Deconfined Global Types for Asynchronous Sessions

Francesco Dagnino, Paola Giannini, Mariangiola Dezani-Ciancaglini

Deconfined Global Types for Asynchronous Sessions

Francesco Dagnino[1], Paola Giannini[2], and Mariangiola Dezani-Ciancaglini[3]

[1] DIBRIS, Università di Genova, Italy
[2] DiSIT, Università del Piemonte Orientale, Alessandria, Italy
[3] Dipartimento di Informatica, Università di Torino, Italy

Abstract. Multiparty sessions with asynchronous communications and global types play an important role for the modelling of interaction protocols in distributed systems. In designing such calculi the aim is to enforce, by typing, good properties for all participants, maximising, at the same time, the behaviours accepted. The global types presented in this paper improve the state-of-the-art by extending the set of typeable asynchronous sessions and preserving decidability of type checking together with the key properties of Subject Reduction, Session Fidelity and Progress.

Keywords: Communication-based programming · Multiparty sessions · Global types

1 Introduction

Multiparty sessions [17,18] are at the core of communication-based programming, since they formalise message exchange protocols. A key choice in the modelling is synchronous versus asynchronous communications, giving rise to synchronous and asynchronous multiparty sessions. In the multiparty session approach *global types* play the fundamental role of describing the whole scenario, while the behaviour of participants is implemented by processes. A natural question is when a set of processes agrees with a global type. The straightforward answer is the design of type assignment systems relating processes and global types. Global types are *projected* onto participants to get the local behaviours prescribed by the protocol. In conceiving such systems one wants to permit all possible typings which guarantee desirable properties: the mandatory Subject Reduction, but also Session Fidelity and Progress. *Session Fidelity* [17,18] means that the content and the order of exchanged messages respect the prescriptions of the global type. *Progress* [12,9] requires that all participants willing to communicate will be able to do it and, in case of asynchronous communication, also that all sent messages (which usually are in a queue) will be received.

A standard way of getting more permissive typings is through *subtyping* [14]. Following the *substitution principle* [20], we can safely put a process of some type where a process of a bigger type is expected. For synchronous multiparty sessions the natural subtyping allows less inputs and more outputs [11]. This

subtyping is not only correct, but also *complete*, that is, any extension of this subtyping would be unsound [15]. A powerful subtyping for asynchronous sessions was proposed in [21] and recently proved to be complete [16]. The key idea of this subtyping is anticipating outputs before inputs to improve efficiency. The drawback of this subtyping is its undecidability [5,19]. To overcome this problem, some decidable restrictions of this subtyping were proposed [5,19,6] and a sound, but not complete, decision algorithm, is presented in [4].

Asynchronous communications better represent the exchange of messages between participants in different localities, and are more suitable for implementations. So it is interesting to find alternatives to subtyping which increase typability of asynchronous multiparty sessions. Recently a more permissive design of global types has been proposed [8]: it is based on the simple idea of splitting outputs and inputs in the syntax of global types. In this way outputs can anticipate inputs. Multiparty sessions are typed by *configuration types*, which are pairs of global types and queues. The freedom gained by this definition is rather confined in [8], whose main focus was to define an event structure semantics for asynchronous multiparty sessions. In particular global types must satisfy well-formedness conditions which strongly limit their use.

In the present paper we start from the syntax of global types in [8], significantly enlarging the set of allowed global types which are well formed. In this way we obtain a decidable type system in which we are able to type also an example requiring a subtyping which fails for the algorithm in [4]. On the negative side, we did not find a global configuration in our system for the running example of [4]. The well-formedness of global types must guarantee that participants in different branches of choices have coherent behaviours and that all sent messages found the corresponding readers. This last condition is particularly delicate for cyclic computations in which the number of unread messages may be unbounded. Our type system gains expressivity by:
- requiring to a participant the knowledge of the chosen branch only when her behaviour depends on that;
- allowing an unbound number of unread messages when all of them will be eventually read.

Our type system enjoys Subject Reduction, Session Fidelity and Progress.

We illustrate the proposed calculus with an example in which the number of unread messages is unbounded. We choose this example since typing this session in standard type systems for multiparty sessions requires a subtyping which is not derivable by the algorithm in [4]. In fact this session is Example 24 of that paper in our notation. In addition this example is not typeable in [8]. The process $P = \mathsf{q}?\{\lambda_1; P_1, \lambda_2; P_2\}$ waits from participant q either the label λ_1 or the label λ_2: in the first case it becomes P_1, in the second case it becomes P_2. Similarly the process $Q = \mathsf{p}!\{\lambda_1; Q_1, \lambda_2; Q_2\}$ sends to participant p either the label λ_1 becoming Q_1 or the label λ_2 becoming Q_2. So the multiparty session $\mathsf{p}[\![\, P \,]\!] \parallel \mathsf{q}[\![\, Q \,]\!] \parallel \emptyset$, where $\emptyset$ is the empty queue, can reduce as follows:

$$\mathsf{p}[\![\, P \,]\!] \parallel \mathsf{q}[\![\, Q \,]\!] \parallel \emptyset \xrightarrow{\mathsf{q}\,\mathsf{p}!\lambda_1} \mathsf{p}[\![\, P \,]\!] \parallel \mathsf{q}[\![\, Q_1 \,]\!] \parallel \langle \mathsf{q}, \lambda_1, \mathsf{p} \rangle \xrightarrow{\mathsf{q}\,\mathsf{p}?\lambda_1} \mathsf{p}[\![\, P_1 \,]\!] \parallel \mathsf{q}[\![\, Q_1 \,]\!] \parallel \emptyset$$

decorating transition arrows with communications and denoting by $\langle \mathsf{q}, \lambda_1, \mathsf{p} \rangle$ the

message exchanging label λ_1 from q to p. If $P_1 = \mathsf{q}!\lambda;\mathsf{q}!\lambda;\mathsf{q}!\lambda;P$ and $Q_1 = \mathsf{p}?\lambda;Q$ we get:

$$\begin{aligned}\mathsf{p}[\![\, P_1 \,]\!] \parallel \mathsf{q}[\![\, Q_1 \,]\!] \parallel \emptyset &\xrightarrow{\mathsf{p}\,\mathsf{q}!\lambda} \mathsf{p}[\![\, \mathsf{q}!\lambda;\mathsf{q}!\lambda;P \,]\!] \parallel \mathsf{q}[\![\, Q_1 \,]\!] \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle \\ &\xrightarrow{\mathsf{p}\,\mathsf{q}!\lambda} \mathsf{p}[\![\, \mathsf{q}!\lambda;P \,]\!] \parallel \mathsf{q}[\![\, Q_1 \,]\!] \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle \cdot \langle \mathsf{p},\lambda,\mathsf{q}\rangle \\ &\xrightarrow{\mathsf{p}\,\mathsf{q}!\lambda} \mathsf{p}[\![\, P \,]\!] \parallel \mathsf{q}[\![\, Q_1 \,]\!] \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle \cdot \langle \mathsf{p},\lambda,\mathsf{q}\rangle \cdot \langle \mathsf{p},\lambda,\mathsf{q}\rangle \\ &\xrightarrow{\mathsf{p}\,\mathsf{q}?\lambda} \mathsf{p}[\![\, P \,]\!] \parallel \mathsf{q}[\![\, Q \,]\!] \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle \cdot \langle \mathsf{p},\lambda,\mathsf{q}\rangle\end{aligned}$$

Then there is a loop in which for each cycle the number of messages $\langle \mathsf{p},\lambda,\mathsf{q}\rangle$ on the queue increases by two. Assuming that $P_2 = \mathsf{q}!\lambda;P_2$ and $Q_2 = \mathsf{p}?\lambda;Q_2$, each of such messages is eventually read by q. This session can be typed by the configuration type whose queue is empty and whose global type is $\mathsf{G} = \mathsf{q}\,\mathsf{p}!\{\lambda_1;\mathsf{q}\,\mathsf{p}?\lambda_1;\mathsf{G}_1\,,\lambda_2;\mathsf{q}\,\mathsf{p}?\lambda_2;\mathsf{G}_2\}$, where $\mathsf{G}_1 = \mathsf{p}\,\mathsf{q}!\lambda;\mathsf{p}\,\mathsf{q}!\lambda;\mathsf{p}\,\mathsf{q}!\lambda;\mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G}$ and $\mathsf{G}_2 = \mathsf{p}\,\mathsf{q}!\lambda;\mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G}_2$. The type G prescribes that q put on the queue either the label λ_1 or λ_2 for p who has to read the label from the queue. Then p and q must follow the protocols described by either G_1 or G_2. It should be intuitively clear why this global type is well formed. The formalisation of this intuition is given in the remainder of the paper and requires some ingenuity.

The effectiveness of our type system is demonstrated by the implementation described in the companion paper [3]. This tool (available at [2]) implements in co-logic programming the necessary predicates for the typing of sessions.

Outline Our calculus of multiparty sessions is presented in Section 2, where the Progress property is defined. Section 3 introduces configuration types and their input/output matching, which will be completed by the algorithm given in Section 5. Global types are projected onto processes in Section 4, where we define the type system and state its properties.

2 A Core Calculus for Multiparty Sessions

Since our focus is on typing by means of global types we only consider one multiparty session instead of many interleaved multiparty sessions. This allows us to depart from the standard syntax of processes with channels [17,1] in favour of simpler processes with output and input operators and explicit participants as in [13,22,15].

