $\tau_1 = \mu\mathbf{t}'.\tau_2''$, other cases are vacuously true. In this case, we proceed by applying the IH, since we are looking at the bodies of the recursive types.

Case $\tau_1 = \langle \varphi_1, \psi_1 \rangle^e$**:** For this case we only need to look at $\tau_2 = \langle \varphi_2, \psi_2 \rangle^e$, all the other cases are vacuously true. For this case we have to consider two subcases depending on whether $e \in E$ or $e \notin E$:

Subcase $e \in E$**:** In this case, by assumption, we have that $\langle \varphi_1, \psi_1 \rangle^e \bowtie \langle \varphi_2, \psi_2 \rangle^e$. Also, by Def. 21 we have that $[\langle \varphi_1, \psi_1 \rangle^e]_E = \psi_1$ and $[\langle \varphi_2, \psi_2 \rangle^e]_E = \psi_2$, hence $\tau_1' = \psi_1$ and $\tau_2 = \psi_2$. Notice that by Def. 17, $\langle \varphi_1, \psi_1 \rangle^e \bowtie \langle \varphi_2, \psi_2 \rangle^e$ implies $\varphi_1 \bowtie \varphi_2$ and $\psi_1 \bowtie \psi_2$ and therefore, we conclude by assumption.

Subcase $e \notin E$**:** This case concludes by applying the IH hypothesis, since the reconditioning is applied to the first part of the watch type.

Case $!S.\tau_1 \bowtie \tau_2$**:** As above, we apply induction on the structure of τ_1 and obtain the same cases. This is possible if we consider the fact that the duality of $!S.\tau_1 \bowtie \tau_2$ implies the duality of $\tau_1 \bowtie \tau_2$.

Case $\tau_1 \bowtie !S.\tau_2$**:** As above.

Case $!S\tau_1 \bowtie !S'.\tau_2$**:** As above, notice that the reasoning is: By Def. 17, $!S\tau_1 \bowtie !S.\tau_2$ implies $\tau_1 \bowtie !S.\tau_2$, which in turn implies $!S\tau_1 \bowtie \tau_2$ and from here on, the reasoning is as the above mentioned cases.

Case 5. *(Watch):*

In this case $\langle \Delta \diamond \Theta \rangle \curvearrowright \langle \Delta' \diamond \Theta' \rangle$ is deduced by Rule 6. of Definition 20. Then $\Delta = \Delta_1 \star_{ev} \Delta_2$. There are two cases two consider, depending on the result of the composition operator in Def. 14:

Case $\Delta_1 = \Delta^{end}$**:** This case is vacuously true since there is no reduction for a terminated environment.

Case $\Delta_1 \neq \Delta^{end}$**:** In this case, again we have two cases to consider, depending on whether $ev \in E$ or not:

Case $ev \in E$**:** In this case, we have by Def. 21 that $[\Delta_1 \star_e \Delta_2]_E = \Delta_2$, since by assumption $e \in E$, also, by assumption we have that $\Delta' = \Delta_2$, hence $\Delta' = [\Delta_1 \star_e \Delta_2]_E$, and applying the IH we have that $\langle \Delta \diamond \Theta \rangle \curvearrowright \langle \Delta' \diamond \Theta^{\emptyset} \rangle$, thus proving the first part of the Lemma. Notice also that by Def. 14 we have that $dom(\Delta_1) = dom(\Delta_2)$ or the function would be undefined and hence $dom(\Delta') \subseteq dom(\Delta)$. Now, let us assume that $\langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle$ satisfies duality, we need to prove that $\langle [\Delta_1 \star_{ev} \Delta_2]_E \diamond \Theta^{\emptyset} \rangle$ also satisfies duality. For this we consider two arbitrary $s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{q}] : T_{\mathtt{q}} \in \Delta$. This means then that, as above we have two possibilities for each participant-type pair:

$$\langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle(s[\mathtt{p}]) = T_{\mathtt{p}} \vee \langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle(s[\mathtt{p}]) = !S.T_{\mathtt{p}}$$
$$\langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle(s[\mathtt{q}]) = T_{\mathtt{p}} \vee \langle \Delta_1 \star_{ev} \Delta_2 \diamond \Theta \rangle(s[\mathtt{q}]) = !S'.T_{\mathtt{q}}$$

Then, let $T_{\mathtt{p}} \upharpoonright \mathtt{q} = \tau_{\mathtt{p}}$ and $T_{\mathtt{q}} \upharpoonright \mathtt{p} = \tau_{\mathtt{q}}$, then, as above, we have the following cases:

$$\begin{array}{rcl} \tau_{\mathtt{p}} & \bowtie & \tau_{\mathtt{q}} \\ !S_{\mathtt{p}}.\tau_{\mathtt{p}} & \bowtie & \tau_{q} \\ \tau_{\mathtt{p}} & \bowtie & !S_{\mathtt{q}}.\tau_{\mathtt{q}} \\ !S_{\mathtt{p}}.\tau_{\mathtt{p}} & \bowtie & !S'_{\mathtt{q}}.\tau_{\mathtt{q}} \end{array}$$

Lastly, the proof proceeds as above by cases. Notice, however, that by Def. 14, one has that for all $c : T \in \Delta$ the type $T = \langle T_1, T_2 \rangle^{ev}$. Furthermore, since duality for the watch

type is defined pairwise (cf. Def. 17) and considering the fact that the reconditioning in the presence of ev does not change the environment Δ_2, we can conclude that $\langle \Delta_2 \diamond \Theta \rangle$ preserves duality, concluding this case.

Case $ev \notin E$**:** This case proceeds with a reasoning similar to the Case $\tau_1 = \mu \mathbf{t}.\tau_1''$ above: by IH and case analysis. We need to consider the full environment, and check only Δ_1, since Δ_2 remains untouched.

□

Lemma 12 (Suspension lemma)**.** *Let* $\langle P, M, E \rangle \ddagger$ *and* $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$*. Then* $\langle \Delta \diamond \Theta \rangle \curvearrowright \langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ *and* $\Gamma \vdash \langle [P]_E, M^{\emptyset}, \emptyset \rangle \triangleright \langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$*. Moreover, if* $\langle \Delta \diamond \Theta \rangle$ *is coherent then* $\langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ *is coherent.*

Proof. By induction on the definition of $\langle P, M, E \rangle \ddagger$. The basic cases correspond to the suspension rules (*pause*), (out_s), (in_s) and (in_s^2), and the inductive cases to rules (par_s), ($watch_s$) and (rec_s).

We start with the basic cases.

Case 1. *(pause):*

Assume $P = \texttt{pause}.P'$. Hence $[P]_E = P'$. By assumption $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$. This judgement is derived by Rule ⌊PAUSE⌋, hence $\Delta = \texttt{pause}.T$ and $[\Delta]_E = T$. By inversion on Rule ⌊PAUSE⌋ we have $\Gamma \vdash M \triangleright \Theta$, $\mathsf{OG}(T)$, and $\Gamma \vdash \langle P', M^{\emptyset}, \emptyset \rangle \triangleright \langle T \diamond \Theta^{\emptyset} \rangle$. The latter is the required judgement $\Gamma \vdash \langle [P]_E, M^{\emptyset}, \emptyset \rangle \triangleright \langle [\texttt{pause}.T]_E \diamond \Theta^{\emptyset} \rangle$. Moreover, $\langle \texttt{pause}.T \diamond \Theta \rangle \curvearrowright \langle [\texttt{pause}.T]_E \diamond \Theta^{\emptyset} \rangle = \langle T \diamond \Theta^{\emptyset} \rangle$ by Clause 1. of Def. 20. Assume now that $\langle \Delta \diamond \Theta \rangle$ is coherent. Then $\langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ is coherent, since Condition 1. of Def. 18 follows from $\mathsf{OG}(T)$ and Condition 2. of Def. 18 follows from Lemma 11.

Case 2. *(out_s):*

Assume $P = s[\mathtt{p}]!\langle e \rangle.P'$ and $M = M_0 \cup \{s[\mathtt{p}] : (w, \Pi)\}$. By assumption $\Gamma \vdash \langle P, M, E \rangle \triangleright \langle \Delta \diamond \Theta \rangle$. This judgement is deduced by Rule ⌊SENDMORE⌋, therefore, assuming $e \downarrow v$, $\Gamma \vdash v : S$ and $\Gamma \vdash w : S'$, it has the form:

$$\Gamma \vdash \langle s[\mathtt{p}]!\langle e \rangle.P', M_0 \cup \{s[\mathtt{p}] : (w, \Pi)\}, E \rangle \triangleright \langle s[\mathtt{p}] : \texttt{tick}.!S.T \diamond \Theta_0, s[\mathtt{p}] : (!S', \Pi) \rangle$$

where $\Delta = s[\mathtt{p}] : \texttt{tick}.!S.T$ and $\Theta = \Theta_0, s[\mathtt{p}] : (!S', \Pi)$. By reconditioning P and Δ we obtain $[P]_E = P = s[\mathtt{p}]!\langle e \rangle.P'$ and $[\Delta]_E = s[\mathtt{p}] :!S.T$. By inversion on Rule ⌊SENDMORE⌋ we have $\Gamma \vdash \langle P', M_0^{\emptyset} \cup s[\mathtt{p}] : (v, \emptyset), \emptyset \rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta_0^{\emptyset} \cup s[\mathtt{p}] : (S, \emptyset) \rangle$, where $\Gamma \vdash M_0 \triangleright \Theta_0$. This is the premise we need to apply Rule ⌊SENDFIRST⌋ to $\langle s[\mathtt{p}]!\langle e \rangle.P', M_0^{\emptyset} \cup s[\mathtt{p}] : \epsilon, \emptyset \rangle = \langle [P]_E, M^{\emptyset}, \emptyset \rangle$. By this rule we deduce:

$$\Gamma \vdash \langle s[\mathtt{p}]!\langle e \rangle.P', M_0^{\emptyset} s[\mathtt{p}] : \epsilon, \emptyset \rangle \triangleright \langle s[\mathtt{p}] :!S.T \diamond \Theta_0^{\emptyset}, s[\mathtt{p}] : \texttt{void} \rangle$$

that is, $\Gamma \vdash \langle [P]_E, M^{\emptyset}, \emptyset \rangle \triangleright \langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$, given that $M^{\emptyset} = M_0^{\emptyset} \cup s[\mathtt{p}] : \epsilon$, $[\Delta]_E = s[\mathtt{p}] :!S.T$ and $\Theta^{\emptyset} = \Theta_0^{\emptyset}, s[\mathtt{p}] : \texttt{void}$ (the latter follows from $\Theta = \Theta_0, s[\mathtt{p}] : (!S', \Pi)$).

