On Global Types and Multi-Party Sessions

Giuseppe Castagna[1], Mariangiola Dezani-Ciancaglini[2], and Luca Padovani[2]

[1] CNRS, Université Paris Diderot – Paris 7
[2] Dipartimento d'Informatica, Università degli Studi di Torino

Abstract. We present a new, streamlined language of global types equipped with a trace-based semantics and whose features and restrictions are semantically justified. The multi-party sessions obtained projecting our global types enjoy a liveness property in addition to the traditional progress and are shown to be sound and complete with respect to the set of traces of the originating global type. Our notion of completeness is less demanding than the classical ones, allowing a multi-party session to leave out redundant traces from an underspecified global type.

1 Introduction

Relating the global specification of a system of communicating entities with an implementation (or description) of the single entities is a great classic in many different areas of computer science. The recent development of *session-based computing* has renewed the interest in this problem. In this work we attack it from the behavioral type and process algebra perspectives and briefly compare the approaches used in other areas.

A (multi-party) session is a place of interaction for a restricted number of participants that communicate messages. The interaction may involve the exchange of arbitrary sequences of messages of possibly different types. Sessions are restricted to a (usually fixed) number of participants, which makes them suitable as a structuring construct for systems of communicating entities. In this work we define a language to describe the interactions that may take place among the participants implementing a given session. In particular, we aim at a definition based on few "essential" assumptions that should not depend on the way each single participant is implemented. To give an example, a bargaining protocol that includes two participants, "seller" and "buyer", can be informally described as follows:

> *Seller sends buyer a price and a description of the product; then buyer sends seller acceptance or it quits the conversation.*

If we abstract from the value of the price and the content of the description sent by the seller, this simple protocol describes just two possible executions, according to whether the buyer accepts or quits. If we consider that the price and the description are in distinct messages then the possible executions become four, according to which communication happens first. While the protocol above describes a finite set of possible interactions, it can be easily modified to accommodate infinitely many possible executions, as well as additional conversations: for instance the protocol may allow "buyer" to answer "seller" with a counteroffer, or it may interleave this bargaining with an independent bargaining with a second seller.

All essential features of protocols are in the example above, which connects some basic communication actions by the flow control points we underlined in the text. More generally, a protocol is a possibly infinite set of finite sequences of interactions between a fixed set of participants. We argue that the set of sequences that characterizes a protocol—and thus the protocol itself—can be described by a language with one form of atomic actions and three composition operators.

Atomic actions. The only atomic action is the interaction, which consists of one (or more) sender(s) (*eg*, "seller sends"), the content of the communication (*eg*, "a price", "a description", "acceptance"), and one (or more) receiver(s) (*eg*, "buyer").

Compound actions. Actions and, more generally, protocols can be composed in three different ways. First, two protocols can be composed sequentially (*eg*, "Seller sends buyer a price…; ***then*** buyer sends…") thus imposing a precise order between the actions of the composed protocols. Alternatively, two protocols can be composed unconstrainedly, without specifying any order (*eg*, "Seller sends a price ***and*** (sends) a description") thus specifying that any order between the actions of the composed protocols is acceptable. Finally, protocols can be composed in alternative (*eg*, "buyer sends acceptance ***or*** it quits"), thus offering a choice between two or more protocols only one of which may be chosen.

More formally, we use $\mathtt{p} \xrightarrow{a} \mathtt{q}$ to state that participant `p` sends participant `q` a message whose content is described by a, and we use ";", "$\wedge$", and "$\vee$" to denote sequential, unconstrained, and alternative composition, respectively. Our initial example can thus be rewritten as follows:

$$\begin{array}{l}(\mathtt{seller} \xrightarrow{\mathit{descr}} \mathtt{buyer} \wedge \mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer});\\ (\mathtt{buyer} \xrightarrow{\mathit{accept}} \mathtt{seller} \vee \mathtt{buyer} \xrightarrow{\mathit{quit}} \mathtt{seller})\end{array} \tag{1}$$

The first two actions are composed unconstrainedly, and they are to be followed by one (and only one) action of the alternative before ending. Interactions of unlimited length can be defined by resorting to a Kleene star notation. For example to extend the previous protocol so that the buyer may send a counter-offer and wait for a new price, it suffices to add a Kleene-starred line:

$$\begin{array}{l}(\mathtt{seller} \xrightarrow{\mathit{descr}} \mathtt{buyer} \wedge \mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer});\\ (\mathtt{buyer} \xrightarrow{\mathit{offer}} \mathtt{seller};\mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer})\mathtt{*};\\ (\mathtt{buyer} \xrightarrow{\mathit{accept}} \mathtt{seller} \vee \mathtt{buyer} \xrightarrow{\mathit{quit}} \mathtt{seller})\end{array} \tag{2}$$

The description above states that, after having received (in no particular order) the price and the description from the seller, the buyer can initiate a loop of zero or more interactions and then decide whether to accept or quit.