We assume the following base sets: *participants* $\mathsf{p},\mathsf{q},\mathsf{r} \in \mathsf{Part}$, and *labels* $\lambda \in \mathsf{Lab}$.

Definition 1 (Processes). Processes P *are defined by:*

$$P ::=_\rho \mathbf{0} \mid \mathsf{p}!\{\lambda_i;P_i\}_{i\in I} \mid \mathsf{p}?\{\lambda_i;P_i\}_{i\in I}$$

where $I \neq \emptyset$ *and* $\lambda_j \neq \lambda_h$ *for* $j \neq h$.

The symbol $::=_\rho$, in the definition above and in other definitions, indicates that the productions should be interpreted *coinductively*. That is, they define possibly infinite processes. However, we assume such processes to be *regular*, that is, with finitely many distinct sub-processes. In this way, we only obtain processes which are solutions of finite sets of equations, see [10].

A process of shape $\mathsf{p}!\{\lambda_i; P_i\}_{i\in I}$ (*internal choice*) chooses a label in the set $\{\lambda_i \mid i \in I\}$ to be sent to p, and then behaves differently depending on the sent label. A process of shape $\mathsf{p}?\{\lambda_i; P_i\}_{i\in I}$ (*external choice*) waits for receiving one of the labels $\{\lambda_i \mid i \in I\}$ from p, and then behaves differently depending on the received label. Note that the set of indexes in choices is assumed to be non-empty, and the corresponding labels to be all different. An internal choice which is a singleton is simply written $\mathsf{p}!\lambda; P$, and $\mathsf{p}!\lambda; \mathbf{0}$ is abbreviated $\mathsf{p}!\lambda$, analogously for an external choice.

In a full-fledged calculus, labels would carry values, namely they would be of shape $\lambda(v)$. For simplicity, here we consider pure labels.

Messages are triples $\langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ denoting that participant p has sent label λ to participant q. Sent messages are stored in a queue, from which they are subsequently fetched by the receiver.

Message queues $\mathcal{M}$ are defined by:

$$\mathcal{M} ::= \emptyset \mid \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}$$

The order of messages in the queue is the order in which they will be read. Since order matters only between messages with the same sender and receiver, we always consider message queues modulo the following structural equivalence:

$$\mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{s}\rangle \cdot \mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{r}, \lambda', \mathsf{s}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}' \quad \text{if} \quad \mathsf{p} \neq \mathsf{r} \quad \text{or} \quad \mathsf{q} \neq \mathsf{s}$$

Note, in particular, that $\langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \langle \mathsf{q}, \lambda', \mathsf{p}\rangle \equiv \langle \mathsf{q}, \lambda', \mathsf{p}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$. These two equivalent queues represent a situation in which both participants p and q have sent a message to the other one, and neither of them has read the message. This situation may happen in a multiparty session with asynchronous communication.

Multiparty sessions are comprised of pairs participant/process of shape $\mathsf{p}[\![\, P \,]\!]$ composed in parallel, each with a different participant p, and a message queue.

Definition 2 (Multiparty sessions). *Multiparty sessions are defined by:*

$$\mathbb{N} \parallel \mathcal{M} \quad \textit{where} \quad \mathbb{N} ::= \mathsf{p}_1[\![\, P_1 \,]\!] \parallel \cdots \parallel \mathsf{p}_n[\![\, P_n \,]\!]$$

and $n > 0$ and $\mathsf{p}_i \neq \mathsf{p}_j$ for $i \neq j$ and $\mathcal{M}$ is a message queue.

In the following we use session as short for multiparty session.

We assume the standard structural congruence on sessions (denoted $\equiv$), that is, we consider sessions modulo permutation of components and adding/removing components of the shape $\mathsf{p}[\![\, \mathbf{0} \,]\!]$.

If $P \neq \mathbf{0}$ we write $\mathsf{p}[\![\, P \,]\!] \in \mathbb{N}$ as short for $\mathbb{N} \equiv \mathsf{p}[\![\, P \,]\!] \parallel \mathbb{N}'$ for some $\mathbb{N}'$. This abbreviation is justified by the associativity and commutativity of $\parallel$.

To define the *asynchronous operational semantics* of sessions, we use an LTS whose labels record the outputs and the inputs. To this end, *communications* (ranged over by β) are either the asynchronous emission of a label λ from participant p to participant q (notation $\mathsf{p}\,\mathsf{q}!\lambda$) or the actual reading by participant q of the label λ sent by participant p (notation $\mathsf{p}\,\mathsf{q}?\lambda$).

The LTS semantics of sessions is specified by the two rules [Send] and [Rcv] given in Figure 1. Rule [Send] allows a participant p with an internal choice (a sender) to send one of its possible labels λ_h, by adding the corresponding message to the queue. Symmetrically, rule [Rcv] allows a participant q with an external choice (a receiver) to read the first message in the queue sent to her by a given participant p, if its label λ_h is one of those she is waiting for.

$$\mathsf{p}[\![\, \mathsf{q}!\{\lambda_i; P_i\}_{i\in I} \,]\!] \parallel \mathbb{N} \parallel \mathcal{M} \xrightarrow{\mathsf{p}\,\mathsf{q}!\lambda_h} \mathsf{p}[\![\, P_h \,]\!] \parallel \mathbb{N} \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_h, \mathsf{q} \rangle \quad \text{where } h \in I \qquad [\textsc{Send}]$$

$$\mathsf{q}[\![\, \mathsf{p}?\{\lambda_j; Q_j\}_{j\in J} \,]\!] \parallel \mathbb{N} \parallel \langle \mathsf{p}, \lambda_h, \mathsf{q} \rangle \cdot \mathcal{M} \xrightarrow{\mathsf{p}\,\mathsf{q}?\lambda_h} \mathsf{q}[\![\, Q_h \,]\!] \parallel \mathbb{N} \parallel \mathcal{M} \quad \text{where } h \in J \qquad [\textsc{Rcv}]$$

Fig. 1. LTS for asynchronous sessions.

The semantic property we aim to ensure, usually called *progress* [12,9], is the conjunction of a safety property, *deadlock-freedom*, and two liveness properties: *input lock-freedom* and *orphan message-freedom*. Intuitively, a session is deadlock-free if, in every reachable state of computation, it is either terminated (i.e. of the shape $\mathsf{p}[\![\, \mathbf{0} \,]\!] \parallel \emptyset$) or it can move. It is input lock-free if every component wishing to do an input can eventually do so. Finally, it is orphan-message-free if every message stored in the queue is eventually read.

The following terminology and notational conventions are standard.

If $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_n} \mathbb{N}' \parallel \mathcal{M}'$ for some $n \geq 0$ (where by convention $\mathbb{N}' \parallel \mathcal{M}' = \mathbb{N} \parallel \mathcal{M}$ if $n = 0$), then we say that $\mathbb{N}' \parallel \mathcal{M}'$ is a *derivative* of $\mathbb{N} \parallel \mathcal{M}$. We write $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta}$ if $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathbb{N}' \parallel \mathcal{M}'$ for some $\mathbb{N}', \mathcal{M}'$.

Definition 3 (Live, terminated, deadlocked sessions). *A session $\mathbb{N} \parallel \mathcal{M}$ is said to be*
- live *if $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta}$ for some β;*
- terminated *if $\mathbb{N} \equiv \mathsf{p}[\![\, \mathbf{0} \,]\!]$ and $\mathcal{M} = \emptyset$;*
- deadlocked *if it is neither live nor terminated.*

To formalise progress (Definition 6) we introduce another transition relation on sessions, which describes their lockstep execution: at each step, all components that are able to move execute exactly one asynchronous output or input.

We define the *player of a communication* as the sender in case of output and as the receiver in case of input:

$$\mathsf{play}(\mathsf{p}\,\mathsf{q}!\lambda) = \mathsf{p} \qquad \mathsf{play}(\mathsf{p}\,\mathsf{q}?\lambda) = \mathsf{q}$$

Let Δ denote a non empty set of communications. We say that Δ is *coherent* for a session $\mathbb{N} \parallel \mathcal{M}$ if

1. for all $\beta_1, \beta_2 \in \Delta$, $\mathsf{play}(\beta_1) = \mathsf{play}(\beta_2)$ implies $\beta_1 = \beta_2$, and
2. for all $\beta \in \Delta$, $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta}$.

The *lockstep transition relation* $\mathbb{N} \parallel \mathcal{M} \overset{\Delta}{\Longrightarrow} \mathbb{N}' \parallel \mathcal{M}'$ is defined by:

$\mathbb{N} \parallel \mathcal{M} \overset{\Delta}{\Longrightarrow} \mathbb{N}' \parallel \mathcal{M}'$ if $\Delta = \{\beta_1, \ldots, \beta_n\}$ is a maximal coherent set for $\mathbb{N} \parallel \mathcal{M}$ and

$$\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_n} \mathbb{N}' \parallel \mathcal{M}'$$

The notion of derivative can be reformulated for lockstep computations as follows.

If $\mathbb{N} \parallel \mathcal{M} \overset{\Delta_1}{\Longrightarrow} \cdots \overset{\Delta_n}{\Longrightarrow} \mathbb{N}' \parallel \mathcal{M}'$ for some $n \geq 0$ (where by convention $\mathbb{N}' \parallel \mathcal{M}' = \mathbb{N} \parallel \mathcal{M}$ if $n = 0$), then we say that $\mathbb{N}' \parallel \mathcal{M}'$ is a *lockstep derivative* of $\mathbb{N} \parallel \mathcal{M}$. Clearly each lockstep derivative is a derivative, but not vice versa.