Again, we have $\langle \Delta \diamond \Theta \rangle \curvearrowright \langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ by Clause 1. of Def. 20. What is left to show is coherence of $\langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$. It is easy to see that Condition 1. is satisfied, because $\mathsf{OG}(!S.T)$ implies $\mathsf{OG}([\Delta^{live}]_E)$. As for Condition 2., it follows again from Lem. 11.

Case 3. *(in_s):*

Here $P = s[\mathtt{q}]?(\mathtt{p}, x).P'$ and $M = M_0 \cup s[\mathtt{p}] : \epsilon$. By assumption $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$. This judgement is deduced by Rule $\lfloor$RcvNext$\rfloor$, hence it has the form:

$$\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P', M_0 \cup s[\mathtt{p}] : \epsilon, E\rangle \triangleright \langle s[\mathtt{q}] :?(\mathtt{p}, S).T \diamond \Theta_0, s[\mathtt{p}] : \mathtt{void}\rangle$$

We will call the derivation tree obtained by the previous hypothesis D_1.

Through reconditioning we obtain: $[P]_E = P = s[\mathtt{q}]?(\mathtt{p}, x).P'$ and we show that $\Gamma \vdash \langle P, M^\emptyset, E\rangle \triangleright \langle \Delta' \diamond \Theta^\emptyset\rangle$. This judgement is again deduced by Rule $\lfloor$RcvNext$\rfloor$, hence it has the form:

$$\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P', M^\emptyset \cup s[\mathtt{p}] : \epsilon, E\rangle \triangleright \langle s[\mathtt{q}] :?(\mathtt{p}, S).T' \diamond \Theta^\emptyset, s[\mathtt{p}] : \mathtt{void}\rangle$$

By inversion we have $\Gamma \vdash \langle P', M^\emptyset \cup s[\mathtt{p}] : (v, \{\mathtt{q}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : T' \diamond s[\mathtt{p}] : (S, \{q\})\rangle$ and $\Gamma \vdash v : S$. We also have that $\Gamma \vdash M^\emptyset \triangleright \Theta^\emptyset$, by applying the Rules in Fig. 15. The previously obtained derivation tree will be called D_2. We now proceed by induction on the structure of P': we will prove that $T' = \mathtt{trm}(T)^0_{\{p\}}$.

- **Case** $P' = \mathbf{0}$**:** By using Rule $\lfloor$Inact$\rfloor$ we obtain that $\Gamma \vdash \langle \mathbf{0}, M^\emptyset \cup s[\mathtt{p}] : (v, \{\mathtt{q}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : \mathtt{end} \diamond s[\mathtt{p}] : (S, \{q\})\rangle$ and by Fig. 18 we have that $\mathtt{end} = \mathtt{trm}(\mathtt{end})^0_{\{p\}}$.
- **Case** $P' = \bar{a}[n]$ **or** $P' = a[\mathtt{p}](\alpha).P''$**:** This case is vacuously true as we are considering single session processes. Hence, there cannot be session initiations in a continuation.
- **Case** $P' = s[\mathtt{q}]!\langle e\rangle.P''$**:** We distinguish the following two cases, depending on the memory M:
  - **Case** $M_0(s[\mathtt{q}]) \neq \epsilon$**:** We have that on D_1 we will have that $T = \mathtt{tick}.!S_1.T_1$, since this derivation is obtained by using Rule $\lfloor$SendMore$\rfloor$. For the D_2 we need to show then that $T' = \mathtt{trm}(T)^0_{\{\mathtt{p}\}}$; for this is enough to observe that $T' = \mathtt{trm}(T)^0_{\{p\}} = !S_1.T_1$, by Fig. 18, and hence, we can apply Rule $\lfloor$SendFirst$\rfloor$. Finally, we can conclude by IH.
  - **Case** $M_0(s[\mathtt{q}]) = \epsilon$**:** We have that on D_1 we will have that $T = !S_1.T_1$, since this derivation is obtained by using Rule $\lfloor$SendFirst$\rfloor$. By inversion we obtain:

    $$\Gamma \vdash \langle P'', M_0' \cup s[\mathtt{q}] : (v', \emptyset) \cup s[\mathtt{p}] : (v, \{\mathtt{q}\}), E\rangle \triangleright \langle s[\mathtt{q}] : T_1 \diamond \Theta', s[\mathtt{p}] : (v, \{\mathtt{q}\}, s[\mathtt{q}] : (S_1, \emptyset)\rangle$$

    with $M_0 = M_0' \cup s[\mathtt{q}] : (v', \emptyset)$ and $\Theta = \Theta', s[\mathtt{q}] : (S_1, \emptyset)$.
    Notice now that on D_2 we have that the judgment will also be deduced by Rule $\lfloor$SendFirst$\rfloor$, hence, by inversion: $\Gamma \vdash \langle P', M_1^\emptyset \cup s[\mathtt{q}] : (v', \emptyset) \cup s[\mathtt{p}] : (v, \{\mathtt{q}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : T_1' \diamond \Theta_1^\emptyset, s[\mathtt{q}] : (S_1\emptyset), s[\mathtt{p}] : (S, \{q\})\rangle$, with $M^\emptyset = M_1^\emptyset \cup s[\mathtt{q}] : (v', \emptyset)$ and $\Theta^\emptyset = \Theta_1^\emptyset, s[\mathtt{q}] : (S_1\emptyset)$.
    We know by Fig. 18 that $T' = \mathtt{trm}(T)^0_{\{p\}} = !S_1.T_1'$. We need then to show that $T_1' = \mathtt{trm}(T_1)^1_{\{\mathtt{p}\}}$; this is done by induction on P'':
    - **Case** $P'' = \mathbf{0}$**:** In this case again, by Rule $\lfloor$Inact$\rfloor$ on D_1 we obtain that $\Gamma \vdash \langle \mathbf{0}, M_1^\emptyset \cup s[\mathtt{q}] : (v', \emptyset) \cup s[\mathtt{p}] : (v, \{\mathtt{p}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : \mathtt{end} \diamond \Theta_1^\emptyset, s[\mathtt{q}] : (S_1\emptyset), s[\mathtt{p}] : (S, \{\mathtt{p}\})\rangle$ and since by Fig. 18, we have that $\mathtt{trm}(\mathtt{end})^1_{\{\mathtt{p}\}} = \mathtt{end}$, then we can conclude in D_2 with Rule $\lfloor$Inact$\rfloor$.
    - **Case** $P'' = \bar{a}[n]$ **or** $P' = a[\mathtt{p}](\alpha).P''$**:** This case is vacuously true as we are considering single session processes. Hence, there cannot be session initiations in a continuation.
    - **Case** $P'' = s[\mathtt{q}]!\langle e\rangle.P''$**:** In this case, on D_1 we use Rule $\lfloor$SendMore$\rfloor$, therefore, we have that $T_1 = \mathtt{tick}.!S_2.T_2$ and since $\mathtt{trm}(T_1)^1_{\{\mathtt{p}\}} = T_1 = \mathtt{tick}.!S_2.T_2$ we have, by Fig. 18, that $T_1' = \mathtt{trm}(T_1)^1_{\{\mathtt{p}\}}$. Therefore, we can conclude D_2 by applying Rule $\lfloor$SendMore$\rfloor$ and the IH.

Case $P'' = s[\mathtt{q}]?(\mathtt{r}, x).P''$**:** In this case, on D_1 we need to distinguish cases depending on $\mathtt{r}$:

Case $\mathtt{r} \in \{\mathtt{p}\}$**:** In this case we have that $\mathtt{r} = \mathtt{p}$ and therefore, the judgment om D_1 is deduced from Rule ⌊RcvMore⌋, which means that $T_1 = \mathtt{tick}.?(\mathtt{p}, S_2).T_2$ and therefore $T_1 = \mathtt{trm}(\mathtt{tick}.?(\mathtt{p}, S_2).T_2)^1_{\{\mathtt{p}\}} = \mathtt{tick}.?(\mathtt{p}, S_2).T_2$, we then conclude by applying IH.

Case $\mathtt{r} \notin \{\mathtt{p}\}$**:** In this case we need to distinguish cases depending on the memory M_0:

Case $M_0(s[\mathtt{r}]) = \epsilon$**:** In this case we have that on D_1 the judgment is deduced from Rule ⌊RcvFirst⌋ and therefore $T_1 = ?(\mathtt{r}, S_2).T_2$ and $T'_1 = ?(\mathtt{r}, S_2).T'_2$. By IH and Fig. 18 we have that $T'_2 = \mathtt{trm}(T_2)^1_{\{\mathtt{p},\mathtt{r}\}}$ and therefore, $T'_1 = \mathtt{trm}(?(\mathtt{r}, S_2).T_2)^1_{\{\mathtt{p}\}}$, which is what we wanted to proof.

Case $M_0(s[\mathtt{r}]) \neq \epsilon$**:** In this case we apply Rule ⌊RcvMore⌋ on D_1 and therefore $T_1 = \mathtt{tick}.?(\mathtt{r}, S_2).T_2$. Observe then that in D_2 we will apply Rule ⌊RcvFirst⌋ and $T'_1 = ?(\mathtt{r}, S_2).T'_2$. Finally by IH and Fig. 18 we have that $\mathtt{trm}(T_1)^1_{\{\mathtt{p}\}} = ?(\mathtt{r}, S_2).T_2$ and $T'_2 = T_2$, hence concluding the proof.

Case $P'' = \mathtt{rec}\ X.P'''$**:** This case concludes by inversion on Rule ⌊Rec⌋ and applying the IH on the premises.

Case $P'' = \mathtt{pause}.P'''$**:** Note that since, by Fig. 18, function $\mathtt{trm}(\mathtt{pause}.T)^1_{\{\mathtt{p}\}} = \mathtt{pause}.T$, the case is straightforward by IH.