Whenever there is an alternative there must be a participant that decides which path to take. In both examples it is `buyer` that makes the choice by deciding whether to send *accept* or *quit*. The presence of a participant that decides holds true in loops too, since it is again `buyer` that decides whether to enter or repeat the iteration (by sending *offer*) or to exit it (by sending *accept* or *quit*). We will later show that absence of such decision-makers gives protocols impossible to implement. This last point critically

depends on the main hypothesis we assume about the systems we are going to the describe, that is the absence of *covert channels*. One the one hand, we try to develop a protocol description language that is as generic as possible; on the other hand, we limit the power of the system and require all communications between different participants to be explicitly stated. In doing so we bar out protocols whose implementation essentially relies on the presence of secret/invisible communications between participants: a protocol description must contain all and only the interactions used to implement it.

Protocol specifications such as the ones presented above are usually called *global types* to emphasize the fact that they describe the acceptable behaviours of a system from a global point of view. In an actual implementation of the system, though, each participant autonomously implements a different part of the protocol. To understand whether an implementation satisfies a specification, one has to consider the set of all possible sequences of synchronizations performed by the implementation and check whether this set satisfies five basic properties:

1. *Sequentiality:* if the specification states that two interactions must occur in a given order (by separating them by a ";"), then this order must be respected by all possible executions. So an implementation in which `buyer` may send *accept* before receiving *price* violates the specification.
2. *Alternativeness:* if the specification states that two interactions are alternative, then every execution must exhibit one and only one of these two actions. So an implementation in which `buyer` emits both *accept* and *quit* (or none of them) in the same execution violates the specification.
3. *Shuffling:* if the specification composes two sequences of interactions in an unconstrained way, then all executions must exhibit some shuffling (in the sense used in combinatorics and algebra) of these sequences. So an implementation in which `seller` emits *price* without emitting *descr* violates the specification.
4. *Fitness:* if the implementation exhibits a sequence of interactions, then this sequence is expected by (*ie*, it fits) the specification. So any implementation in which `seller` sends `buyer` any message other than *price* and *descr* violates the specification.
5. *Exhaustivity:* if some sequence of interactions is described by the specification, then there must exist at least an execution of the implementation that exhibits these actions (possibly in a different order). So an implementation in which no execution of `buyer` emits *accept* violates the specification.

Checking whether an implemented system satisfies a specification by comparing the actual and the expected sequences of interactions is non-trivial, for systems are usually infinite-state. Therefore, on the lines of [9, 16], we proceed the other way round: we extract from a global type the local specification (usually dubbed *session type* [20, 15]) of each participant in the system and we type-check the implementation of each participant against the corresponding session type. If the projection operation is done properly and the global specification satisfies some well-formedness conditions, then we are guaranteed that the implementation satifies the specification. As an example, the global type (1) can be projected to the following behaviors for `buyer` and `seller`:

$$\texttt{seller} \mapsto \texttt{buyer}!\mathit{descr}.\texttt{buyer}!\mathit{price}.(\texttt{buyer}?\mathit{accept} + \texttt{buyer}?\mathit{quit})$$
$$\texttt{buyer} \mapsto \texttt{seller}?\mathit{descr}.\texttt{seller}?\mathit{price}.(\texttt{seller}!\mathit{accept} \oplus \texttt{seller}!\mathit{quit})$$

or to

$$\begin{array}{l}\texttt{seller} \mapsto \texttt{buyer}!\mathit{price}.\texttt{buyer}!\mathit{descr}.(\texttt{buyer}?\mathit{accept} + \texttt{buyer}?\mathit{quit}) \\ \texttt{buyer} \mapsto \texttt{seller}?\mathit{price}.\texttt{seller}?\mathit{descr}.(\texttt{seller}!\mathit{accept} \oplus \texttt{seller}!\mathit{quit})\end{array}$$