A lockstep computation is an either finite or infinite sequence of lockstep transitions, and it is *maximal* if either it is finite and cannot be extended (because the last session is not live), or it is infinite. Let γ range over lockstep computations.

Formally, a lockstep computation γ can be denoted as follows, where $x \in \mathbf{N} \cup \{\omega\}$ is the length of γ:

$$\gamma = \{\mathbb{N}_k \parallel \mathcal{M}_k \xRightarrow[k]{\Delta_k} \mathbb{N}_{k+1} \parallel \mathcal{M}_{k+1}\}_{k<x}$$

That is, γ is represented as the set of its successive lockstep transitions, where the arrow subscript k is used to indicate that the transition occurs in the k-th step of the computation. This is needed in order to distinguish equal transitions occurring in different steps. For instance, in the session $\mathbb{N} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$, where $\mathbb{N} = \mathsf{p}[\![P]\!] \parallel \mathsf{q}[\![Q]\!]$ with $P = \mathsf{q}!\lambda; P$ and $Q = \mathsf{p}?\lambda; Q$, all lockstep transitions with $k \geq 1$ are of the form

$$\mathbb{N} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \xRightarrow[k]{\{\mathsf{p}\,\mathsf{q}!\lambda, \mathsf{p}\,\mathsf{q}?\lambda\}} \mathbb{N} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$$

We can now formalise the progress property:

Definition 4 (Input-enabling session). *A session* $\mathbb{N} \parallel \mathcal{M}$ *is* input-enabling *if* $\mathsf{p}[\![\mathsf{q}?\{\lambda_i; P_i\}_{i \in I}]\!] \in \mathbb{N}$ *implies that, for all maximal*

$$\gamma = \{\mathbb{N}_k \parallel \mathcal{M}_k \xRightarrow[k]{\Delta_k} \mathbb{N}_{k+1} \parallel \mathcal{M}_{k+1}\}_{k<x}$$

with $\mathbb{N}_0 \parallel \mathcal{M}_0 = \mathbb{N} \parallel \mathcal{M}$*, there exists* $h < x$ *such that* $\mathsf{p}\,\mathsf{q}?\lambda_i \in \Delta_h$ *for some* $i \in I$*.*

Definition 5 (Queue-consuming session). *A session* $\mathbb{N} \parallel \mathcal{M}$ *is* queue-consuming *if* $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$ *implies that, for all maximal*

$$\gamma = \{\mathbb{N}_k \parallel \mathcal{M}_k \xRightarrow[k]{\Delta_k} \mathbb{N}_{k+1} \parallel \mathcal{M}_{k+1}\}_{k<x}$$

with $\mathbb{N}_0 \parallel \mathcal{M}_0 = \mathbb{N} \parallel \mathcal{M}$*, there exists* $h < x$ *such that* $\mathsf{p}\,\mathsf{q}?\lambda \in \Delta_h$*.*

Definition 6 (Progress). *A session has the progress property if:*

1. *(Deadlock-freedom) None of its lockstep derivatives is deadlocked;*
2. *(No locked inputs) All its lockstep derivatives are input-enabling;*
3. *(No orphan messages) All its lockstep derivatives are queue-consuming.*

It is easy to see that deadlock-freedom implies no locked inputs and no orphan messages for finite computations.

Example 1. Let $\mathbb{N} = \mathsf{p}[\![P]\!] \parallel \mathsf{q}[\![Q]\!] \parallel \mathsf{r}[\![R]\!]$, where $P = \mathsf{q}!\lambda; P$, $Q = \mathsf{p}?\lambda; \mathsf{r}?\lambda'; Q$ and $R = \mathsf{q}!\lambda'; R$.
The unique maximal lockstep computation of $\mathbb{N} \parallel \emptyset$ is the following:

$$\begin{aligned}
\mathbb{N} \parallel \emptyset &\xRightarrow{\{\mathsf{p}\,\mathsf{q}!\lambda, \mathsf{r}\,\mathsf{q}!\lambda'\}} && \mathbb{N} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{q} \rangle \\
&\xRightarrow{\{\mathsf{p}\,\mathsf{q}!\lambda, \mathsf{p}\,\mathsf{q}?\lambda, \mathsf{r}\,\mathsf{q}!\lambda'\}} && \mathsf{p}[\![P]\!] \parallel \mathsf{q}[\![\mathsf{r}?\lambda'; Q]\!] \parallel \mathsf{r}[\![R]\!] \parallel \langle \mathsf{r}, \lambda', \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{q} \rangle \\
&\xRightarrow{\{\mathsf{p}\,\mathsf{q}!\lambda, \mathsf{r}\,\mathsf{q}?\lambda', \mathsf{r}\,\mathsf{q}!\lambda'\}} && \mathbb{N} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{q} \rangle \\
&\ldots && \ldots
\end{aligned}$$

It is easy to check that $\mathbb{N} \parallel \emptyset$ has the progress property. Indeed, every input

communication in Q is eventually enabled, and, even though the queue grows at each step of the lockstep computation, every message in the queue is eventually read.

3 Configuration Types

As in [8], the key difference with respect to classical formulations of global types in literature is the splitting between output choices and inputs. Sessions are typed by configuration types, which are pairs of global types and queues.

Definition 7 (Global and configuration types).

1. Global types G *are defined by:*
$$\mathsf{G} ::=_\rho \mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I} \mid \mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G} \mid \mathsf{End}$$
where $I \neq \emptyset$ *and* $\mathsf{p} \neq \mathsf{q}$ *and* $\lambda_j \neq \lambda_h$ *for* $j \neq h$.
2. Configuration types *are pairs* $\mathsf{G} \parallel \mathcal{M}$*, where* G *is a global type and* $\mathcal{M}$ *is a message queue.*

As for processes, $::=_\rho$ indicates that global types are *regular*.

The global type $\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}$ specifies that player p sends a label λ_k with $k \in I$ to participant q and then the interaction described by the global type G_k takes place. The global type $\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}$ specifies that player q receives label λ from participant p and then the interaction described by the global type G takes place. A choice between different inputs is useless, since only one label can be received in each branch of an output choice.

In configuration types, as specified by the previous definition, inputs and outputs may be unrelated both between them and with the messages in the queue. In order to get a type system which guarantees progress we need to single out suitable configuration types. Moreover, we want to do this without imposing unnecessary restrictions. In this section we introduce the first two conditions we impose on configuration types: input/output matching and boundedness. A third and last condition (projectability) will be discussed in Section 4, where we will relate configurations types to session computations.

To ensure correspondence between inputs and outputs, in Figure 2 we define the input/output matching of configuration types. A configuration type $\mathsf{G} \parallel \mathcal{M}$ is *input/output matching* if we can derive $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$. The double dotted line indicates that the rules should be interpreted coinductively, i.e., we allow infinite proof trees. The intuition is that every input in the global type will find, when it is the first communication of a player, a corresponding message on the queue and every message put on the queue will be eventually read. To satisfy the first constraint rule [In] requires that the message sent from p to q with the expected label be indeed the first message from p to q on the queue. To satisfy the second constraint if the global type is End, then there should be no messages on the queue, rule [End]. If the global type is a choice of outputs, rule [Out] says that, after performing any output, the configurations (in which the queues have been augmented by the messages sent) should have matching inputs/outputs.

$$[\textsc{End}]\ \frac{}{\vdash_{\mathsf{iom}} \mathsf{End} \parallel \emptyset} \qquad [\textsc{In}]\ \frac{\vdash_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}}{\vdash_{\mathsf{iom}} \mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G} \parallel \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}} \vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M})$$

$$[\textsc{Out}]\ \frac{\vdash_{\mathsf{iom}} \mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \quad \forall i \in I}{\vdash_{\mathsf{iom}} \mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I} \parallel \mathcal{M}} \vdash_{\mathsf{read}} (\mathsf{G}_i, \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle)\ \forall i \in I$$

$$[\textsc{Empty-R}]\ \frac{}{\vdash_{\mathsf{read}} (\mathsf{G}, \emptyset)} \qquad [\textsc{Out-R}]\ \frac{\vdash_{\mathsf{read}} (\mathsf{G}_i, \mathcal{M}) \quad (\forall i \in I)}{\vdash_{\mathsf{read}} (\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}, \mathcal{M})}$$

$$[\textsc{In-R1}]\ \frac{\vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M})}{\vdash_{\mathsf{read}} (\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M})} \qquad [\textsc{In-R2}]\ \frac{\vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M})}{\vdash_{\mathsf{read}} (\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}, \mathcal{M})} \quad \mathcal{M} \not\equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'$$

Fig. 2. Input/output matching of configuration types (Coinductive version).

$$[\textsc{End-I}]\ \frac{}{\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{End} \parallel \emptyset} \qquad [\textsc{Cycle}]\ \frac{}{\mathcal{H}, (\mathsf{G}, \mathcal{M}) \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}'} \vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')$$

$$[\textsc{Out-I}]\ \frac{\mathcal{H}, (\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}, \mathcal{M}) \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \quad \forall i \in I}{\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I} \parallel \mathcal{M}}$$

$$[\textsc{In-I}]\ \frac{\mathcal{H}, (\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}) \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}}{\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G} \parallel \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}}$$

Fig. 3. Input/output matching of configuration types (Inductive version).