Case $P'' = P''' \mid Q$**,** $P'' = \mathtt{if}\ e\ \mathtt{then}\ P'''\ \mathtt{else}\ Q$**,** $P'' = \mathtt{emit}\ ev.P'''$ **and** $P'' = \mathtt{watch}\ ev\ \mathtt{do}\ P'''\{Q\}$**:** Again, these cases conclude by applying the IH on the premises of the Rules, obtained by inversion.

Case $P' = s[\mathtt{q}]?(\mathtt{r}, x).P''$**:** We distinguish cases depending on $\mathtt{r}$:

Case $\mathtt{r} \in \{\mathtt{p}\}$**:** This case proceeds as above, by considering that in D_1 and D_2 we apply ⌊RcvMore⌋. The equality is preserved, since this type is not changed by function $\mathtt{trm}(T)^0_{\{\mathtt{p}\}}$.

Case $\mathtt{r} \in \{\mathtt{p}\}$**:** We distinguish cases depending on the memory M_0:

Case $M_0(s[\mathtt{r}]) = \epsilon$**:** This case concludes by applying the IH, since we have that function $\mathtt{trm}(\cdot)^0_{\{\mathtt{p},\mathtt{r}\}}$ is applied recursively on the continuation T_1.

Case $M_0(s[\mathtt{r}]) \neq \epsilon$**:** This case concludes as Case $\mathtt{r} \in \{\mathtt{p}\}$, since the judgment on D_1 is deduced from ⌊RcvMore⌋.

Case $P' = \mathtt{rec}\ X.P''$**:** This case proceeds by applying the IH on the premises obtained by inversion Rule ⌊Rec⌋.

Case $P' = \mathtt{pause}.P''$**:** This case proceeds straightforward since function $\mathtt{trm}(\cdot)^0_{\{\mathtt{p}\}}$ does not change type $T = \mathtt{pause}.T_1$ and both D_1, D_2 use Rule ⌊Pause⌋ and conclude by IH.

Case $P' = P'' \mid Q$**,** $P' = \mathtt{if}\ e\ \mathtt{then}\ P''\ \mathtt{else}\ Q$**,** $P' = \mathtt{emit}\ ev.P''$ **and** $\mathtt{watch}\ ev\ \mathtt{do}\ P''\{Q\}$**:** These three cases are concluded by applying the IH on the premises obtained by inversion in each of the respective Rules (i.e., ⌊Conc⌋, ⌊If⌋,⌊Emit⌋, ⌊Watch⌋).

Note that once more, we have $\langle \Delta \diamond \Theta \rangle \curvearrowright \langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ by Clause 3. of Definition 20. Moreover, $\langle [\Delta]_E \diamond \Theta^{\emptyset} \rangle$ is coherent: Condition 1. holds since, assuming that $\langle \Delta \diamond \Theta \rangle$ is coherent since, by assumption $\mathsf{OG}(?(\mathtt{p}, S)).T$ is true, and Condition 2. follows from Lemma 11.

Case 4. *(in_s^2):*

Here $P = s[\mathtt{q}]?(\mathtt{p}, x).P'$ and $M = M_0 \cup \{s[\mathtt{p}] : (w, \Pi)\}$ with $\mathtt{q} \in \Pi$. By assumption $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$. This judgement is deduced by Rule $\lfloor$RcvMore$\rfloor$, hence it has the form:

$$\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P', M_0 \cup s[\mathtt{p}] : (w, \Pi), E\rangle \triangleright \langle s[\mathtt{q}] : \mathtt{pause}.?(\mathtt{p}, S).T \diamond \Theta_0, s[\mathtt{p}] : (S', \Pi)\rangle$$

Through reconditioning we obtain: $[P]_E = P = s[\mathtt{q}]?(\mathtt{p}, x).P'$ and $[\Delta]_E = s[\mathtt{p}] :?(\mathtt{p}, S).T$. By inversion on Rule $\lfloor$RcvMore$\rfloor$ we have $\mathtt{q} \in \Pi$, $\Gamma \vdash v : S$, $\Gamma \vdash w : S'$, $\Gamma \vdash M_0 \triangleright \Theta_0$, $\mathsf{OG}(T)$, and $\Gamma, x : S \vdash \langle P', M_0^\emptyset \cup s[\mathtt{p}] : (v, \{\mathtt{q}\}), \emptyset\rangle \triangleright \langle s[\mathtt{q}] : T \diamond \Theta_0^\emptyset, s[\mathtt{p}] : (S, \{\mathtt{q}\})\rangle$.

These statements may now be used as premises for applying Rule $\lfloor$RcvNext$\rfloor$ to the configuration $\langle [P]_E, M^\emptyset, \emptyset\rangle = \langle s[\mathtt{q}]?(\mathtt{p}, x).P', M_0^\emptyset \cup s[\mathtt{p}] : \epsilon, \emptyset\rangle$. By this Rule we obtain:

$$\Gamma \vdash \langle s[\mathtt{q}]?(\mathtt{p}, x).P', M_0^\emptyset \cup s[\mathtt{p}] : \epsilon, \emptyset\rangle \triangleright \langle s[\mathtt{q}] :?(\mathtt{p}, S).T \diamond \Theta_0^\emptyset,\ s[\mathtt{p}] : \mathtt{void}\rangle$$

This is the required judgement $\Gamma \vdash \langle [P]_E, M^\emptyset, \emptyset\rangle \triangleright \langle [\Delta]_E \diamond \Theta^\emptyset\rangle$. Note that once more, we have $\langle \Delta \diamond \Theta\rangle \curvearrowright \langle [\Delta]_E \diamond \Theta^\emptyset\rangle$ by Clause 1. of Def. 20. Moreover, $\langle [\Delta]_E \diamond \Theta^\emptyset\rangle$ is coherent: Condition 1. holds because $\mathsf{OG}(T)$, and Condition 2. follows from Lem. 11.

We now turn to the inductive cases, corresponding to rules (par_s), ($watch_s$) and (rec_s):

Case 5. *(par_s):*

Here $P = P_1 \mid P_2$, and $\langle P_1 \mid P_2, M, E\rangle\ddagger$ is deduced from $\langle P_1, M, E\rangle\ddagger$ and $\langle P_2, M, E\rangle\ddagger$. Notice that the two components P_1 and P_2 are run in the same memory M and set of events E. By assumption $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$. This judgement is deduced by Rule $\lfloor$Conc$\rfloor$, with $M_1 = M_2 = M$ and $E_1 = E_2 = E$, hence the deduction has the form:

$$\frac{\Gamma \vdash \langle P_i, M, E\rangle \triangleright \langle \Delta_i \diamond \Theta\rangle,\ i = 1, 2}{\Gamma \vdash \langle P_1 \mid P_2, M, E\rangle \triangleright \langle \Delta_1, \Delta_2 \diamond \Theta\rangle}$$

By induction, $\langle P_i, M, E\rangle\ddagger$ and $\Gamma \vdash \langle P_i, M, E\rangle \triangleright \langle \Delta_i \diamond \Theta\rangle$ imply $\Gamma \vdash \langle [P_i]_E, M^\emptyset, \emptyset\rangle \triangleright \langle [\Delta_i]_E \diamond \Theta^\emptyset\rangle$ for $i = 1, 2$. Moreover, $\langle \Delta_i \diamond \Theta\rangle \curvearrowright \langle [\Delta_i]_E \diamond \Theta^\emptyset\rangle$ and coherence of $\langle \Delta_i \diamond \Theta\rangle$ implies coherence of $\langle [\Delta_i]_E \diamond \Theta^\emptyset\rangle$. We want to show that $\Gamma \vdash \langle [P_1 \mid P_2]_E, M^\emptyset, \emptyset\rangle \triangleright \langle [\Delta_1, \Delta_2]_E \diamond \Theta^\emptyset\rangle$, and that coherence of $\langle \Delta_1, \Delta_2 \diamond \Theta\rangle$ implies coherence of $\langle [\Delta_1, \Delta_2]_E \diamond \Theta^\emptyset\rangle$. From Fig. 15, we have that $\Gamma \vdash M^\emptyset \triangleright \Theta^\emptyset$. We may then apply Rule $\lfloor$Conc$\rfloor$ to the premises $\Gamma \vdash M^\emptyset \triangleright \Theta^\emptyset$ and $\Gamma \vdash \langle [P_i]_E, M^\emptyset, \emptyset\rangle \triangleright \langle [\Delta_i]_E \diamond \Theta^\emptyset\rangle$ for $i = 1, 2$, to deduce $\Gamma \vdash \langle [P_1]_E \mid [P_2]_E, M^\emptyset, \emptyset\rangle \triangleright \langle [\Delta_1]_E, [\Delta_2]_E, \diamond \Theta^\emptyset\rangle$. By definition of reconditioning, $[P_1 \mid P_2]_E = [P_1]_E \mid [P_2]_E$ and $\langle [\Delta_1, \Delta_2]_E \diamond \Theta^\emptyset\rangle = \langle [\Delta_1]_E, [\Delta_2]_E \diamond \Theta^\emptyset\rangle$. Hence we have the required judgement $\Gamma \vdash \langle [P_1 \mid P_2]_E, M^\emptyset, \emptyset\rangle \triangleright \langle [\Delta_1, \Delta_2]_E, \diamond \Theta^\emptyset\rangle$. Let us check now coherence preservation. Assuming $Co\,\langle \Delta_1, \Delta_2 \diamond \Theta\rangle$, we want to show $Co\,\langle [\Delta_1, \Delta_2]_E \diamond \Theta^\emptyset\rangle$. To prove Condition 1., observe that $Co\,\langle \Delta_1, \Delta_2 \diamond \Theta\rangle$ implies $Co\,\langle \Delta_i \diamond \Theta\rangle$ for each $i = 1, 2$. Then by induction we have $Co\,\langle [\Delta_i]_E \diamond \Theta^\emptyset\rangle$. This implies $\mathsf{OG}([\Delta_i^{live}]_E)$ for each i, from which we deduce $\mathsf{OG}([\Delta_1^{live}]_E, [\Delta_2^{live}]_E)$, which implies $\mathsf{OG}([\Delta_1^{live}, \Delta_2^{live}]_E)$.