where $\texttt{p}!a$ denotes the output of a message a to participant $\texttt{p}$, $\texttt{p}?a$ the input of a message a from participant $\texttt{p}$, $\texttt{p}?a.T + \texttt{q}?b.S$ the (external) choice to continue as T or S according to whether a is received from $\texttt{p}$ or b is received from $\texttt{q}$ and, finally, $\texttt{p}!a.T \oplus \texttt{q}!b.S$ denotes the (internal) choice between sending a to $\texttt{p}$ and continue as T or sending S to $\texttt{q}$ and continue as T. We will call *session environments* the mappings from participants to their session types. It is easy to see that any two processes implementing `buyer` and `seller` will satisfy the global type (1) if *and only if* their visible behaviors matches one of the two session environments above (these session environments thus represent some sort of minimal typings of processes implementing `buyer` and `seller`). In particular, both the above session environments are fitting and exhaustive with respect to the specification since they precisely describe what the single participants are expected and bound to do.

We conclude this introduction by observing that there are global types that are intrinsically flawed, in the sense that they do not admit any implementation (without covert channels) satisfying them. We classify flawed global types in three categories, according to the seriousness of their flaws.

[*No sequentiality*] The mildest flaws are those in which the global type specifies some sequentiality constraint between independent interactions, such as in $(\texttt{p} \xrightarrow{a} \texttt{q};\texttt{r} \xrightarrow{b} \texttt{s})$, since it is impossible to implement $\texttt{r}$ so that it sends b only after that $\texttt{q}$ has received a (unless this reception is notified on a covert channel, of course). Therefore, it is possible to find exhaustive (but not fitting) implementations that include some unexepected sequences which differ from the expected ones only by a permutation of interactions done by independent participants. The specification at issue can be easily patched by replacing some ";" by "$\wedge$".

[*No knowledge for choice*] A more severe kind of flaw occurs when the global type requires some participant to behave in different ways in accordance with some choice it is unaware of. For instance, in the global type

$$(\texttt{p} \xrightarrow{a} \texttt{q};\texttt{q} \xrightarrow{a} \texttt{r};\texttt{r} \xrightarrow{a} \texttt{p}) \quad \vee \quad (\texttt{p} \xrightarrow{b} \texttt{q};\texttt{q} \xrightarrow{a} \texttt{r};\texttt{r} \xrightarrow{b} \texttt{p})$$

participant $\texttt{p}$ chooses the branch to execute, but after having received a from $\texttt{q}$ participant $\texttt{r}$ has no way to know whether it has to send a or b. Also in this case it is possible to find exhaustive (but not fitting) implementations of the global type where the participant $\texttt{r}$ chooses to send a or b independently of what $\texttt{p}$ decided to do.

[*No knowledge, no choice*] In the worst case it is not possible to find an exhaustive implementation of the global type, for it specifies some combination of incompatible behaviors, such as performing and input or an output in mutual exclusion. This typically is the case of the absence of a decision-maker in the alternatives such as in

$$\texttt{p} \xrightarrow{a} \texttt{q} \vee \texttt{q} \xrightarrow{b} \texttt{p}$$

where each participant is required to choose between sending or receiving. There seems to be no obvious way to patch these global types without reconsidering also the intended semantics.

Contributions and outline. A first contribution of this work is to introduce a streamlined language of global specifications—that we dub *global types* (Section 2)—and to relate it with *session environments* (Section 3), that is, with sets of independent, sequential, asynchronous *session types* to be type-checked against implementations. Global types are just regular expressions augmented with a shuffling operator and their semantics is defined in terms of finite sequences of interactions. The second contribution, which is a consequence of the chosen semantics of global types, is to ensure that every implementation of a global type preserves the possibility to reach a state where *every* participant has successfully terminated.

In Section 4 we study the relationship between global types and sessions. We do so by defining a projection operation that extracts from a global type *all* the (sets of) possible session types of its participants. This projection is useful not only to check the implementability of a global description (and, incidentally, to formally define the notions of errors informally described so far) but, above all, to relate in a compositional and modular way a global type with the sets of distributed processes that implement it. We also identify a class of well-formed global types whose projections need no covert channels. Interestingly, we are able to effectively characterize well-formed global types solely in terms of their semantics.

In Section 5 we present a projection algorithm for global types. The effective generation of all possible projections is impossible. The reason is that the projectability of a global type may rely on some global knowledge that is no longer available when working at the level of single session types: while in a global approach we can, say, add to some participant new synchronization offerts that, thanks to our global knowledge, we know will never be used, this cannot be done when working at the level of single participant. Therefore in order to work at the projected level we will use stronger assumptions that ensure a sound implementation in all possible contexts.

In Section 6 we show some limitations deriving from the use of the Kleene star operator in our language of global types, and we present one possible way to circumvent them. We conclude with an extended discussion on related work (Section 7) and a few final considerations (Section 8).