Moreover rules [In] and [Out] require that all messages in the queue will be eventually read. This last restriction is enforced by the auxiliary judgment $\vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M})$, which means that (each path of) G *reads all the messages in* $\mathcal{M}$. The inductive definition of this judgment is given at the bottom of Figure 2. If the queue is empty, rule [Empty-R], then the judgement holds. If G is a choice of outputs, rule [Out-R], then in each branch of the choice all the messages in the queue must be read. For an input type $\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}$, if the message at the top of the queue (considered modulo $\equiv$) is $\langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ we read it, rule [In-R1], otherwise we do not to read messages, rule [In-R2].

For example we can derive $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \emptyset$, where G is the global type discussed in the Introduction. The proof has an infinite non-regular branch showing the judgments $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \underbrace{\langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdots \langle \mathsf{p}, \lambda, \mathsf{q}\rangle}_{2n}$ for $n \geq 1$.

The coinductive formulation of input/output matching for configuration types is natural and elegant, but to get an effective type system we need an inductive formulation. Figure 3 gives such a formulation, which is parametric on the auxiliary judgment $\vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')$. This judgment will be detailed in Section 5, where the soundness proof of the inductive formulation w.r.t. the coinductive one is sketched. By $\mathcal{H}$ we denote a set of pairs $(\mathsf{G}, \mathcal{M})$ needed to detect when, starting from a configuration, we encounter a configuration with the same global type. Rules [End-I], [Out-I] and [In-I] are obtained from the corresponding rules in Figure 2 by increasing the set of circular hypotheses as usual and by

dropping the reading condition. The rule [Cycle] can by applied when the global type already appeared in the derivation and it requires a condition involving the global type and the two queues associated with it, expressed by the judgment $\vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')$. This judgement must enforce the property that all messages in $\mathcal{M}'$ will be eventually read by G and moreover that every input in G should find, when it is enabled, a corresponding message in $\mathcal{M}'$. In [8] the trivial condition $\mathcal{M} = \mathcal{M}' = \emptyset$ is used. Requiring $\mathcal{M} \equiv \mathcal{M}'$ would mean to restrict the coinductive system to regular derivations. We will discuss more significant and expressive conditions in Section 5. Note that, to obtain an actual algorithm from the rules in Figure 3, we have to force the application of [Cycle] as soon as possible, and to fail if the judgement $\vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')$ does not hold.

Unfortunately input/output matching is not enough to avoid computations where some participant remains stuck forever. To enforce lock-freedom by typing, we need the boundedness condition (Definition 9) as shown in Example 4.

Let the *players* of a global type to be its active participants. The function $\mathsf{players}$ associates to global types their sets of *players*, which are the smallest sets such that:

$$\mathsf{players}(\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}) = \{\mathsf{p}\} \cup \textstyle\bigcup_{i\in I} \mathsf{players}(\mathsf{G}_i)$$
$$\mathsf{players}(\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}) = \{\mathsf{q}\} \cup \mathsf{players}(\mathsf{G}) \qquad \mathsf{players}(\mathsf{End}) = \emptyset$$

Note that the regularity assumption on global types ensures that the set of players of a global type is finite.

Global types can be naturally seen as trees. We use ξ to denote a *path* in global type trees, i.e., a possibly infinite sequence of communications $\mathsf{p}\,\mathsf{q}!\lambda$ or $\mathsf{p}\,\mathsf{q}?\lambda$. With ξ_n we represent the n-th communication in the path ξ, where $0 \leq n < x$ and $x \in \mathbf{N} \cup \{\omega\}$ is the length of ξ. With ϵ we denote the empty sequence and with $\cdot$ the concatenation of a finite sequence with a possibly infinite sequence. The function Paths gives the set of *paths* of global types, which are the greatest sets such that:

$$\mathsf{Paths}(\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}) = \textstyle\bigcup_{i\in I}\{\mathsf{p}\,\mathsf{q}!\lambda_i \cdot \xi \mid \xi \in \mathsf{Paths}(\mathsf{G}_i)\}$$
$$\mathsf{Paths}(\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}) = \{\mathsf{p}\,\mathsf{q}?\lambda \cdot \xi \mid \xi \in \mathsf{Paths}(\mathsf{G})\} \qquad \mathsf{Paths}(\mathsf{End}) = \{\epsilon\}$$

We can now formalise the requirement that the first occurrences of players in a global type are at a bounded depth in all paths starting from the root. This is done by defining the *depth* of a player p in a global type G, $\mathsf{depth}(\mathsf{G}, \mathsf{p})$.

Definition 8 (Depth of a player). *Let* G *be a global type. For* $\xi \in \mathsf{Paths}(\mathsf{G})$ *set* $\mathsf{depth}(\xi, \mathsf{p}) = \inf\{n \mid \mathsf{play}(\xi_n) = \mathsf{p}\}$*, and define* $\mathsf{depth}(\mathsf{G}, \mathsf{p})$*, the* depth *of* p *in* G*, as follows:*

$$\mathsf{depth}(\mathsf{G}, \mathsf{p}) = \begin{cases} 1 + \sup\{\mathsf{depth}(\xi, \mathsf{p}) \mid \xi \in \mathsf{Paths}(\mathsf{G})\} & \mathsf{p} \in \mathsf{players}(\mathsf{G}) \\ 0 & \textit{otherwise} \end{cases}$$

Note that, if $\mathsf{p} \neq \mathsf{play}(\xi_n)$ for some path ξ and all $n \in \mathbf{N}$, then $\mathsf{depth}(\xi, \mathsf{p}) = \inf \emptyset = \infty$. Hence, if p is a player of a global type G, but it does not occur as a player in some path of G, then $\mathsf{depth}(\mathsf{G}, \mathsf{p}) = \infty$.

Definition 9 (Boundedness). *A global type* G *is* bounded *if* $\mathsf{depth}(\mathsf{G}', \mathsf{p})$ *is finite for all participants* $\mathsf{p} \in \mathsf{players}(\mathsf{G})$ *and all types* G' *which occur in* G*.*

Example 2. The following example shows the necessity of considering all types occurring in a global type for defining boundedness. Consider $\mathsf{G} = \mathsf{r}\,\mathsf{q}!\lambda; \mathsf{r}\,\mathsf{q}?\lambda; \mathsf{G}'$, where

$$\mathsf{G}' = \mathsf{p}\,\mathsf{q}!\{\lambda_1; \mathsf{p}\,\mathsf{q}?\lambda_1; \mathsf{q}\,\mathsf{r}!\lambda_3; \mathsf{q}\,\mathsf{r}?\lambda_3\,, \lambda_2; \mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{G}'\}$$

Then we have: $\mathsf{depth}(\mathsf{G}, \mathsf{r}) = 1 \qquad \mathsf{depth}(\mathsf{G}, \mathsf{p}) = 3 \qquad \mathsf{depth}(\mathsf{G}, \mathsf{q}) = 2$ whereas

$$\mathsf{depth}(\mathsf{G}', \mathsf{r}) = \infty \qquad \mathsf{depth}(\mathsf{G}', \mathsf{p}) = 1 \qquad \mathsf{depth}(\mathsf{G}', \mathsf{q}) = 2$$

Since global types are regular the boundedness condition is decidable.

4 Type System

Usually in type assignment systems for multiparty sessions [17,18] global types are projected onto local types and local types are assigned to processes. The simplicity of our calculus allows us to project global types directly onto processes as in [22,8].

Figure 4 gives the rules defining the judgment $\mathsf{G} \upharpoonright \mathsf{p} \mapsto P$, saying that the global type G, projected onto participant p, gives the process P. The double dotted line indicates that such rules should be interpreted coinductively. Notice that proof trees are *regular*, that is, with finitely many distinct sub-trees.

The definition uses process contexts which can have an arbitrary number of holes indexed with natural numbers, where we assume that each hole has a different index. Given a context $\mathcal{C}$ with holes indexed in J, we denote by $\mathcal{C}[P_j]_{j \in J}$ the process obtained by filling the hole indexed by j with P_j, for all $j \in J$.

Rule [Ext] states that projecting onto a participant which is not a player gives the inactive process. The following three rules describe the effect of projecting a global type which starts with the output of a label, chosen in a set, from player p to player q, and continues as G_i, if the label chosen was λ_i.

Rule [Out-Snd] projects the global type onto the sender p and, as expected, the resulting process is an output process sending the chosen label and continuing with the corresponding projections.

Rule [Out-Rcv] projects the output choice onto the receiver of the message and [Out-Ext] onto any other player of the global type. Such projections are well defined if the projections of the branches G_i, for $i \in I$, give processes which can be consistently combined to provide the resulting processes. More precisely, they have a common structure, modelled by a multi-hole context $\mathcal{C}$ with $j \in J$ holes, where the j-th holes in the projections of G_h and G_k for $h \neq k \in I$ can be filled with different processes. The processes filling the j-th holes of different branches must start with inputs from the same sender and labels identifying the branches. In this way, the processes in the j-th holes of all branches can be combined in an external choice, which is used to fill the same context in the resulting projection. If the context has no holes, i.e., it is a process, then the projections of all branches are just this process, which is also the resulting projection. For instance, if $\mathsf{G} = \mathsf{p}\,\mathsf{q}!\{\lambda_1; \mathsf{p}\,\mathsf{q}?\lambda_1; \mathsf{p}\,\mathsf{r}?\lambda\,, \lambda_2; \mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{p}\,\mathsf{r}?\lambda\}$, then $\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{p}?\lambda$ since $(\mathsf{p}\,\mathsf{q}?\lambda_1; \mathsf{p}\,\mathsf{r}?\lambda) \upharpoonright \mathsf{r} = (\mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{p}\,\mathsf{r}?\lambda) \upharpoonright \mathsf{r} = \mathsf{p}?\lambda$.