Moreover, from $\langle \Delta_i \diamond \Theta\rangle \curvearrowright \langle [\Delta_i]_E \diamond \Theta^\emptyset\rangle$ for $i = 1, 2$ we deduce $\langle \Delta_1, \Delta_2 \diamond \Theta\rangle \curvearrowright \langle [\Delta_1, \Delta_2]_E \diamond \Theta^\emptyset\rangle$ by Def. 20(4.). Then we may use Lem. 11 again to obtain Condition 2.

Case 6. ($watch_s$)*: Here* $P = \texttt{watch}\, ev\, \texttt{do}\, P_1\{P_2\}$ *and* $\langle \texttt{watch}\, ev\, \texttt{do}\, P_1\{P_2\}, M, E\rangle\ddagger$ *is deduced from* $\langle P_1, M, E\rangle\ddagger$*. The judgement* $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ *is deduced by Rule* $\lfloor$WATCH$\rfloor$ *and has the form:*

$$\Gamma \vdash \langle \texttt{watch}\, ev\, \texttt{do}\, P_1\{P_2\}, M, E\rangle \triangleright \langle s[\mathtt{p}] : T_1 \star_{ev} T_2 \diamond \Theta\rangle$$

By inversion on rule $\lfloor$WATCH$\rfloor$ *we obtain* $\Gamma \vdash \langle P_1, M, E\rangle \triangleright \langle s[\mathtt{p}] : T_1 \diamond \Theta\rangle$*, as well as* $\Gamma \vdash \langle P_2, M^{\emptyset}, E\rangle \triangleright \langle s[\mathtt{p}] : T_2 \diamond \Theta^{\emptyset}\rangle$ *and* $\mathsf{OG}(\Delta_2)$*. Note that it is not the case that* $T_1 = \mathsf{end}$ *because in this case the configuration* $\langle \texttt{watch}\, ev\, \texttt{do}\, P_1\{P_2\}, M, E\rangle$ *would be terminated and not suspended.*

Now, the form of $[P]_E$ *and* $[\Delta]_E$ *will depend on the presence or absence of* ev *in* E*, namely:*

1. *If* $ev \in E$ *then* $[P]_E = P_2$ *and* $[s[\mathtt{p}] : T_1 \star_{ev} T_2]_E = s[\mathtt{p}] : T_2$*; in this case, by inversion on rule* $\lfloor$WATCH$\rfloor$ *we obtain the required judgement* $\Gamma \vdash \langle P_2, M^{\emptyset}, E\rangle \triangleright \langle s[\mathtt{p}] : T_2 \diamond \Theta^{\emptyset}\rangle$*. Now, assume* $Co\,\langle \Delta \diamond \Theta\rangle$*. We want to show* $Co\,\langle s[\mathtt{p}] : T_2 \diamond \Theta^{\emptyset}\rangle$*. Condition 1. follows immediately from* $\mathsf{OG}(T_2)$*. As for Condition 2., notice that it follows from Lem. 11.*
2. *If* $ev \notin E$*, then* $[P]_E = \texttt{watch}\, ev\, \texttt{do}\, [P_1]_E\{P_2\}$ *and* $[s[\mathtt{p}] : T_1 \star_{ev} T_2]_E = s[\mathtt{p}] : [T_1]_E \star_{ev} T_2$*. By inversion on Rule* $\lfloor$WATCH$\rfloor$ *we get* $\Gamma \vdash \langle P_1, M, E\rangle \triangleright \langle s[\mathtt{p}] : T_1 \diamond \Theta\rangle$*, and by applying the IH, we can obtain that* $\Gamma \vdash \langle P_1, M, E\rangle \triangleright \langle s[\mathtt{p}] : [T_1]_E \diamond \Theta^{\emptyset}\rangle$ *and we can use this hypothesis to conclude using Rule* $\lfloor$WATCH$\rfloor$*. Now, for the second part, assume* $Co\,\langle s[\mathtt{p}] : T_1 \star_{ev} T_2 \diamond \Theta\rangle$*, we want to show that* $Co\,\langle s[\mathtt{p}] : [T_1]_E \star_{ev} T_2 \diamond \Theta\rangle$*. By inversion and IH we know then that* $Co\,\langle [s[\mathtt{p}] : T_1]_E \diamond \Theta^{\emptyset}\rangle$ *and since* $T_1 = \mathsf{end}$ *we also have that* $\mathsf{OG}(T_2)$*. Hence, it is true that* $Co\,\langle s[\mathtt{p}] : [T_1]_E \star_{ev} T_2 \diamond \Theta^{\emptyset}\rangle$*, satisfying Condition 1.. The second condition follows directly from Lemma 11.*

Case 7. *(*rec_s*):*

Here $P = (\texttt{rec}\ X\,.\,P')$ *and* $\langle(\texttt{rec}\ X\,.\,P'), M, E\rangle\ddagger$ *is deduced from*

$$\langle P'\{\texttt{pause}.\,\texttt{rec}\ X\,.\,P'/X\}), M, E\rangle\ddagger$$

The judgement $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta\rangle$ *is obtained from Rule* $\lfloor$REC$\rfloor$ *and is as follows:*

$$\Gamma \vdash \langle(\texttt{rec}\ X\,.\,P), M, E\rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta\rangle$$

by inversion, we have that $\Gamma, X : \mathsf{pause}.T \vdash \langle P, M^{\emptyset}, E\rangle \triangleright \langle s[\mathtt{p}] : T \diamond \Theta^{\emptyset}\rangle$ *and then by IH we have that* $\Gamma, X : \mathsf{pause}.T \vdash \langle P, M^{\emptyset}, E\rangle \triangleright \langle [s[\mathtt{p}] : T]_E \diamond \Theta^{\emptyset}\rangle$*. Then, it is enough to apply Rule* $\lfloor$REC$\rfloor$ *with the previous hypothesis used twice to obtain:*

$$\Gamma \vdash \langle(\texttt{rec}\ X\,.\,P), M^{\emptyset}, E\rangle \triangleright \langle [s[\mathtt{p}] : T]_E \diamond \Theta^{\emptyset}\rangle$$

to proof the first part of the theorem. Supposing then $Co\,\langle \Delta \diamond \Theta\rangle$ *and supposing that* $\langle \Delta \diamond \Theta\rangle \curvearrowright \langle [\Delta]_E \diamond \Theta^{\emptyset}\rangle$ *then* $\langle [\Delta]_E \diamond \Theta^{\emptyset}\rangle$ *is coherent. This follows directly by IH and Lem. 11 as the proof is reduced to checking each type inside* Δ*.*

□

We can now finally prove the subject reduction theorem. As most of our previous results, this theorem deals only with reachable configurations. Let $\rightsquigarrow^+$ denote the transitive closure of the relation $\rightsquigarrow$.

Theorem 3 (Subject Reduction)**.** *Let* C *be a reachable configuration and* $C \rightsquigarrow^+ C'$*. If* $\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta\rangle$ *then* $\Gamma \vdash C' \triangleright \langle \Delta' \diamond \Theta'\rangle$ *and* $\langle \Delta \diamond \Theta\rangle \approx\!\!\Rightarrow \langle \Delta' \diamond \Theta'\rangle$ *for some* Δ', Θ'*. Moreover if* $\langle \Delta \diamond \Theta\rangle$ *is coherent then* $\langle \Delta' \diamond \Theta'\rangle$ *is coherent.*

Proof. By induction on the length n of the reduction sequence $\rightsquigarrow^+$.

Case 1. $n = 1$.

We distinguish two more cases, depending on whether C is an initial configuration or not.

a) $C = \langle P, M, E\rangle$ is initial. In this case $C = \langle a[1](\alpha_1).P_1 \mid \ldots \mid a[n](\alpha_n).P_n \mid \bar{a}[n], \emptyset, \emptyset\rangle$ and $C \rightsquigarrow C'$ is a reduction deduced by Rule [Init], hence $C' = (\nu s)\langle P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_1\{s[n]/\alpha_n\}, M_s^{\emptyset}, \emptyset\rangle$. Moreover, the type judgement $\Gamma \vdash C \triangleright \langle \Delta \diamond \Theta\rangle$ is deduced using Rules $\lfloor$MInit$\rfloor$, $\lfloor$MAcc$\rfloor$ and $\lfloor$Conc$\rfloor$ and thus it has the form (i) $\Gamma \vdash C \triangleright \langle \emptyset \diamond \emptyset\rangle$. From (i), using inversion on Rules $\lfloor$Conc$\rfloor$ and $\lfloor$MAcc$\rfloor$ we obtain (ii) $\Gamma \vdash \langle P_i\{s[i]/\alpha_i\}, M_s^{\emptyset}, \emptyset\rangle \triangleright \langle s[i] : G_i\lfloor_i \diamond \Theta_s^{\emptyset}\rangle$. By applying Rule $\lfloor$Conc$\rfloor$ to (ii) we deduce (iii) $\Gamma \vdash \langle P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\}, M_s^{\emptyset}, \emptyset\rangle \triangleright \langle \Delta \diamond \Theta_s^{\emptyset}\rangle$. By Prop. 2 we get (iv) $Co\langle \Delta \diamond \Theta_s^{\emptyset}\rangle$. We may then apply $\lfloor$Res$\rfloor$ to (iii) and (iv) to get $\Gamma \vdash (\nu c)\langle P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\}, M_s^{\emptyset}, \emptyset\rangle \triangleright \langle \emptyset \diamond \emptyset\rangle$. Since $\langle \Delta \diamond \Theta\rangle = \langle \Delta' \diamond \Theta'\rangle$ we trivially have $\langle \Delta \diamond \Theta\rangle \Rrightarrow \langle \Delta' \diamond \Theta'\rangle$ and there is nothing to prove about coherence.

b) $C = (\nu s)\langle P, M, E\rangle$ and $C \rightsquigarrow C'$ is either a reduction deduced by Rule [Restr], or a tick transition deduced by Rule (tick). In both cases C' has the form $C' = (\nu s)\langle P', M', E'\rangle$ by Prop. 1. In the first case $C \longrightarrow C'$ is deduced by Rule [Restr] from $\langle P, M, E\rangle \longrightarrow \langle P', M', E'\rangle$ and we get the result by Lemma 10 (Reduction Lemma). In the second case $C \hookrightarrow_E C'$ is deduced by Rule (tick) from $\langle P, M, E\rangle\ddagger$, and we get the result by Lemma 12 (Suspension Lemma).