Proofs, more examples and an extended survey of related work were omitted and can be found in the long version available on the authors' home pages.

2 Global Types

In this section we define syntax and semantics of global types. We assume a set $\mathscr{A}$ of *message types*, ranged over by a, b, ..., and a set Π of *roles*, ranged over by $\mathtt{p}$, $\mathtt{q}$, ..., which we use to uniquely identify the participants of a session; we let π, ... range over non-empty, finite sets of roles.

Global types, ranged over by $\mathscr{G}$, are the terms generated by the grammar in Table 1. Their syntax was already explained in Section 1 except for two novelties. First, we include a skip atom which denotes the unit of sequential composition (it plays the same role as the empty word in regular expressions). This is useful, for instance, to express optional interactions. Thus, if in our example we want the buyer to do at most one

Table 1. Syntax of global types.

$\mathscr{G}$	::=		**Global Type**		
		skip	(skip)	$\mid \pi \xrightarrow{a} \mathtt{p}$	(interaction)
	$\mid$	$\mathscr{G};\mathscr{G}$	(sequence)	$\mid \mathscr{G} \wedge \mathscr{G}$	(both)
	$\mid$	$\mathscr{G} \vee \mathscr{G}$	(either)	$\mid \mathscr{G}^*$	(star)

counteroffer instead of several ones, we just replace the starred line in (2) by

$$(\mathtt{buyer} \xrightarrow{\mathit{offer}} \mathtt{seller};\mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer}) \vee \mathsf{skip}$$

which, using syntactic sugar of regular expressions, might be rendered as

$$(\mathtt{buyer} \xrightarrow{\mathit{offer}} \mathtt{seller};\mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer})?$$

Second, we generalize interactions by allowing a finite set of roles on the l.h.s. of interactions. Therefore, $\pi \xrightarrow{a} \mathtt{p}$ denotes the fact that (the participant identified by) $\mathtt{p}$ waits for an a message from all of the participants whose tags are in π. We will write $\mathtt{p} \xrightarrow{a} \mathtt{q}$ as a shorthand for $\{\mathtt{p}\} \xrightarrow{a} \mathtt{q}$.

To be as general as possible, one could also consider interactions of the form $\pi \xrightarrow{a} \pi'$, which could be used to specify broadcast communications between participants. It turns out that this further generalization is superfluous in our setting since the interaction $\pi \xrightarrow{a} \{\mathtt{p}_i\}_{i\in I}$ can be encoded as $\bigwedge_{i\in I}(\pi \xrightarrow{a} \mathtt{p}_i)$. The encoding is made possible by the fact that communication is asynchronous and output actions are not blocking (see Section 3), therefore the order in which the participants in π send a to the $\mathtt{p}_i$'s is irrelevant. Vice versa, we will see that allowing sets of multiple senders enriches the expressiveness of the calculus, because $\pi \xrightarrow{a} \mathtt{p}$ can be used to *join* different activities involving the participants in $\pi \cup \{\mathtt{p}\}$, while $\bigwedge_{i\in I}(\mathtt{p}_i \xrightarrow{a} \mathtt{q})$ represents *fork* of parallel activities. For example, we can represent two buyers waiting for both the price from a seller and the mortgage from a bank before deciding the purchase:

$$\begin{array}{l}(\mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer1} \wedge \mathtt{bank} \xrightarrow{\mathit{mortgage}} \mathtt{buyer2});\\ (\{\mathtt{buyer1},\mathtt{buyer2}\} \xrightarrow{\mathit{accept}} \mathtt{seller} \wedge \{\mathtt{buyer1},\mathtt{buyer2}\} \xrightarrow{\mathit{accept}} \mathtt{bank})\end{array} \tag{3}$$

In general we will assume $\mathtt{p} \notin \pi$ for every interaction $\pi \xrightarrow{a} \mathtt{p}$ occurring in a global type, that is, we forbid participants to send messages to themselves. For the sake of readability we adopt the following precedence of global type operators $\longrightarrow \ ^* \ ; \ \wedge \ \vee$.

Global types denote languages of legal interactions that can occur in a multi-party session. These languages are defined over the alphabet of interactions

$$\Sigma = \{\pi \xrightarrow{a} \mathtt{p} \mid \pi \subset_{\mathit{fin}} \Pi, \mathtt{p} \in \Pi, \mathtt{p} \notin \pi, a \in \mathscr{A}\}$$

and we use α as short for $\pi \xrightarrow{a} \mathtt{p}$ when possible; we use φ, ψ, ... to range over strings in Σ^* and ε to denote the empty string, as usual. To improve readability we will sometimes use ";" to denote string concatenation.