The only difference between [Out-Rcv] and [Out-Ext] is that, in rule [Out-Rcv] the sender and the labels are those of the external choice to be projected,

$$\mathcal{C} ::= [\,]_n \quad | \quad \mathsf{p}?\{\lambda_i;\mathcal{C}_i\}_{i\in I} \quad | \quad \mathsf{p}!\{\lambda_i;\mathcal{C}_i\}_{i\in I} \quad | \quad P \qquad \text{where } I \neq \emptyset,\ \lambda_j \neq \lambda_h \text{ for } j \neq h$$

$$[\textsc{Ext}]\ \frac{}{\mathsf{G}\upharpoonright \mathsf{p} \mapsto \mathbf{0}}\ \mathsf{p} \notin \mathsf{players}(\mathsf{G}) \qquad [\textsc{Out-Snd}]\ \frac{\mathsf{G}_i \upharpoonright \mathsf{p} \mapsto P_i \quad \forall i \in I}{(\mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i\}_{i\in I})\upharpoonright \mathsf{p} \mapsto \mathsf{q}!\{\lambda_i;P_i\}_{i\in I}}$$

$$[\textsc{Out-Rcv}]\ \frac{\mathsf{G}_i \upharpoonright \mathsf{q} \mapsto \mathcal{C}[\mathsf{p}?\lambda_i;P_{i,j}]_{j\in J} \quad \forall i \in I}{(\mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i\}_{i\in I})\upharpoonright \mathsf{q} \mapsto \mathcal{C}[\mathsf{p}?\{\lambda_i;P_{i,j}\}_{i\in I}]_{j\in J}}\ \mathsf{q} \in \mathsf{players}(\mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i\}_{i\in I})$$

$$[\textsc{Out-Ext}]\ \frac{\mathsf{G}_i \upharpoonright \mathsf{s} \mapsto \mathcal{C}[\mathsf{r}?\lambda'_i;R_{i,j}]_{j\in J} \quad \forall i \in I}{(\mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i\}_{i\in I})\upharpoonright \mathsf{s} \mapsto \mathcal{C}[\mathsf{r}?\{\lambda'_i;R_{i,j}\}_{i\in I}]_{j\in J}}\ \begin{array}{l}\mathsf{s} \notin \{\mathsf{p},\mathsf{q}\}\\ \mathsf{s} \in \mathsf{players}(\mathsf{G}_i)\ \forall i \in I\end{array}$$

$$[\textsc{In-Rcv}]\ \frac{\mathsf{G}\upharpoonright \mathsf{q} \mapsto P}{(\mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G})\upharpoonright \mathsf{q} \mapsto \mathsf{p}?\lambda;P} \qquad [\textsc{In-Ext}]\ \frac{\mathsf{G}\upharpoonright \mathsf{s} \mapsto P}{(\mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G})\upharpoonright \mathsf{s} \mapsto P}\ \begin{array}{l}\mathsf{s} \neq \mathsf{q}\\ \mathsf{s} \in \mathsf{players}(\mathsf{G})\end{array}$$

Fig. 4. Projection of global types.

whereas in rule [Out-Ext] they are arbitrary. Note that, if the participant r sending the message to s in rule [Out-Ext] is not the participant p who chooses the branch, then also r before behaving differently, i.e., sending distinct messages to s in different branches, must receive distinct messages exposing the various branches. So in order to behave differently in various branches a player must know in which branch of the choice she is, by receiving a label from a player who knows in which branch of the choice she is.
The last two rules describe the effect of projecting a global type starting with player q reading label λ sent by participant p, and continuing as G. In rule [In-Rcv], projecting onto player q gives the process waiting for the input λ from p, and then continuing as the projection of G. In rule [In-Ext], projecting onto any other player of G simply gives the projection of G. Note that the case where the participant is not a player in G is covered by rule [Ext].

Rules in Figure 4 define a relation, while usually the projection of a global type is expected to be a function. We can prove that for bounded global types this is the case. We conjecture that the boundedness condition could be avoided. Assuming boundedness strongly simplifies the proof and it is anyway necessary to ensure progress, as shown in Example 4.

Proposition 1. *If* G *is a bounded global type and* $\mathsf{G}\upharpoonright \mathsf{p} \mapsto P$ *and* $\mathsf{G}\upharpoonright \mathsf{p} \mapsto Q$*, then* $P = Q$*.*

Thanks to the above proposition, for a bounded global type G, we denote by $\mathsf{G}\upharpoonright \mathsf{p}$ the unique P such that $\mathsf{G}\upharpoonright \mathsf{p} \mapsto P$, thus we have $\mathsf{G}\upharpoonright \mathsf{p} = P$. Since from now on we will only deal with bounded global types we will use this notation.

In the following example we consider a global type obtained by anticipating output choices and we show the need for multihole contexts in projecting a choice of outputs onto the receiver of the communication.

Example 3. Let $\mathsf{G} = \mathsf{pq}!\{\lambda_1;\mathsf{pq}?\lambda_1;\mathsf{G}'\,,\lambda_2;\mathsf{pq}?\lambda_2;\mathsf{G}'\}$ where
$\mathsf{G}' = \mathsf{qp}!\{\lambda_3;\mathsf{qp}?\lambda_3\,,\lambda_4;\mathsf{qp}?\lambda_4;\mathsf{G}\}$

$$[\leq\text{-}\mathbf{0}]\ \frac{}{\mathbf{0}\leq\mathbf{0}} \qquad [\leq\text{-OUT}]\ \frac{P_i\leq Q_i \qquad i\in I}{\mathsf{p}!\{\lambda_i;P_i\}_{i\in I}\leq\mathsf{p}!\{\lambda_i;Q_i\}_{i\in I\cup J}} \qquad [\leq\text{-IN}]\ \frac{P_i\leq Q_i \qquad i\in I}{\mathsf{p}?\{\lambda_i;P_i\}_{i\in I\cup J}\leq\mathsf{p}?\{\lambda_i;Q_i\}_{i\in I}}$$

Fig. 5. Preorder on processes.

$$[\textsc{Type}]\ \frac{P_i\leq \mathsf{G}\upharpoonright\mathsf{p}_i \quad i\in I \qquad \mathsf{players}(\mathsf{G})\subseteq\{\mathsf{p}_i\mid i\in I\}}{\vdash \mathit{\Pi}_{i\in I}\mathsf{p}_i[\![\,P_i\,]\!]\parallel\mathcal{M}:\mathsf{G}\parallel\mathcal{M}}$$

Fig. 6. Multiparty session typing rule.

We have the projection

$$\mathsf{G}\upharpoonright\mathsf{q}=\mathsf{p}?\{\lambda_1;\mathsf{p}!\{\lambda_3\,,\lambda_4;\mathsf{G}\upharpoonright\mathsf{q}\}\,,\lambda_2;\mathsf{p}!\{\lambda_3\,,\lambda_4;\mathsf{G}\upharpoonright\mathsf{q}\}\}$$

by filling the empty context with the external choice obtained by combining the projections onto q of the branches $\mathsf{pq}?\lambda_1;\mathsf{G}'$ and $\mathsf{pq}?\lambda_2;\mathsf{G}'$.
Consider now $\mathsf{G}''=\mathsf{pq}!\{\lambda_1;\mathsf{G}_1\,,\lambda_2;\mathsf{G}_2\}$ where

$$\mathsf{G}_i=\mathsf{qp}!\{\lambda_3;\mathsf{pq}?\lambda_i;\mathsf{qp}?\lambda_3\,,\lambda_4;\mathsf{pq}?\lambda_i;\mathsf{qp}?\lambda_4;\mathsf{G}''\}\text{ for }i=1,2.$$

G'' is obtained from G by anticipating the output choice in G' before the inputs $\mathsf{pq}?\lambda_1$ and $\mathsf{pq}?\lambda_2$ in the two branches of G. To compute the projection of G'' onto q we find the context $\mathcal{C}=\mathsf{p}!\{\lambda_3;[\,]_1,\lambda_4;[\,]_2\}$ to obtain

$$\begin{aligned}\mathsf{G}''\upharpoonright\mathsf{q}&=\mathcal{C}[\ \mathsf{p}?\{\lambda_1\,,\lambda_2\}\ ]_1[\ \mathsf{p}?\{\lambda_1;\mathsf{G}''\upharpoonright\mathsf{q}\,,\lambda_2;\mathsf{G}''\upharpoonright\mathsf{q}\}\ ]_2\\&=\mathsf{p}!\{\lambda_3;\mathsf{p}?\{\lambda_1\,,\lambda_2\},\lambda_4;\mathsf{p}?\{\lambda_1;\mathsf{G}''\upharpoonright\mathsf{q}\,,\lambda_2;\mathsf{G}''\upharpoonright\mathsf{q}\}\}\end{aligned}$$

Configuration types describing well behaved sessions must be input/output matching, with global types which are bounded and projectable onto all their players.