Case 2. $n > 1$.

Let $(C \rightsquigarrow (\nu s)\langle P_1, M_1, E_1\rangle \rightsquigarrow \cdots \rightsquigarrow (\nu s)\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \rightsquigarrow (\nu s)\langle P'_n, M_n, E_n\rangle = C'$.

By induction $\Gamma \vdash \langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \triangleright \langle \Delta_{n-1} \diamond \Theta_{n-1}\rangle$ and $\langle \Delta \diamond \Theta\rangle \Rrightarrow \langle \Delta_{n-1} \diamond \Theta_{n-1}\rangle$. Moreover if $\langle \Delta \diamond \Theta\rangle$ is coherent then $\langle \Delta_{n-1} \diamond \Theta_{n-1}\rangle$ is coherent. Consider now the last reduction:

$$(\nu s)\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \rightsquigarrow (\nu s)\langle P_n, M_n, E_n\rangle$$

There are two possibilities:

1. $\rightsquigarrow = \hookrightarrow_E$. In this case $(\nu s)\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \hookrightarrow_E (\nu s)\langle [P_{n-1}]_E, M_{n-1}^{\emptyset}, \emptyset\rangle = (\nu s)\langle P_n, M_n, E_n\rangle$. Since $(\nu s)\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle\ddagger$ if and only if $\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle\ddagger$ and $\Gamma \vdash \langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \triangleright \langle \Delta_{n-1} \diamond \Theta_{n-1}\rangle$, by Lem. 12 we deduce $\Gamma \vdash \langle [P_{n-1}]_{E_{n-1}}, M_{n-1}^{\emptyset}, \emptyset\rangle \triangleright \langle [\Delta_{n-1}]_{E_{n-1}} \diamond \Theta_{n-1}^{\emptyset}\rangle$, $\langle \Delta_{n-1} \diamond \Theta_{n-1}\rangle \curvearrowright_{E_{n-1}} \langle [\Delta_{n-1}]_{E_{n-1}} \diamond \Theta^{\emptyset}\rangle$ and $Co\,\langle [\Delta]_{E_{n-1}} \diamond \Theta^{\emptyset}\rangle$.

2. $\rightsquigarrow = \longrightarrow$. In this case $(\nu s)\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \longrightarrow (\nu s)\langle P_n, M_n, E_n\rangle$, and this reduction is deduced by Rule [Res] from $\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle \longrightarrow \langle P_n, M_n, E_n\rangle$, which in turn is deduced by the contextual Rules [Cont] and [Struct] from a reduction $\langle Q, M_{n-1}, E_{n-1}\rangle \longrightarrow \langle Q', M_n, E_n\rangle$ that is deduced by the computational rules only, where $P_{n-1} \equiv \mathcal{E}[Q]$ and $P_n \equiv \mathcal{E}[Q']$ for some evaluation context $\mathcal{E}$. This means that:

$$(\nu s)\langle P_{n-1}, M_{n-1}, E_{n-1}\rangle = (\nu s)\langle \mathcal{E}[Q], M_{n-1}, E_{n-1}\rangle$$

Then, assuming $\Gamma \vdash \langle Q, M_{n-1}, E_{n-1}\rangle \triangleright \langle \Delta_Q \diamond \Theta_Q\rangle$, and $\Gamma \vdash \langle \mathcal{E}[\mathbf{0}], M_{n-1}, E_{n-1}\rangle \triangleright \langle \Delta'' \diamond \Theta''\rangle$, by the typing rules [Conc] and [CRes] we have $\Delta_{n-1} = (\Delta_Q, \Delta'')$ and $\Theta_{n-1} = (\Theta_Q, \Theta'')$.
Since the reduction $\langle Q, M_{n-1}, E_{n-1}\rangle \longrightarrow \langle Q', M_n, E_n\rangle$ is deduced by computational rules only, by Lem. 10 we have $\langle \Delta_Q \diamond \Theta_Q\rangle \Rightarrow \langle \Delta'_Q \diamond \Theta'_Q\rangle$ and $\Gamma \vdash \langle Q', M_n, E_n\rangle \triangleright \langle \Delta'_Q \diamond \Theta'_Q\rangle$. Moreover, the same lemma states that if $\langle \Delta_Q, \Delta'' \diamond \Theta_Q, \Theta''\rangle$ is coherent, then $\langle \Delta'_Q, \Delta'' \diamond \Theta'_Q, \Theta''\rangle$ is coherent. From this we deduce that if $\langle \Delta_{n-1} \diamond \Theta_{n-1}\rangle$ is coherent then also $\langle \Delta' \diamond \Theta'\rangle$ is coherent.

□

5 Time-related properties

In this section we prove that our type system enforces some desirable "real-time" properties. More precisely, we prove that any configuration that is reachable from an initial configuration and complies with a global type G (the notion of G-compliance will be made precise below) satisfies the following properties:

P1. *Output persistence*: Every participant broadcasts exactly once during every instant;

P2. *Input timeliness*: Every unguarded input is matched by an output during the current instant, if not preceded by another input with equal source and target, or during the next instant, if not preempted.

To formalise Property P1, we first define the auxiliary property of *output readiness* for reachable configurations of the form $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$, which states that all participants occurring in P perform a broadcast during the current instant. Intuitively, a reachable configuration $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ represents the state of a running session at the start of an instant (or more precisely, before any message exchange or event emission within an instant).

Definition 23 (Output readiness)**.** *A reachable configuration $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ is* output-ready *if whenever $C \Downarrow (\nu s)\langle P', M', E'\rangle$, then $s[\mathtt{p}] \in vdom(M')$ for every $s[\mathtt{p}] \in nm(P)$.*

Note that only the participants $\mathtt{p}$ whose behaviour is nonterminated at the beginning of the instant (condition $s[\mathtt{p}] \in nm(P)$) are required to perform an output before the end of the instant (condition $s[\mathtt{p}] \in vdom(M')$). Then, output persistence for such configurations essentially amounts to requiring output readiness at every instant. For an initial configuration $\langle Q, \emptyset, \emptyset\rangle$, output persistence amounts to requiring output readiness for any configuration $(\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ such that $\langle Q, \emptyset, \emptyset\rangle \rightsquigarrow^{+} (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$.

Definition 24 (Output persistence)**.** *A reachable configuration C is* output-persistent if whenever $C \rightsquigarrow^{*} (\nu s)\langle P, M, E\rangle \Downarrow (\nu s)\langle P', M', E'\rangle$, *then $s[\mathtt{p}] \in vdom(M')$ for every $s[\mathtt{p}] \in nm(P)$.*

Again, only nonterminated participants $\mathtt{p}$ at the beginning of the last instant (condition $s[\mathtt{p}] \in nm(P)$) are required to perform an output. For instance, our auction protocol (cf. Section 2) satisfies output persistence although the terminated participant *Forwarder* does not output anything during the last instant.

We now formalise Property P2, which again rests on an auxiliary property defined only on reachable configurations of the form $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$, called input readiness.

Definition 25 (Input readiness)**.** *A reachable configuration $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ is* input ready *if whenever $C \Downarrow (\nu s)\langle \mathcal{E}[s[\mathtt{q}]?(\mathtt{p}, x).P'], M', E'\rangle$, then $s[\mathtt{p}] \in vdom(M')$.*

It is easy to see that $s[\mathtt{p}] \in vdom(M')$ implies $s[\mathtt{p}] : (v, \Pi \cup \{q\}) \in M'$ for some v, Π, namely q must have read a message from $\mathtt{p}$ in the current instant, otherwise the final configuration would not be suspended.

Input timeliness amounts to input readiness at every instant. For an initial configuration C, input timeliness amounts to requiring input readiness for the configurations derivable from C.

Definition 26 (Input timeliness)**.** *A reachable configuration C is* input timely *if whenever $C \rightsquigarrow^{*} (\nu s)\langle \mathcal{E}[s[\mathtt{q}]?(\mathtt{p}, x).P], M, E\rangle \Downarrow (\nu s)\langle P', M', E'\rangle$, then $s[\mathtt{p}] \in vdom(M')$.*

Example 5. $C = \langle (\nu s)(s[1]?(2, x).s[1]?(2, y).\mathbf{0} \mid s[2]!\langle v\rangle.s[2]!\langle v\rangle.\mathbf{0}), M_s, \emptyset\rangle$ *satisfies input timeliness*

$C' = \langle (\nu s)(s[1]?(2, x).\mathbf{0} \mid \mathtt{pause}.s[2]!\langle v\rangle.\mathbf{0}), M_s, \emptyset\rangle$ *does not satisfy input timeliness*

Indeed, in C the expectations of the two participants are dual and "well-timed": their first communication takes place in the first instant and their second communication takes place in the second instant.

In C', on the other hand, participant 1 is ready to receive a message from participant 2 in the first instant, and entitled to do so because she hasn't read previously from participant 2, but there is no available message from participant 2 in the first instant. In fact, C is G-compliant whereas C' is not even typable.

$$\Phi(\mathtt{p}\uparrow\langle S,\Pi\rangle.G) = \mathtt{p}\uparrow\langle S,\Pi\rangle.\Phi(G)$$

$$\Phi(\mathtt{pause}.G) = \Phi(\mathtt{tick}.G) = \Phi(\mathbf{t}) = \Phi(\mathtt{end}) = \mathtt{end}$$

$$\Phi(\mathtt{watch}\ ev\ \mathtt{do}\ G\ \mathtt{else}\ G') = \Phi(\mu\mathbf{t}.G) = \Phi(G)$$

$$\Phi(!S.T) = !S.\Phi(T)$$

$$\Phi(?(\mathtt{p},S).T) = ?(\mathtt{p},S).\Phi(T)$$

$$\Phi(\mathtt{pause}.T) = \Phi(\mathtt{tick}.T) = \Phi(\mathbf{t}) = \Phi(\mathtt{end}) = \mathtt{end}$$

$$\Phi(\langle T,T'\rangle^{ev}) = \Phi(\mu\mathbf{t}.T) = T$$

Figure 19: Flattening of saturated global and local types.