In order to express the language of a global type having the shape $\mathscr{G}_1 \wedge \mathscr{G}_2$ we need a standard shuffling operator over languages, which can be defined as follows:

Definition 2.1 (shuffling). *The* shuffle *of L_1 and L_2, denoted by $L_1 \sqcup\!\sqcup L_2$, is the language defined by:* $L_1 \sqcup\!\sqcup L_2 \stackrel{def}{=} \{\varphi_1\psi_1 \cdots \varphi_n\psi_n \mid \varphi_1 \cdots \varphi_n \in L_1 \wedge \psi_1 \cdots \psi_n \in L_2\}$.

Observe that, in $L_1 \sqcup\!\sqcup L_2$, the order of interactions coming from one language is preserved, but these interactions can be interspersed with other interactions coming from the other language.

Definition 2.2 (traces of global types). *The set of* traces *of a global type is inductively defined by the following equations:*

$$\begin{array}{lll} \mathrm{tr}(\mathsf{skip}) = \{\varepsilon\} & \mathrm{tr}(\mathscr{G}_1;\mathscr{G}_2) = \mathrm{tr}(\mathscr{G}_1)\mathrm{tr}(\mathscr{G}_2) & \mathrm{tr}(\mathscr{G}_1 \vee \mathscr{G}_2) = \mathrm{tr}(\mathscr{G}_1) \cup \mathrm{tr}(\mathscr{G}_2) \\ \mathrm{tr}(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} & \mathrm{tr}(\mathscr{G}^*) = (\mathrm{tr}(\mathscr{G}))^\star & \mathrm{tr}(\mathscr{G}_1 \wedge \mathscr{G}_2) = \mathrm{tr}(\mathscr{G}_1) \sqcup\!\sqcup \mathrm{tr}(\mathscr{G}_2) \end{array}$$

where juxtaposition denotes concatenation and $(\cdot)^\star$ is the usual Kleene closure of regular languages.

Before we move on, it is worth noting that $\mathrm{tr}(\mathscr{G})$ is a regular language (recall that regular languages are closed under shuffling). Since a regular language is made of *finite strings*, we are implicitly making the assumption that a global type specifies interactions of finite length. This means that we are considering interactions of arbitraty length, but such that the termination of all the involved participants is always within reach. This is a subtle, yet radical change from other multi-party session theories, where infinite interactions are considered legal.

3 Multi-Party Sessions

We devote this section to the formal definition of the behavior of the participants of a multiparty session.

3.1 Session Types

We need an infinite set of recursion variables ranged over by $X, \ldots$. Pre-session types, ranged over by $T, S, \ldots$, are the terms generated by the grammar in Table 2 such that all recursion variables are guarded by at least one input or output prefix. We consider pre-session types modulo associativity, commutativity, and idempotence of internal and external choices, fold/unfold of recursions and the equalities

$$\pi!a.T \oplus \pi!a.S = \pi!a.(T \oplus S) \qquad \pi?a.T + \pi?a.S = \pi?a.(T + S)$$

Pre-session types are behavioral descriptions of the participants of a multiparty session. Informally, end describes a successfully terminated party that no longer participates to a session. The pre-session type $\mathtt{p}!a.T$ describes a participant that sends an a message to participant $\mathtt{p}$ and afterwards behaves according to T; the pre-session type $\pi?a.T$ describes a participant that waits for an a message from all the participants in π

Table 2. Syntax of pre-session types.

$T ::=$	**Pre-Session Type**		
end	(termination)	$\mid X$	(variable)
$\mid \mathtt{p}!a.T$	(output)	$\mid \pi?a.T$	(input)
$\mid T \oplus T$	(internal choice)	$\mid T + T$	(external choice)
$\mid \mathtt{rec}\, X.T$	(recursion)		