Definition 10 (Well-formed configuration types). *We say that* $\mathsf{G}\parallel\mathcal{M}$ *is* well formed *if* $\vdash_{\mathsf{iom}}\mathsf{G}\parallel\mathcal{M}$ *holds and* G *is bounded and* $\mathsf{G}\upharpoonright\mathsf{p}$ *is defined for all* $\mathsf{p}\in\mathsf{players}(\mathsf{G})$.

In our type assignment system we permit only well-formed configuration types. Processes and projections of global types are compared using the pre-order on processes defined in Figure 5. To compare two processes they must be both internal choices or external choices or the inactive process. In the first case it is better the process doing less outputs and in the second case the one expecting more inputs and in both cases their continuations after the same label must be in the same relation. The only typing rule is given in Figure 6. It requires that for each $\mathsf{p}_i[\![\,P_i\,]\!]$ in the session the process P_i be better than the projection of the global type onto p_i for all $i\in I$. The condition $\mathsf{players}(\mathsf{G})\subseteq\{\mathsf{p}_i\mid i\in I\}$ allows participants in the session paired with process $\mathbf{0}$: this is necessary to guarantee invariance of typing with respect to structural equivalence of sessions. Notice that we can compute the projections of global types, since the regularity assumption ensures that there is only a finite number of cases to examine. Moreover the contexts required in rules [Out-Rcv] and [Out-Ext] can be determined by examining the projections of the branches in the output choices. The regularity of processes ensures that with a simple strategy based on cycle detection we can compare processes according to the preorder of Figure 5. Therefore our type

$$[\textsc{Top-Out}]\ \mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i\}_{i\in I} \parallel \mathcal{M} \xrightarrow{\mathsf{p}\,\mathsf{q}!\lambda_j} \mathsf{G}_j \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_j,\mathsf{q}\rangle \ \ j\in I$$

$$[\textsc{Top-In}]\ \mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G} \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M} \xrightarrow{\mathsf{p}\,\mathsf{q}?\lambda} \mathsf{G}\parallel\mathcal{M}$$

$$[\textsc{Inside-Out}]\ \frac{\mathsf{G}_i \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}_i' \parallel \mathcal{M}'\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \quad i\in I}{\mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i\}_{i\in I} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{p}\,\mathsf{q}!\{\lambda_i;\mathsf{G}_i'\}_{i\in I} \parallel \mathcal{M}'}\ \mathsf{p}\neq\mathsf{play}(\beta)$$

$$[\textsc{Inside-In}]\ \frac{\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'}{\mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G} \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M} \xrightarrow{\beta} \mathsf{p}\,\mathsf{q}?\lambda;\mathsf{G}' \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M}'}\ \mathsf{q}\neq\mathsf{play}(\beta)$$

Fig. 7. LTS for configuration types.

system is effective once we establish the decidability of $\vdash_{\mathsf{ok}}(\mathsf{G},\mathcal{M},\mathcal{M}')$ and that $\vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G}\parallel\mathcal{M}$ implies $\vdash_{\mathsf{iom}}\mathsf{G}\parallel\mathcal{M}$. We do not know if $\vdash_{\mathsf{iom}}\mathsf{G}\parallel\mathcal{M}$ implies $\vdash^{\mathcal{I}}_{\mathsf{iom}}\mathsf{G}\parallel\mathcal{M}$.

Example 4. Without the boundedness condition the configuration type $\mathsf{G}'\parallel\emptyset$ with G' defined in Example 2 could type the session $\mathsf{p}[\![\,P\,]\!]\parallel\mathsf{q}[\![\,Q\,]\!]\parallel\mathsf{r}[\![\,\mathsf{q}?\lambda_3\,]\!]\parallel\emptyset$, where $P=\mathsf{q}!\{\lambda_1\,,\lambda_2;P\}$ and $Q=\mathsf{p}?\{\lambda_1;\mathsf{r}!\lambda_3\,,\lambda_2;Q\}$. In this session participant r may wait forever.

In order to state the properties of our type system it is useful to introduce an LTS for configuration types as in [8], see Figure 7. The first two rules show transitions of top level output choices and inputs. The other two rules deal with transitions performed inside output choices and inputs. For these to happen the player of the performed communication must be different from the player of the enclosing output choice or input. In the case of an output choice the same communication must be done in all branches. The relation between the queues in the premises and in the conclusions of these rules mirror those of the rules for input/output matching, see Figure 2, so that starting with well-formed types the LTS produces always well-formed types. The inside rules are needed to allow transitions of configuration types to mimic those of sessions. For example if $\mathbb{N}\equiv\mathsf{p}[\![\,\mathsf{q}?\lambda\,]\!]\parallel\mathsf{q}[\![\,\mathsf{p}?\lambda'\,]\!]$, $\mathcal{M}\equiv\langle\mathsf{q},\lambda',\mathsf{p}\rangle\cdot\langle\mathsf{p},\lambda,\mathsf{q}\rangle$ and $\mathsf{G}=\mathsf{p}\,\lambda?\mathsf{q};\mathsf{q}\,\mathsf{p}?\lambda'$, then we get $\vdash\mathbb{N}\parallel\mathcal{M}:\mathsf{G}\parallel\mathcal{M}$ and $\mathbb{N}\parallel\mathcal{M}\xrightarrow{\mathsf{q}\,\mathsf{p}?\lambda'}\mathsf{p}[\![\,\mathsf{q}?\lambda\,]\!]\parallel\langle\mathsf{p},\lambda,\mathsf{q}\rangle$. We have $\mathsf{G}\parallel\mathcal{M}\xrightarrow{\mathsf{q}\,\mathsf{p}?\lambda'}\mathsf{p}\,\lambda?\mathsf{q}\parallel\langle\mathsf{p},\lambda,\mathsf{q}\rangle$ by rule [Inside-In].

As usual we start with Inversion and Canonical Form lemmas.

Lemma 1 (Inversion). *If* $\vdash\mathbb{N}\parallel\mathcal{M}:\mathsf{G}\parallel\mathcal{M}$, *then for all* $\mathsf{p}[\![\,P\,]\!]\in\mathbb{N}$ *we have* $P\leq\mathsf{G}\upharpoonright\mathsf{p}$.

Lemma 2 (Canonical Form). *If* $\vdash\mathbb{N}\parallel\mathcal{M}:\mathsf{G}\parallel\mathcal{M}$ *and* $\mathsf{p}\in\mathsf{players}(\mathsf{G})$, *then* $\mathsf{p}[\![\,P\,]\!]\in\mathbb{N}$ *and* $P\leq\mathsf{G}\upharpoonright\mathsf{p}$.

Subject Reduction ensures not only that the reduced session is typeable, but also that this session is typed by the configuration type obtained from the initial one by doing the same communication of the session transition.

Theorem 1 (Subject Reduction). *If* $\vdash \mathbb{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ *and* $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathbb{N}' \parallel \mathcal{M}'$, *then* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ *and* $\vdash \mathbb{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$.

A transition of configuration types is mimicked by a transition of sessions with a communication which can differ for the label in case of output, while it is the same in case of input. The reason is that a process with less outputs is better than a process with more outputs. Therefore a global type in which a player chooses a label can have more outputs than the process implementing the player. To state Session Fidelity we define $\mathsf{p}\,\mathsf{q}!\lambda \cong \mathsf{p}\,\mathsf{q}!\lambda'$ and $\mathsf{p}\,\mathsf{q}?\lambda \cong \mathsf{p}\,\mathsf{q}?\lambda$ for all $\mathsf{p}, \mathsf{q}, \lambda, \lambda'$.

Theorem 2 (Session Fidelity). *If* $\vdash \mathbb{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ *and* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta}$, *then* $\mathbb{N} \parallel \mathcal{M} \xrightarrow{\beta'}$ *with* $\beta \cong \beta'$.

The more interesting property of our type system is Progress. The proof of input enabling is based on the fact that all players have a finite depth in bounded global types and that this depth (when greater than 1) decreases by reducing global types at the top level. The proof of queue consuming uses input/output matching to ensure that messages in the queue will find suitable readers. In both cases we use Session Fidelity to get the transitions of sessions from the transitions of the configuration types typing them.

Theorem 3 (Progress). *If* $\vdash \mathbb{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$, *then* $\mathbb{N} \parallel \mathcal{M}$ *has the progress property.*

5 An Algorithm for Input/Output Matching

In this section we show the effectiveness of our type system by completing the inductive definition of input/output matching, see Figure 3. This amounts to specify the condition $\vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')$ of rule [Cycle]. The coinductive definition of input/output matching, see Figure 2, checks that all messages on the queue are read at each application of rules [In] and [Out], therefore rule [Cycle] needs to do a similar check on the (final) queue $\mathcal{M}'$. Note that, if a message in $\mathcal{M}$ is not in $\mathcal{M}'$, then a coinductive derivation of the judgement $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}'$ would get stuck on rule [In], i.e., the judgement would not be derivable. So we require that *all messages in* $\mathcal{M}$ *be in* $\mathcal{M}'$. Since we want to allow also derivations in which the queue between two occurrences of the same global type may increase we *allow* $\mathcal{M}' \equiv \mathcal{M} \cdot \mathcal{M}''$. In order to ensure that *the messages in the queue* $\mathcal{M}''$ do not interfere with the input/output matching of G we demand that they *can be moved after the outputs of* G. As a result the messages in $\mathcal{M}''$ will be accumulated at the end of the queue and must be read in all paths of G.

The following examples show the need for the above restrictions on the messages of $\mathcal{M}'$.