Observe that if $\Gamma \vdash C = \langle P, M^{\emptyset}, \emptyset\rangle \triangleright \langle \Delta \diamond \Theta\rangle$, the coherence of the configuration environment $\langle \Delta \diamond \Theta\rangle$ is not sufficient to ensure input timeliness of $(v\tilde{s})C$, because of the possibility of circular dependencies. Indeed, duality is a binary property which does not exclude n-ary circular dependencies such as that of the following process, which is reminiscent of the dining philosophers deadlock:

$$R = s[1]?(2,x).s[1]!\langle v_1\rangle.\mathbf{0} \mid s[2]?(3,y).s[2]!\langle v_2\rangle.\mathbf{0} \mid s[3]?(1,z).s[3]!\langle v_3\rangle.\mathbf{0} \tag{2}$$

Note that the configuration $(vs)C = vs\langle R, M_s^{\emptyset}, \emptyset\rangle$ is not deadlocked but *livelocked* in MRS. Indeed, as shown in Section 3.3, MRS is deadlock-free: thanks to our semantic rule (in_s) (cf. Fig. 7), all potential deadlocks due to missing messages are turned into livelocks. In fact, it is easy to see that $(vs)C = vs\langle R, M_s^{\emptyset}, \emptyset\rangle$ does not satisfy input timeliness, because all participants are waiting for each other. Since the configuration is reconditioned to itself at the following instant, it gives rise to a livelock.

In order to obtain input timeliness (and thus livelock-freedom), we need to require that the types of all participants be *projections of the same saturated* global type in the initial configuration of the session. We first define what it means for a configuration to implement a session specified by a saturated global type G. From now on, we will consider only saturated global types. As a consequence we will use the standard projection $\upharpoonright$ defined in (Fig. 12) and not the saturated projection $\lfloor$.

Definition 27 (G-compliant configuration)**.** *Let G be a saturated global type. A reachable configuration $C = (vs)\langle P, M, E\rangle$ is* G-compliant *if there exist Δ, Θ such that $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ and $\langle \Delta \diamond \Theta\rangle(s[\mathtt{p}]) = G \upharpoonright \mathtt{p}$ for every $\mathtt{p} \in \mathsf{Part}(G)$.*

The key property for proving both output persistence and input timeliness is the absence of circular dependencies in G-compliant configurations. The proof makes use of two *flattening* functions $\Phi(G)$ and $\Phi(T)$ on saturated global and local types. These functions extract from a type the sequence of I/O communications occurring in the first instant, forgetting about recursion and selecting the principal behaviour in watch types.

Definition 28 (Flattening)**.** *The* flattening functions $\Phi(G)$ *and* $\Phi(T)$ *are defined in Fig. 19.*

Since $\Phi(G)$ is again a global type, it can be projected on participants. It is easy to see that:

Lemma 13. $\Phi(G) \upharpoonright \mathtt{p} = \Phi(G \upharpoonright \mathtt{p})$

Proof. By induction on G.

□

We can now prove that G-compliant configurations are free of circular dependencies such as that exhibited by the three-philosopher process (Example (2)).

Lemma 14 (Absence of circular dependencies)**.** *Let $C = (\nu s)\langle P, M, E\rangle$ be a G-compliant configuration, and let $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\} \subseteq \mathsf{Part}(G)$, with $n \geq 2$. If $s[\mathtt{p}_i] \notin vdom(M)$ for all $i \in \{1, \ldots, n\}$, then there cannot exist $\mathcal{E}_1, \ldots, \mathcal{E}_n$ such that:*

$$\begin{array}{c} C \equiv \langle \mathcal{E}_1[s[\mathtt{p}_1]?(\mathtt{p}_2, x_1).P_1], M^{\emptyset}, \emptyset\rangle \\ C \equiv \langle \mathcal{E}_2[s[\mathtt{p}_2]?(\mathtt{p}_3, x_2).P_2], M^{\emptyset}, \emptyset\rangle \\ \vdots \\ C \equiv \langle \mathcal{E}_n[s[\mathtt{p}_n]?(\mathtt{p}_1, x_n).P_n], M^{\emptyset}, \emptyset\rangle \end{array} \tag{3}$$

Proof. By contradiction. Assume the situation in (3). By hypothesis C is G-compliant, hence $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ and $\langle \Delta \diamond \Theta\rangle(s[\mathtt{p}_i]) = G \upharpoonright \mathtt{p}_i$ for every p_i. From $s[\mathtt{p}_i] \notin vdom(M)$ it follows that $s[\mathtt{p}_i] \notin vdom(\Theta)$, thus $\langle \Delta \diamond \Theta\rangle(s[\mathtt{p}_i] = \Delta(s[\mathtt{p}_i])$. Therefore for all $i \in \{1, \ldots, n\}$ we have:

$$\Delta(s[\mathtt{p}_i]) = G \upharpoonright \mathtt{p}_i \tag{4}$$

Observe now that $C \equiv \langle \mathcal{E}_i[s[\mathtt{p}_i]?(\mathtt{p}_{(i+1) \bmod n}, x_i).P_i], M, E\rangle$ implies that for some T_i

$$\Phi(\Delta(s[\mathtt{p}_i])) = ?(\mathtt{p}_{(i+1) \bmod n}, S_i).\Phi(T_i) \tag{5}$$

Pick now an arbitrary $k \in \{1, \ldots, n\}$. It follows that

$$\begin{array}{rll} \Phi(G) \upharpoonright \mathtt{p}_k &= \Phi(G \upharpoonright \mathtt{p}_k) & \text{by Lemma 13} \\ &= \Phi(\Delta(s[\mathtt{p}_k])) & \text{by Equation 4} \\ &= ?(\mathtt{p}_{(k+1) \bmod n}, S_k).\Phi(T_i) & \text{by Equation 5} \end{array}$$

This means that $\Phi(G)$ is of the form:

$$\Phi(G) = \sigma_k.\mathtt{p}_{(k+1) \bmod n} \uparrow \langle S_k, \Pi_k\rangle.\Phi(G_k) \tag{6}$$

where σ_k is a possibly empty sequence of communications not involving $\mathtt{p}_k$ and not involving $\mathtt{p}_{(k+1) \bmod n}$ as a sender, and $\mathtt{p}_k \in \Pi_k$. This implies:

$$\Phi(G) \upharpoonright \mathtt{p}_{(k+1) \bmod n} = \sigma'_k.!S_k.T$$

where σ'_k is the projection of σ_k on $\mathtt{p}_{(k+1) \bmod n}$ Now there are two possible cases:

1. σ'_k is the empty sequence. Then

$$\begin{array}{rll} \Phi(G) \upharpoonright \mathtt{p}_{(k+1) \bmod n} &= !S_k.T & \\ &\neq ?(\mathtt{p}_{(k+2) \bmod n}, S_{k+1}).T' & \text{by Equation 5} \\ &= \Phi(\Delta(s[\mathtt{p}_{(k+1) \bmod n}])) & \text{by Equation 4} \\ &= \Phi(G \upharpoonright \mathtt{p}_{(k+1) \bmod n}) & \text{by Lemma 13} \end{array}$$

   This inequality contradicts Lemma 13.

2. σ'_k starts with $?(\mathtt{p}_{(k+2) \bmod n}, S_{k+1}).T'$. Then we iterate the reasoning until we reach the $\mathtt{p}_k$ we started with (which we are sure to reach since the number of participants is finite), and at this point we have a contradiction since by hypothesis σ_k does not contain $\mathtt{p}_k$ in Equation (6).

This concludes the proof.

□

Corollary 2. *Let $C = (\nu s)\langle P, M, E\rangle$ be a G-compliant configuration such that $C\ddagger$. Then $P = \mathcal{E}[s[\mathtt{q}]?(\mathtt{p}, x).P']$ implies $s[\mathtt{p}] \in vdom(M)$.*

Proof. By contradiction. Suppose that $s[\mathtt{p}] \notin vdom(M)$. Since C is G-compliant, we know that $\Gamma \vdash \langle P, M, E\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ and $\langle \Delta \diamond \Theta\rangle(s[\mathtt{r}]) = G \upharpoonright \mathtt{r}$ for all $\mathtt{r} \in \mathsf{Part}(G)$. From $s[\mathtt{p}] \notin vdom(M)$ it follows that $s[\mathtt{p}] \notin vdom(\Theta)$, hence $\langle \Delta \diamond \Theta\rangle(s[\mathtt{p}]) = \Delta(s[\mathtt{p}])$. This means that $\Delta(s[\mathtt{p}]) = G \upharpoonright \mathtt{p}$. On the other hand, given the shape of P, we know that it must be $\Phi(G \upharpoonright \mathtt{q}) = \Phi(\langle \Delta \diamond \Theta\rangle(s[\mathtt{q}])) = \{!S_{\mathtt{q}}.\}?(\mathtt{p}, S).\Phi(T)$ for some $S_{\mathtt{q}}, S$ and T, where the notation $\{!S_{\mathtt{q}}.\}$ means that the initial output is possibly missing (in case $s[\mathtt{q}] \notin vdom(M)$). By Lemma 13 $\Phi(G \upharpoonright \mathtt{q}) = \Phi(G) \upharpoonright \mathtt{q}$. This means that $\Phi(G)$ is of the form:

$$\Phi(G) = \sigma.\mathtt{p} \uparrow \langle S, \Pi\rangle.\Phi(G') \tag{7}$$

where σ is a possibly empty sequence of communications not involving $\mathtt{q}$ as a receiver nor $\mathtt{p}$ as a sender, and $\mathtt{q} \in \Pi$. Now, consider the projection $\Phi(G) \upharpoonright \mathtt{p}$ of $\Phi(G)$ on $\mathtt{p}$. Note that the projection of σ on $\mathtt{p}$ cannot be empty, because in this case we would have $\Phi(G) \upharpoonright \mathtt{p} = !S.T_{\mathtt{p}}$ for some $T_{\mathtt{p}}$, contradicting the fact that C is suspended. Then σ must consist of inputs by $\mathtt{p}$. This means that $P = \mathcal{E}[s[\mathtt{p}]?(\mathtt{r}, x).R]$. Since C is suspended, we know that either $s[\mathtt{r}] \notin vdom(M)$ or $s[\mathtt{r}] : (v, \Pi \cup \{p\}) \in M$ for some v, Π. In the latter case, by Rule $\lfloor$RcvMore$\rfloor$ we would have $\Delta(s[\mathtt{p}]) = \mathtt{pause}.?(\mathtt{r}, S').\Phi(T') = G \upharpoonright \mathtt{p}$ for some S' and T'. Then $\Phi(G \upharpoonright \mathtt{p}) = \Phi(G) \upharpoonright \mathtt{p}$ would be $\mathtt{end}$, and thus so would $\Phi(G)$, contradicting equation (7), where σ is supposed to consist only of communications. Therefore it must be $s[\mathtt{r}] \notin vdom(M)$. But now we have for participant $\mathtt{r}$ the same hypotheses that we had for $\mathtt{p}$ in the beginning. Hence we can iterate the reasoning and since the number of participants is finite this leads us to the circular situation (3), which by Lemma 14 is impossible. □

We will prove now that the above lemma entails input readiness, and as a consequence, also input timeliness. We first show that compliance with a global type is preserved along execution.