and, upon arrival of the message, behaves according to T; we will usually abbreviate $\{\mathtt{p}\}?a.T$ with $\mathtt{p}?a.T$. Behaviors can be combined by means of behavioral choices $\oplus$ and $+$: $T \oplus S$ describes a participant that internally decides whether to behave according to T or S; $T + S$ describes a participant that offers to the other participants two possible behaviors, T and S. The choice as to which behavior is taken depends on the messages sent by the other participant. In the following, we sometimes use n-ary versions of internal and external choices and write, for example, $\bigoplus_{i=1}^{n} \mathtt{p}_i!a_i.T_i$ for $\mathtt{p}_1!a_1.T_1 \oplus \cdots \oplus \mathtt{p}_n!a_n.T_n$ and $\sum_{i=1}^{n} \pi_i?a_i.T_i$ for $\pi_1?a_1.T_1 + \cdots + \pi_n?a_n.T_n$. As usual, terms X and $\mathtt{rec}\, X.T$ are used for describing recursive behaviors. As an example, $\mathtt{rec}\, X.(\mathtt{p}!a.X \oplus \mathtt{p}!b.\mathsf{end})$ describes a participant that sends an arbitrary number of a messages to $\mathtt{p}$ and terminates by sending a b message; dually, $\mathtt{rec}\, X.(\mathtt{p}?a.X + \mathtt{p}?b.\mathsf{end})$ describes a participant that is capable of receiving an arbitrary number of a messages from $\mathtt{p}$ and terminates as soon a b message is received.

Session types are the pre-session types where internal choices are used to combine outputs, external choices are used to combine inputs, and the continuation after every prefix is uniquely determined by the prefix. Formally:

Definition 3.1 (session types). *A pre-session type T is a* session type *if either:*

- $T = \mathsf{end}$, *or*
- $T = \bigoplus_{i \in I} \mathtt{p}_i!a_i.T_i$ *and* $\forall i, j \in I$ *we have that* $\mathtt{p}_i!a_i = \mathtt{p}_j!a_j$ *implies* $i = j$ *and each* T_i *is a session type, or*
- $T = \sum_{i \in I} \pi_i?a_i.T_i$ *and* $\forall i, j \in I$ *we have that* $\pi_i \subseteq \pi_j$ *and* $a_i = a_j$ *imply* $i = j$ *and each* T_i *is a session type.*

3.2 Session Environments

A session environment is defined as the set of the session types of its participants, where each participant is uniquely identified by a role. Formally:

Definition 3.2 (session environment). *A* session environment *(briefly,* session*) is a finite map* $\{\mathtt{p}_i : T_i\}_{i \in I}$.

In what follows we use Δ to range over sessions and we write $\Delta \uplus \Delta'$ to denote the union of sessions, when their domains are disjoint.

To describe the operational semantics of a session we model an asynchronous form of communication where the messages sent by the participants of the session are stored within a *buffer* associated with the session. Each message has the form $\mathtt{p} \xrightarrow{a} \mathtt{q}$ describing the sender $\mathtt{p}$, the receiver $\mathtt{q}$, and the type a of message. Buffers, ranged over by $\mathbb{B}$,

$\ldots$, are finite sequences $\mathtt{p}_1 \xrightarrow{a_1} \mathtt{q}_1 :: \cdots :: \mathtt{p}_n \xrightarrow{a_n} \mathtt{q}_n$ of messages considered modulo the least congruence $\simeq$ over buffers such that:

$$\mathtt{p} \xrightarrow{a} \mathtt{q} :: \mathtt{p}' \xrightarrow{b} \mathtt{q}' \simeq \mathtt{p}' \xrightarrow{b} \mathtt{q}' :: \mathtt{p} \xrightarrow{a} \mathtt{q} \qquad \text{for } \mathtt{p} \neq \mathtt{p}' \text{ or } \mathtt{q} \neq \mathtt{q}'$$

that is, we care about the order of messages in the buffer only when these have both the same sender and the same receiver. In practice, this corresponds to a form of communication where each pair of participants of a multiparty session is connected by a distinct FIFO buffer.

There are two possible reductions of a session:

$$\mathbb{B} \fatsemi \{\mathtt{p} : \textstyle\bigoplus_{i\in I} \mathtt{p}_i!a_i.T_i\} \uplus \Delta \quad \longrightarrow \quad (\mathtt{p} \xrightarrow{a_k} \mathtt{p}_k) :: \mathbb{B} \fatsemi \{\mathtt{p} : T_k\} \uplus \Delta \qquad (k \in I)$$

$$\mathbb{B} :: (\mathtt{p}_i \xrightarrow{a} \mathtt{p})_{i\in I} \fatsemi \{\mathtt{p} : \textstyle\sum_{j\in J} \pi_j?a_j.T_j\} \uplus \Delta \xrightarrow{\pi_k \xrightarrow{a} \mathtt{p}} \mathbb{B} \fatsemi \{\mathtt{p} : T_k\} \uplus \Delta \qquad \begin{pmatrix} k\in J & a_k = a \\ \pi_k = \{\mathtt{p}_i \mid i \in I\} \end{pmatrix}$$