Example 5. 1. Let $\mathsf{G} = \mathsf{p}\,\mathsf{q}!\lambda_1; \mathsf{p}\,\mathsf{q}?\lambda_1; \mathsf{G}'$, where $\mathsf{G}' = \mathsf{p}\,\mathsf{q}!\{\lambda_1; \mathsf{G}\,, \lambda_2; \mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{G}\}$.

To derive $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \emptyset$ we should have $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$ since the message

$\langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$ is left on the queue in the branch of G' starting with $\mathsf{p}\,\mathsf{q}!\lambda_1$. The derivation of $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$ would require first $\vdash_{\mathsf{iom}} \mathsf{G}' \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$ and then $\vdash_{\mathsf{iom}} \mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{G} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda_2, \mathsf{q} \rangle$. But rule [In] is not applicable to $\vdash_{\mathsf{iom}} \mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{G} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda_2, \mathsf{q} \rangle$. Indeed the message $\langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$ prevents the message $\langle \mathsf{p}, \lambda_2, \mathsf{q} \rangle$ from being read by $\mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{G}$.

2. Let $\mathsf{G} = \mathsf{p}\,\mathsf{q}!\{\lambda_1; \mathsf{p}\,\mathsf{q}?\lambda_1; \mathsf{q}\,\mathsf{p}!\lambda_3; \mathsf{q}\,\mathsf{p}?\lambda_3; \mathsf{G}', \lambda_2; \mathsf{p}\,\mathsf{q}?\lambda_2; \mathsf{q}\,\mathsf{p}!\lambda_3; \mathsf{q}\,\mathsf{p}!\lambda_3; \mathsf{q}\,\mathsf{p}?\lambda_3; \mathsf{G}\}$, where
$\mathsf{G}' = \mathsf{p}\,\mathsf{q}!\lambda_4; \mathsf{p}\,\mathsf{q}?\lambda_4; \mathsf{G}'$. To get $\vdash_{\mathsf{read}} (\mathsf{G}, \langle \mathsf{q}, \lambda_3, \mathsf{p} \rangle)$ we need $\vdash_{\mathsf{read}} (\mathsf{G}', \langle \mathsf{q}, \lambda_3, \mathsf{p} \rangle)$, which does not hold. In fact the message $\langle \mathsf{q}, \lambda_3, \mathsf{p} \rangle$ may remain forever in the queue.

The first restriction is formalised by the judgement $\vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M})$ in Figure 8, to be read G *agrees with* $\mathcal{M}$. Rule [Out-C] requires that the queue $\mathcal{M}$ either does not contain a message from p to q, or I is a singleton, let $I = \{1\}$, $\mathcal{M} \equiv \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \mathcal{M}_1$ and $\mathcal{M}' \equiv \mathcal{M}_1 \cdot \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$ for some $\mathcal{M}_1$. We avoid non determinism asking to use rule [Cycle-C] whenever applicable. Considering Example 5(1) we can see that $\vdash_{\mathsf{agr}} (\mathsf{G}', \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle)$ does not hold, since $\langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda_2, \mathsf{q} \rangle \not\equiv \langle \mathsf{p}, \lambda_2, \mathsf{q} \rangle \cdot \mathcal{M}_0$ for all $\mathcal{M}_0$.
The second restriction is formalised by the judgement $\vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M})$, dubbed G *deeply reads* $\mathcal{M}$. If the global type is End, then the queue must be empty, as expected. In case the global type is a choice of outputs the messages in the queue must be consumed in all branches and likewise if it is an input. As soon as a cycle is reached we require that $\mathcal{M}$ be read with the standard judgment. The cycle is discovered by adding as premises the examined global types. With $\mathcal{G}$ we denote a set of global types. It is not difficult to show that $\vdash_{\mathsf{dread}} (\mathsf{G}, \emptyset)$ for any G. Looking at Example 5(2) we see that $\vdash_{\mathsf{read}} (\mathsf{G}', \langle \mathsf{q}, \lambda_3, \mathsf{p} \rangle)$ does not hold. Therefore $\vdash_{\mathsf{dread}} (\mathsf{G}, \langle \mathsf{q}, \lambda_3, \mathsf{p} \rangle)$ is not derivable, as expected.

To sum up $\vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')$ is defined by the following rule:

$$[\text{OK}]\ \frac{\vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M}) \quad \vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M}'') \quad \vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M}'')}{\vdash_{\mathsf{ok}} (\mathsf{G}, \mathcal{M}, \mathcal{M}')}\ \ \mathcal{M}' \equiv \mathcal{M} \cdot \mathcal{M}''$$

The correctness of this definition is stated by the main result of this section, i.e. the Soundness Theorem (Theorem 4). The proof of this theorem is based on the fact that the agreements of two queues imply the agreement of their concatenation and the same holds for deep readability.

Lemma 3. *1. If* $\vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M}_1)$ *and* $\vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M}_2)$*, then* $\vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M}_1 \cdot \mathcal{M}_2)$.
2. If $\vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M}_1)$ *and* $\vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M}_2)$*, then* $\vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M}_1 \cdot \mathcal{M}_2)$.

Theorem 4 (Soundness). *If* $\vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$*, then* $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$.

Proof (Sketch). First of all we observe that the definition of the judgement $\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$ can be equivalently expressed assuming $\mathcal{H}$ to be a sequence rather than a set, the only difference is the rule [cycle] which will have the following shape

$$[\text{Cycle}']\ \frac{\vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M}) \quad \vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M}'') \quad \vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M}'')}{\mathcal{H}_1, (\mathsf{G}, \mathcal{M}), \mathcal{H}_2 \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}'}\ \mathcal{M}' \equiv \mathcal{M} \cdot \mathcal{M}''$$

where we have used rule [OK].

$$[\textsc{End-C}]\ \frac{}{\mathcal{H} \vdash_{\mathsf{agr}} (\mathsf{End}, \mathcal{M})} \qquad [\textsc{Cycle-C}]\ \frac{}{\mathcal{H}, (\mathsf{G}, \mathcal{M}) \vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M})}$$

$$[\textsc{In-C}]\ \frac{\mathcal{H}, (\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}, \mathcal{M}) \vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M})}{\mathcal{H} \vdash_{\mathsf{agr}} (\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}, \mathcal{M})}$$

$$[\textsc{Out-C}]\ \frac{\mathcal{H}, (\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}, \mathcal{M}) \vdash_{\mathsf{agr}} (\mathsf{G}_i, \mathcal{M}')\ \ \forall i \in I}{\mathcal{H} \vdash_{\mathsf{agr}} (\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}, \mathcal{M})}\ \ \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \equiv \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \cdot \mathcal{M}'\ \forall i \in I$$

$$[\textsc{Cycle-DR}]\ \frac{\vdash_{\mathsf{read}} (\mathsf{G}, \mathcal{M})}{\mathcal{G}, \mathsf{G} \vdash_{\mathsf{dread}} (\mathsf{G}, \mathcal{M})} \quad [\textsc{End-DR}]\ \frac{}{\mathcal{G} \vdash_{\mathsf{dread}} (\mathsf{End}, \emptyset)}$$

$$[\textsc{In-DR}]\ \frac{\mathcal{G}, \mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}' \vdash_{\mathsf{dread}} (\mathsf{G}', \mathcal{M})}{\mathcal{G} \vdash_{\mathsf{dread}} (\mathsf{p}\,\mathsf{q}?\lambda; \mathsf{G}', \mathcal{M})} \qquad [\textsc{Out-DR}]\ \frac{\mathcal{G}, \mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I} \vdash_{\mathsf{dread}} (\mathsf{G}_i, \mathcal{M})\ \ (\forall i \in I)}{\mathcal{G} \vdash_{\mathsf{dread}} (\mathsf{p}\,\mathsf{q}!\{\lambda_i; \mathsf{G}_i\}_{i\in I}, \mathcal{M})}$$

Fig. 8. Agreement and deep read judgments.

We say that a sequence $\mathcal{H}$ is coherent if $\mathcal{H}_1 \vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$ holds for any decomposition $\mathcal{H} = \mathcal{H}_1, (\mathsf{G}, \mathcal{M}), \mathcal{H}_2$.