Lemma 15. *Let C be a G-compliant configuration and $\langle \Delta \diamond \Theta\rangle$ be the corresponding configuration environment. If $C \longrightarrow C'$ (resp., $C \hookrightarrow_E C'$), then there exists G' with corresponding configuration environment $\langle \Delta' \diamond \Theta'\rangle$ such that C' is G'-compliant and $\langle \Delta \diamond \Theta\rangle \Rightarrow \langle \Delta' \diamond \Theta'\rangle$ (resp., $\langle \Delta \diamond \Theta\rangle \curvearrowright_E \langle \Delta' \diamond \Theta'\rangle$).*

Proof. By induction on the inference of $\longrightarrow$ (resp., $\hookrightarrow_E$).

□

Lemma 16 (G-compliance implies input readiness)**.** *Every G-compliant configuration $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ satisfies input readiness.*

Proof. Let $C \Downarrow (\nu s)\langle P', M', E'\rangle = C'$, where $P' = \mathcal{E}[s[\mathtt{q}]?(\mathtt{p}, x).P'']$. By Lemma 15, C' is G'-compliant for some G'. Then $s[\mathtt{p}] \in vdom(M')$ by Corollary 2.

□

Theorem 4 (G-compliance implies input timeliness)**.** *Every G-compliant configuration $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ satisfies input timeliness.*

Proof. Suppose $C \rightsquigarrow^* (\nu s)\langle \mathcal{E}[s[\mathtt{q}]?(\mathtt{p}, x).P'], M, E\rangle = C'$ and $C'\ddagger$. The proof is by induction on the number n of $\hookrightarrow_E$ steps in the execution $\rightsquigarrow^*$. If $n = 0$ then we are in the case $C \Downarrow C'$ and we have the result by Lemma 16.

If $n > 0$, the result follows again by Lemma 16, observing that by Lemma 15 G-compliance is preserved by execution, and that every $\hookrightarrow_E$ step gives rise again to a configuration of the form $(\nu s)\langle Q, M^{\emptyset}, \emptyset\rangle$. □

We proceed now to prove output persistence.

Lemma 17 (*G*-compliance implies output readiness)**.** *Let* $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle$ *be a G-compliant configuration. If* $C \Downarrow (\nu s)\langle P', M', E\rangle$*, then* $s[\mathtt{p}] \in vdom(M')$ *for every* $s[\mathtt{p}] \in nm(P)$*.*

Proof. Let $C' = (\nu s)\langle P', M', E'\rangle$. Since C is G-compliant, by Lemma 15 C' is G'-compliant for some G'. Therefore there exist Δ, Θ such that $\Gamma \vdash \langle P', M', E'\rangle \triangleright \langle \Delta \diamond \Theta\rangle$ and $\langle \Delta \diamond \Theta\rangle(s[\mathtt{p}]) = G' \upharpoonright \mathtt{p}$ for every $\mathtt{p} \in \mathsf{Part}(G')$. Now, the statement $(\nu s)\langle P', M', E'\rangle\ddagger$ is deduced by Rule ($restr_s$) from $\langle P', M', E'\rangle\ddagger$.

The proof then proceeds by contradiction. Suppose there exists a $s[\mathtt{p}] \in nm(P)$ such that $s[\mathtt{p}] \notin vdom(M')$. We have $\langle \Delta \diamond \Theta\rangle(s[\mathtt{p}]) = (\Delta(s[\mathtt{p}])) = T$ and by coherence $\mathsf{OG}(T)$. This means that

$$\Phi(T) = \sigma.!S.T'$$

for some σ, S, T' and there are three possible cases for σ:

1. σ is empty, so T is derived using rule $\lfloor$SENDFIRST$\rfloor$:

   $$\Gamma \vdash \langle s[\mathtt{p}]!\langle e\rangle.R, M'' \cup \{s[\mathtt{p}] : \epsilon\}, E\rangle \triangleright \langle \Delta, s[\mathtt{p}] :!S.T \diamond \Theta,\ s[\mathtt{p}] : \mathtt{void}\rangle$$

   with $P' \equiv \mathcal{E}[s[\mathtt{p}]!\langle e\rangle.R]$ and $M' = M'' \cup \{s[\mathtt{p}] : \epsilon\}$ but this is a contradiction since $\langle s[\mathtt{p}]!\langle e\rangle.R, M'' \cup \{s[\mathtt{p}] : \epsilon\}, E\rangle$ cannot be suspended.

2. σ is a non empty sequence of inputs and T is derived using rule $\lfloor$RCVFIRST$\rfloor$:

   $$\Gamma \vdash \langle s[\mathtt{p}]?(\mathtt{q}, x).R, M'' \cup \{s[\mathtt{q}] : (v, \Pi)\}, E\rangle \triangleright \langle \Delta, s[\mathtt{p}] :?(\mathtt{q}, S).T \diamond \Theta,\ s[\mathtt{q}] : (S, \Pi)\rangle$$

   with $P' \equiv \mathcal{E}[s[\mathtt{p}]?(\mathtt{q}, x).R]$ and $M' = M'' \cup \{s[\mathtt{q}] : (v, \Pi)\}$ with $\mathtt{p} \notin \Pi$, but this is a contradiction since $\langle s[\mathtt{p}]?(\mathtt{q}, x).R, M'' \cup \{s[\mathtt{q}] : (v, \Pi)\}, E\rangle$ cannot be suspended.

3. σ is a non empty sequence of inputs and T is derived using rule $\lfloor$RCVNEXT$\rfloor$:

   $$\Gamma \vdash \langle s[\mathtt{p}]?(\mathtt{q}, x).R, M'' \cup \{s[\mathtt{q}] : \epsilon\}, E\rangle \triangleright \langle \Delta, s[\mathtt{p}] :?(\mathtt{q}, S).T \diamond \Theta,\ s[\mathtt{q}] : \mathtt{void}\rangle$$

   with $P' \equiv \mathcal{E}[s[\mathtt{p}]?(\mathtt{q}, x).R]$ and $M' = M'' \cup \{s[\mathtt{q}] : \epsilon\}$. This is a contradiction since by Corollary 2 we should have $s[\mathtt{q}] \in vdom(M')$.

Hence in all cases we reach a contradiction, concluding the proof.

□

Theorem 5 (*G*-compliance implies output persistence)**.** *Let C be a G-compliant configuration. If* $C \rightsquigarrow^* (\nu s)\langle P_0, M_0, E_0\rangle \Downarrow (\nu s)\langle P', M', E'\rangle$*, then* $s[\mathtt{p}] \in vdom(M')$ *for every* $s[\mathtt{p}] \in nm(P_0)$*.*

Proof. Let $C = (\nu s)\langle P, M^{\emptyset}, \emptyset\rangle \rightsquigarrow^* (\nu s)\langle P_0, M_0, E_0\rangle \Downarrow (\nu s)\langle P', M', E'\rangle$. We want to show that $s[\mathtt{p}] \in vdom(M')$ for every $s[\mathtt{p}] \in nm(P_0)$. The proof is by induction on the number n of $\hookrightarrow_E$ steps in the execution sequence $\rightsquigarrow^*$. If $n = 0$ then we are in the case $C \Downarrow C'$ and we have the result by Lemma 17. If $n > 0$, the result follows again by Lemma 17, since by Lemma 15 G-compliance is preserved by execution and every $\hookrightarrow_E$ step gives rise again to a configuration of the form $(\nu s)\langle Q, M^{\emptyset}, \emptyset\rangle$.

□

6 Related Work

Most related work has been already mentioned in the introduction and throughout the paper. Here we briefly discuss other relevant literature. In the realm of (multiparty) session types, forms of reactive behaviours have been addressed via constructs for exceptions, events, and run-time adaptation. This way, e.g., interactional exceptions for binary sessions were developed in [12]; an associated type system ensures consistent handling of exceptions. The work [11] extends this approach to multiparty sessions. In general, there is a tension between forms of (interactional) exceptions and behavioural type systems, mainly due to the linearity enforced by such systems, which conflicts with the possibility of not (fully) consuming behaviours abstracted by types. The works [26, 27] address this tension for binary sessions by appealing to *affinity* rather than to linearity. A similar approach is adopted in [17] for a functional programming language. The recent work [9] gives an alternative treatment of non-determinism and control effects within a Curry-Howard interpretation of binary session types; the proposed framework allows to represent exceptions. Concerning events and adaptation, the work [21] is the first to integrate events within a session-typed framework, supported by dynamic type inspection (a type-case construct). This work, however, is limited to binary sessions; the work [16] extends this framework to the multiparty setting, with the aim of handling run-time adaptation of choreographies. None of these works supports declarative or timed conditions, which are naturally expressible using synchronous programming constructs. To our knowledge, the only prior work that considers (unreliable) broadcasting in session types is [20], which focuses on binary sessions. The work in [15] develops run-time verification techniques for interruptible, multiparty conversations. Also, the work [13] proposes protocol types for handling partial failures and ensuring absence of orphan messages and deadlocks, among other properties.

The most closely related work is that in [2, 10], which encodes of a binary session π-calculus into the synchronous reactive language ReactiveML. Exploiting the duality between the two partners in binary sessions, this encoding simulates messages-over-channels in the session π-calculus by values-over-events in ReactiveML, slicing every session into a sequence of atomic instants: each instant corresponds to exactly one step in the protocol of the session. In contrast, instead of encoding a multiparty session calculus into RML, here we pursue a different goal: devise a minimal extension of a multiparty session calculus that accommodates reactive features, and provide a session type system that ensures the usual session properties together with some new semantic properties of interest: output persistence, input lock-freedom, safe event handling.