The first rule describes the effect of an output operation performed by participant $\mathtt{p}$, which stores the message $\mathtt{p} \xrightarrow{a_k} \mathtt{p}_k$ in the buffer and leaves participant $\mathtt{p}$ with a residual session type T_k corresponding to the message that has been sent. The second rule describes the effect of an input operation performed by participant $\mathtt{p}$. If the buffer contains enough messages of type a coming from all the participants in π_k, those messages are removed from the buffer and the receiver continues as described in T_k. In this rule we decorate the reduction relation with $\pi_k \xrightarrow{a} \mathtt{p}$ that describes the occurred interaction (as we have already remarked, we take the point of view that an interaction is completed when messages are received). This decoration will allow us to relate the behavior of an implemented session with the traces of a global type (see Definition 2.2). We adopt some conventional notation: we write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; we write $\overset{\alpha}{\Longrightarrow}$ for the composition $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$ and $\overset{\alpha_1\cdots\alpha_n}{\Longrightarrow}$ for the composition $\overset{\alpha_1}{\Longrightarrow}\cdots\overset{\alpha_n}{\Longrightarrow}$.

We can now formally characterize the "correct sessions" as those in which, no matter how they reduce, it is always possible to reach a state where all of the participants are successfully terminated and the buffer has been emptied.

Definition 3.3 (live session). *We say that* Δ *is a* live session *if* $\varepsilon \fatsemi \Delta \overset{\varphi}{\Longrightarrow} \mathbb{B} \fatsemi \Delta'$ *implies* $\mathbb{B} \fatsemi \Delta' \overset{\psi}{\Longrightarrow} \varepsilon \fatsemi \{\mathtt{p}_i : \mathsf{end}\}_{i\in I}$ *for some* ψ.

We adopt the term "live session" to emphasize the fact that Definition 3.3 states a *liveness property*: every finite computation $\varepsilon, \Delta \overset{\varphi}{\Longrightarrow} \mathbb{B} \fatsemi \Delta'$ can always be extended to a successful computation $\varepsilon \fatsemi \Delta \overset{\varphi}{\Longrightarrow} \mathbb{B} \fatsemi \Delta' \overset{\psi}{\Longrightarrow} \varepsilon \fatsemi \{\mathtt{p}_i : \mathsf{end}\}_{i\in I}$. This is stronger than the progress property enforced by other multiparty session type theories, where it is only required that a session must never get stuck (but it is possible that some participants starve for messages that are never sent). As an example, the session

$$\Delta_1 = \{\mathtt{p} : \mathtt{rec}\, X.(\mathtt{q}!a.X \oplus \mathtt{q}!b.\mathsf{end}) \;,\; \mathtt{q} : \mathtt{rec}\, Y.(\mathtt{p}?a.Y + \mathtt{p}?b.\mathsf{end})\}$$

is alive because, no matter how many a messages $\mathtt{p}$ sends, $\mathtt{q}$ can receive all of them and, if $\mathtt{p}$ chooses to send a b message, the interaction terminates successfully for both $\mathtt{p}$ and $\mathtt{q}$. This example also shows that, despite the fact that session types describe finite-state

processes, the session Δ_1 is not finite-state, in the sense that the set of configurations $\{(\mathbb{B} \fatsemi \Delta') \mid \exists \varphi, \mathbb{B}, \Delta' : \varepsilon \fatsemi \Delta_1 \stackrel{\varphi}{\Longrightarrow} \mathbb{B} \fatsemi \Delta'\}$ is infinite. This happens because there is no bound on the size of the buffer and an arbitrary number of a messages sent by $\mathtt{p}$ can accumulate in $\mathbb{B}$ before $\mathtt{q}$ receives them. As a consequence, the fact that a session is alive cannot be established in general by means of a brute force algorithm that checks every reachable configuration. By contrast, the session

$$\Delta_2 = \{\mathtt{p} : \mathtt{rec}\ X.\mathtt{q}!a.X\ ,\ \mathtt{q} : \mathtt{rec}\ Y.\mathtt{p}?a.Y\}$$

which is normally regarded correct in other session type theories, is not alive because there is no way for $\mathtt{p}$ and $\mathtt{q}$ to reach a successfully terminated state. The point is that hitherto correctness of session was associated to progress (*ie*, the system is never stuck). This is a weak notion of correctness since, for instance the session $\Delta_2 \uplus \{\mathtt{r} : \mathtt{p}?c.\mathsf{end}\}$ satisfies progress even though role $\mathtt{r}$ starves waiting for its input. While in this example starvation is clear since no c message is ever sent, determining starvation is in general more difficult, as for