The proof is by coinduction on the definition of $\vdash_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$ (see Figure 2). To this end, we define the set $\mathcal{A}$ as follows:
$\widehat{\mathsf{G}} \parallel \widehat{\mathcal{M}} \in \mathcal{A}$ if $\widehat{\mathcal{M}} \equiv \mathcal{M}_1 \cdot \mathcal{M}_2$, $\vdash_{\mathsf{agr}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$, $\vdash_{\mathsf{dread}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$ and $\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}_1$ for some coherent $\mathcal{H}$.
From the hypothesis $\vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \mathcal{M}$, we have immediately that $\mathsf{G} \parallel \mathcal{M} \in \mathcal{A}$, since $\mathcal{M} \equiv \mathcal{M} \cdot \emptyset$, $\vdash_{\mathsf{agr}} (\mathsf{G}, \emptyset)$ and $\vdash_{\mathsf{dread}} (\mathsf{G}, \emptyset)$ always hold, and the empty sequence is coherent. Thus, to conclude the proof, we just have to show that $\mathcal{A}$ is consistent with respect to the rules in Figure 2. We prove that for all $\mathcal{H}$ coherent, $\widehat{\mathsf{G}}$, $\mathcal{M}_1$ and $\mathcal{M}_2$, if $\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}_1$, $\vdash_{\mathsf{agr}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$ and $\vdash_{\mathsf{dread}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$, then $\vdash_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}_1 \cdot \mathcal{M}_2$ is the conclusion of a rule in Figure 2, whose premises are in $\mathcal{A}$.
The proof is by induction on the length of $\mathcal{H}$, splitting cases on the last rule used to derive $\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}_1$. Cases for rules [End-I], [In-I] and [Out-I] just use the corresponding rules in Figure 2, relying on inversion lemmas for $\vdash_{\mathsf{agr}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$ and $\vdash_{\mathsf{dread}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$, on the fact that $\mathcal{H} \vdash^{\mathcal{I}}_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}_1$ implies $\vdash_{\mathsf{read}} (\widehat{\mathsf{G}}, \mathcal{M}_1)$ and that this together with $\vdash_{\mathsf{dread}} (\widehat{\mathsf{G}}, \mathcal{M}_2)$ implies $\vdash_{\mathsf{read}} (\widehat{\mathsf{G}}, \mathcal{M}_1 \cdot \mathcal{M}_2)$. We only remark that, for the case [Out-I], the commutativity requirement in the side condition of rule [Out-C] is essential. In the case of rule [Cycle′] we have $\mathcal{H} = \mathcal{H}_1, (\widehat{\mathsf{G}}, \mathcal{M}'), \mathcal{H}_2$, $\mathcal{M}_1 \equiv \mathcal{M}' \cdot \mathcal{M}''$, $\vdash_{\mathsf{agr}} (\widehat{\mathsf{G}}, \mathcal{M}'')$ and $\vdash_{\mathsf{dread}} (\widehat{\mathsf{G}}, \mathcal{M}'')$, and, since $\mathcal{H}$ is coherent, we have $\mathcal{H}_1 \vdash^{\mathcal{I}}_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}'$ and $\mathcal{H}_1$ is coherent as well. Then, the thesis follows by induction hypothesis, applied to $\mathcal{H}_1 \vdash^{\mathcal{I}}_{\mathsf{iom}} \widehat{\mathsf{G}} \parallel \mathcal{M}'$ and $\mathcal{M}'' \cdot \mathcal{M}_2$, because $\mathcal{H}_1$ is shorter than $\mathcal{H}$ and $\vdash_{\mathsf{agr}} (\mathsf{G}, \mathcal{M}'' \cdot \mathcal{M}_2)$ and $\vdash_{\mathsf{dread}} (\widehat{\mathsf{G}}, \mathcal{M}'' \cdot \mathcal{M}_2)$ hold by Lemma 3.

Considering the global type G of the Introduction to prove the judgment $\vdash^{\mathcal{I}}_{\mathsf{iom}} \mathsf{G} \parallel \emptyset$ we have to show that the messages on the queue do not interfere

with the outputs of the type G and they will be eventually read. This is done by deriving $\vdash_{\mathsf{agr}} (\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$ and $\vdash_{\mathsf{dread}} (\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$.

6 Conclusion and Future Work

In this paper we presented a new definition of well-formedness for global types specifying protocols for multiparty sessions. Using the proposal of [8], communications between participants are split into output choices and inputs and a queue is added to keep the messages sent but yet not received. The flexibility gained by this syntax must however be disciplined, since we do not want to loose the matching between outputs and inputs that is immediate in the standard formulations of global types. Moreover we require the progress property, that in this asynchronous setting means not only that participants willing to communicate will eventually do it, but also that no message will stay on the queue forever. To this aim already in [8] some well-formedness conditions were defined. We deconfine the global and configuration types by extending:
- the definition of input/output matching allowing the typing of multiparty sessions, such as our running example, in which the queue will contain an unbounded number of messages;
- the definition of projection allowing to anticipate output choices over inputs (see Example 3).

We give a coinductive definition of an input/output matching that shows explicitly the properties enforced, i.e., when the protocol gets to the point in which a participant is waiting for a message, the message is on the queue and at any point of the protocol the messages in the queue can be all consumed. The definition is not effective, so we formulate an inductive version, which we prove to be sound and still expressive enough to type our running example. An important advantage of the splitting of communications into outputs and inputs is that we can specify, at the type level, protocols in which outputs of a given participant may be anticipated before some of its inputs without the need for asynchronous subtyping. To take advantage of this feature we give a definition of projection which takes into account the tree-like shape of global types and generalises the one of [8].

As future work we plan to adapt the definition of projection to permit protocols in which a participant can send labels to different receivers in choices as in [7]. A problem left open is the completeness of the inductive definition of input/output matching with respect to the coinductive one. This is related to the proof of decidability/undecidability of the coinductive definition of input/output matching. We conjecture that completeness does not hold, but we did not find a counter-example.

Acknowledgment We are grateful to Ilaria Castellani and Elena Zucca for enlightening discussions on the subject of this paper. We thank Elena Zucca also for her careful reading of the paper. Her suggestions led to many improvements. Last but not least we are indebted to the anonymous referees for their constructive remarks.

References

1. Bettini, L., Coppo, M., D'Antoni, L., De Luca, M., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR. LNCS, vol. 5201, pp. 418–433. Springer (2008). https://doi.org/10.1007/978-3-540-85361-9_33
2. Bianchini, R., Dagnino, F.: Asynchronous-global-types-implementation. `https://github.com/RiccardoBianc/Asynchronous-global-types-implementation`
3. Bianchini, R., Dagnino, F.: Asynchronous global types in co-logic programming. In: Coordination. LNCS (2021), tool paper, to appear
4. Bravetti, M., Carbone, M., Lange, J., Yoshida, N., Zavattaro, G.: A sound algorithm for asynchronous session subtyping. In: Fokkink, W.J., van Glabbeek, R. (eds.) CONCUR. LIPIcs, vol. 140, pp. 38:1–38:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.38
5. Bravetti, M., Carbone, M., Zavattaro, G.: Undecidability of asynchronous session subtyping. Information and Computation **256**, 300–320 (2017). https://doi.org/10.1016/j.ic.2017.07.010
6. Bravetti, M., Carbone, M., Zavattaro, G.: On the boundary between decidability and undecidability of asynchronous session subtyping. Theoretical Computer Science **722**, 19–51 (2018). https://doi.org/10.1016/j.tcs.2018.02.010
7. Castellani, I., Dezani-Ciancaglini, M., Giannini, P.: Reversible sessions with flexible choices. Acta Informatica **56**(7), 553–583 (2019). https://doi.org/10.1007/s00236-019-00332-y
8. Castellani, I., Dezani-Ciancaglini, M., Giannini, P.: Global types and event structure semantics for asynchronous multiparty sessions. CoRR **abs/2102.00865** (2021), `https://arxiv.org/abs/2102.00865`
9. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global progress for dynamically interleaved multiparty sessions. Mathematical Structures in Computer Science **26**(2), 238–302 (2016). https://doi.org/10.1017/S0960129514000188
10. Courcelle, B.: Fundamental properties of infinite trees. Theoretical Computer Science **25**, 95–169 (1983). https://doi.org/10.1016/0304-3975(83)90059-2
11. Demangeon, R., Honda, K.: Nested protocols in session types. In: Koutny, M., Ulidowski, I. (eds.) CONCUR. LNCS, vol. 7454, pp. 272–286. Springer (2012). https://doi.org/10.1007/978-3-642-32940-1_20
12. Deniélou, P.M., Yoshida, N.: Dynamic multirole session types. In: Thomas Ball, M.S. (ed.) POPL. pp. 435–446. ACM Press (2011). https://doi.org/10.1145/1926385.1926435
13. Dezani-Ciancaglini, M., Ghilezan, S., Jaksic, S., Pantovic, J., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. In: Gay, S., Alglave, J. (eds.) PLACES. EPTCS, vol. 203, pp. 29 – 44. Open Publishing Association (2016). https://doi.org/10.4204/EPTCS.203.3
14. Gay, S., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica **42**(2/3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z
15. Ghilezan, S., Jaksic, S., Pantovic, J., Scalas, A., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. Journal of Logic and Algebraic Methods in Programming **104**, 127–173 (2019). https://doi.org/10.1016/j.jlamp.2018.12.002
16. Ghilezan, S., Pantović, J., Prokić, I., Scalas, A., Yoshida, N.: Precise subtyping for asynchronous multiparty sessions. Proceedings of the ACM on Programming Languages **5**(POPL), 1–28 (2021). https://doi.org/10.1145/3434297

17. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Necula, G.C., Wadler, P. (eds.) POPL. pp. 273–284. ACM Press (2008). https://doi.org/10.1145/1328897.1328472
18. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. Journal of ACM **63**(1), 9:1–9:67 (2016). https://doi.org/10.1145/2827695
19. Lange, J., Yoshida, N.: On the undecidability of asynchronous session subtyping. In: Esparza, J., Murawski, A.S. (eds.) FOSSACS. LNCS, vol. 10203, pp. 441–457 (2017). https://doi.org/10.1007/978-3-662-54458-7_26
20. Liskov, B., Wing, J.M.: A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems **16**(6), 1811–1841 (1994). https://doi.org/10.1145/197320.197383
21. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP. LNCS, vol. 5502, pp. 316–332. Springer (2009). https://doi.org/10.1007/978-3-642-00590-9_23
22. Severi, P., Dezani-Ciancaglini, M.: Observational Equivalence for Multiparty Sessions. Fundamenta Informaticae **167**, 267–305 (2019). https://doi.org/10.1007/s00236-019-00332-y