7 Conclusion

We have developed a typed framework for multiparty sessions by building upon MRS, a new process calculus that integrates constructs from session-based concurrency with constructs from synchronous reactive programming (SRP). The calculus MRS accounts for broadcast communication, logical instants, and preemption – all of which are hard to represent in existing process languages for sessions, usually based on the π-calculus. For instance, a session π-calculus with broadcasting has been studied in [20], but it does not support time-dependent interactions nor reactivity. Indeed, there are useful interaction patterns, such as *hot-service replacement* [1], which are representable in SRP but not in asynchronous calculi. The semantics of MRS ensures typical properties of SRP, such as deadlock-freedom and reactivity.

Our type system for MRS crucially relies on *saturation* for global types, a notion that we developed to address the subtle distinction between explicit and implicit pauses, and to capture the "timing" of a protocol interaction within the global type itself. Another salient feature of our static analysis is a new notion of duality, suited to our broadcast setting, in which outputs do not need to be matched by inputs. Our notion of duality is also "time-aware" in that it requires dual participants to have matching pauses. The benefits of our integration reflect also in the semantic properties enforced by our type system: besides

classical session safety properties, our static analysis guarantees two new time-related properties that seem to be desirable for sessions in a reactive setting: input timeliness and output persistence.

Design choices

- In our calculus MRS, the properties of deadlock-freedom and (bounded) reactivity are enforced by the operational semantics, while the classical properties of sessions, as well as the new time-related properties (input timeliness and output persistence), are enforced by our specific session type system. It could be argued that with a type system at hand, a more liberal semantics could have been used for the calculus, letting the type system take care also of the deadlock-freedom and reactivity properties. The reason for our choice is that deadlock-freedom and reactivity are essential properties of SRP: they are required for the synchronous reactive model to make sense. Hence they should be an integral part of the calculus itself. This is the case also for real SRP languages such as ReactiveML.
- Saturation could have been conceived as a well-formedness property of types rather than as an operation on them. In other words, our type system could have been designed so as to enforce the explicit separation of instants rather than relying on it. For instance, we could have defined a "well-timedness" predicate on global types requiring that subsequent broadcasts from the same sender and recursion unfoldings be separated by pauses. This way, we could have omitted some suspension rules ((out_s) and (in_s^2)) from the semantics, and the "More" typing rules from the type system. However, we chose the latter solution to avoid blurring the readability of real-world programs with the presence of too many pauses. The former solution would become an attractive option if combined with a "pause inference" mechanism.

Future work Directions for future work include extending our model with mechanisms for run-time monitoring, adaptation and interactional exceptions, as well as developing an implementation for our model in a programming language such as ReactiveML (RML). Along these lines, the recent work [10] proposes an encoding of a *binary* session calculus $S\pi$ into RML. In essence, this encoding simulates messages-over-channels in $S\pi$ by values-over-events in RML, and slices every session into a sequence of instants, each consisting of a single output by one partner and a single input by the other partner.

In particular, the `send` and `recv` constructs of $S\pi$ are encoded as follows, where $\texttt{await}\, ev_c(x)\, \texttt{in}\, [\![P]\!]$ is a program that waits for the valued event $ev_c(v)$ and then replaces the value v for x in the continuation P:

$$(*) \qquad \begin{array}{lcl} [\![\texttt{send}\, c\,\langle e\rangle.P]\!] & = & \texttt{emit}\, ev_c(e); \texttt{pause}; [\![P]\!] \\ [\![\texttt{recv}\, c(x).P]\!] & = & \texttt{await}\, ev_c(x)\, \texttt{in}\, [\![P]\!] \end{array}$$

In RML, the evaluation of the expression carried by a valued event $ev_c(e)$ only completes at the end of the instant. Therefore, in the encoding of the `recv` construct, the continuation $[\![P]\!]\{val(e)/x\}$ can only start at the next instant. This is why a `pause` is inserted after $\texttt{emit}\, ev_c(e)$ in the encoding of the `send` construct.

However, this encoding does not carry over to the multiparty case. Consider the following process, where $s[\mathtt{p}]!\langle \mathtt{q}, v\rangle.P$ represents an output from participant $\mathtt{p}$ to participant $\mathtt{q}$ in session s, followed by P:

$$s[1]!\langle 2, v\rangle.\mathbf{0} \mid s[2]!\langle 3, v'\rangle.s[2]?(1, x).\mathbf{0} \mid s[3]?(2, y).\mathbf{0}$$

If we used an encoding similar to $(*)$ for this process, then a `pause` would be inserted after the output in the second participant, preventing the subsequent input to take place in the first instant. Hence the message sent by the first participant would be lost. This means that the encoding would not preserve the $S\pi$ process behaviour, and in particular it would transform a lock-free $S\pi$ process into an input-locked RML program.

By contrast, in the present work we adopted a different approach to the integration of multiparty sessions and SRP, by trying to devise a minimal extension of multiparty $S\pi$ that accommodates reactive features, and by providing a session type system that ensures classical session properties together with some time-related properties that seem to be of interest for "reactive sessions".

References

[1] A. Almeida Matos, G. Boudol, and I. Castellani. Typing noninterference for reactive programs. *J. Log. Algebr. Program.*, 72(2):124–156, 2007.

[2] J. Arias, M. Cano, and J. A. Pérez. A reactive interpretation of session-based concurrency. In *Proc. REBLS'16*, 2016.

[3] M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. Compliance and subtyping in timed session types. In *FORTE'15, DisCoTec 2015, Grenoble, France, June 2-4, 2015*, pages 161–177, 2015.

[4] G. Berry and G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[5] L. Bocchi, M. Murgia, V. T. Vasconcelos, and N. Yoshida. Asynchronous timed session types - from duality to time-sensitive processes. In *ESOP'19, ETAPS'19, Prague, Czech Republic, April 6-11, 2019*, pages 583–610, 2019.

[6] L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *CONCUR'14, Rome, Italy, September 2-5, 2014. Proceedings*, pages 419–434, 2014.

[7] F. Boussinot and R. de Simone. The SL synchronous language. *Software Engineering*, 22(4):256–266, 1996.

[8] F. Boussinot, B. Monasse, and J.-F. Susini. Reactive programming of simulations in physics. *International Journal of Modern Physics C*, 26(12):1550132, 2015.

[9] L. Caires and J. A. Pérez. Linearity, control effects, and behavioral types. In *ESOP'17, ETAPS'17, Uppsala, Sweden, April 22-29, 2017*, pages 229–259, 2017.

[10] M. Cano, J. Arias, and J. A. Pérez. Session-based concurrency, reactively. In *FORTE'17, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017*, pages 74–91, 2017.

[11] S. Capecchi, E. Giachino, and N. Yoshida. Global escape in multiparty sessions. *Mathematical Structures in Computer Science*, 26(2):156–205, 2016.

[12] M. Carbone, K. Honda, and N. Yoshida. Structured interactional exceptions in session types. In *CONCUR'8, Toronto, Canada, August 19-22, 2008*, pages 402–417, 2008.

[13] T. Chen, M. Viering, A. Bejleri, L. Ziarek, and P. Eugster. A type theory for robust failure handling in distributed systems. In *FORTE'16, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, pages 96–113, 2016.

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

[15] R. Demangeon, K. Honda, R. Hu, R. Neykova, and N. Yoshida. Practical interruptible conversations: distributed dynamic verification with multiparty session types and python. *Formal Methods in System Design*, 46(3):197–225, 2015.

[16] C. Di Giusto and J. A. Pérez. Event-based run-time adaptation in communication-centric systems. *Formal Asp. Comput.*, 28(4):531–566, 2016.

[17] S. Fowler, S. Lindley, J. G. Morris, and S. Decova. Exceptional Asynchronous Session Types: Session Types Without Tiers. In *Proc. POPL'19*, pages 1–29, New York, NY, USA, 2019. ACM.

[18] K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *Proc. ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

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

[20] D. Kouzapas, R. Gutkovas, and S. J. Gay. Session types for broadcasting. In *PLACES 2014*, volume 155 of *EPTCS*, pages 25–31, 2014.

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

[22] L. Mandel, C. Pasteur, and M. Pouzet. Time refinement in a functional synchronous language. *Sci. Comput. Program.*, 111:190–211, 2015.

[23] L. Mandel and M. Pouzet. ReactiveML: a reactive extension to ML. In *PPDP'05, July 11-13 2005, Lisbon, Portugal*, pages 82–93. ACM, 2005.

[24] R. Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, LFCS, University of Edinburgh, 1991.

[25] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. CUP, 1999.

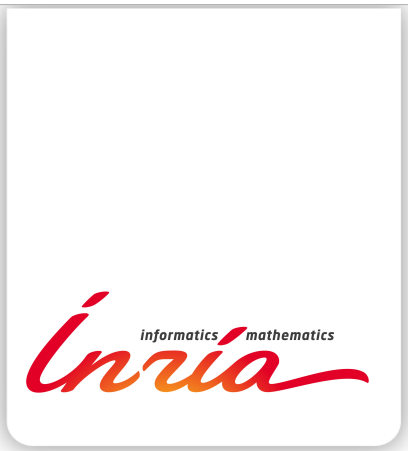
[26] D. Mostrous and V. T. Vasconcelos. Affine sessions. In *COORDINATION'14, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, pages 115–130, 2014.

[27] F. Pfenning and D. Griffith. Polarized substructural session types. In *FoSSaCS 2015, ETAPS'15, London, UK, April 11-18, 2015*, pages 3–22, 2015.

[28] D. Potop-Butucaru, S. A. Edwards, and G. Berry. *Compiling Esterel*. Springer, 2007.

[29] R. C. M. Santos, G. F. Lima, F. Sant'Anna, R. Ierusalimschy, and E. H. Haeusler. A memory-bounded, deterministic and terminating semantics for the synchronous programming language céu. In *Proceedings of LCTES 2018*, pages 1–18. ACM, 2018.

[30] O. Tardieu and R. d. Simone. Loops in Esterel. *ACM Trans. Embed. Comput. Syst.*, 4(4):708–750, Nov. 2005.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399