$$\Delta_3 = \{\mathtt{p} : \mathtt{rec}\ X.\mathtt{q}!a.\mathtt{q}!b.X\ ,\ \mathtt{q} : \mathtt{rec}\ Y.(\mathtt{p}?a.\mathtt{p}?b.Y + \mathtt{p}?b.\mathtt{r}!c.\mathsf{end})\ ,\ \mathtt{r} : \mathtt{q}?c.\mathsf{end}\}$$

which satisfies progress, where every input corresponds to a compatible output, and viceversa, but which is not alive.

We can now define the traces of a session as the set of sequences of interactions that can occur in every possible reduction. It is convenient to define the traces of an incorrect (*ie*, non-live) session as the empty set (observe that $\mathrm{tr}(\mathscr{G}) \neq \emptyset$ for every $\mathscr{G}$).

Definition 3.4 (session traces).

$$\mathrm{tr}(\Delta) \stackrel{def}{=} \begin{cases} \{\varphi \mid \varepsilon \fatsemi \Delta \stackrel{\varphi}{\Longrightarrow} \varepsilon \fatsemi \{\mathtt{p}_i : \mathsf{end}\}_{i \in I}\} & \textit{if } \Delta \textit{ is a live session} \\ \emptyset & \textit{otherwise} \end{cases}$$

It is easy to verify that $\mathrm{tr}(\Delta_1) = \mathrm{tr}((\mathtt{p} \stackrel{a}{\longrightarrow} \mathtt{q})^*;\mathtt{p} \stackrel{b}{\longrightarrow} \mathtt{q})$ while $\mathrm{tr}(\Delta_2) = \mathrm{tr}(\Delta_3) = \emptyset$ since neither Δ_2 nor Δ_3 is a live session.

4 Semantic projection

In this section we show how to project a global type to the session types of its participants —*ie*, to a session— in such a way that the projection is correct with respect to the global type. Before we move on, we must be more precise about what we mean by correctness of a session Δ with respect to a global type $\mathscr{G}$. In our setting, correctness refers to some relationship between the traces of Δ and those of $\mathscr{G}$. In general, however, we cannot require that $\mathscr{G}$ and Δ have exactly the same traces: when projecting $\mathscr{G}_1 \wedge \mathscr{G}_2$ we might need to impose a particular order in which the interactions specified by $\mathscr{G}_1$ and $\mathscr{G}_2$ must occur (shuffling condition). At the same time, asking only $\mathrm{tr}(\Delta) \subseteq \mathrm{tr}(\mathscr{G})$ would lead us to immediately loose the exhaustivity property, since for instance $\{\mathtt{p} : \mathtt{q}!a.\mathsf{end}\ ,\ \mathtt{q} : \mathtt{p}?a.\mathsf{end}\}$ would implement $\mathtt{p} \stackrel{a}{\longrightarrow} \mathtt{q} \vee \mathtt{p} \stackrel{b}{\longrightarrow} \mathtt{q}$ even though

Table 3. Rules for semantic projection.

$$\text{(SP-SKIP)}\quad \Delta \vdash \mathsf{skip} \triangleright \Delta$$

$$\text{(SP-ACTION)}\quad \{\mathtt{p}_i : T_i\}_{i\in I} \uplus \{\mathtt{p} : T\} \uplus \Delta \vdash \{\mathtt{p}_i\}_{i\in I} \xrightarrow{a} \mathtt{p} \triangleright \{\mathtt{p}_i : \mathtt{p}!a.T_i\}_{i\in I} \uplus \{\mathtt{p} : \{\mathtt{p}_i\}_{i\in I}?a.T\} \uplus \Delta$$

$$\text{(SP-SEQUENCE)}\quad \frac{\Delta \vdash \mathscr{G}_2 \triangleright \Delta' \qquad \Delta' \vdash \mathscr{G}_1 \triangleright \Delta''}{\Delta \vdash \mathscr{G}_1;\mathscr{G}_2 \triangleright \Delta''}$$

$$\text{(SP-ALTERNATIVE)}\quad \frac{\Delta \vdash \mathscr{G}_1 \triangleright \{\mathtt{p} : T_1\} \uplus \Delta' \qquad \Delta \vdash \mathscr{G}_2 \triangleright \{\mathtt{p} : T_2\} \uplus \Delta'}{\Delta \vdash \mathscr{G}_1 \vee \mathscr{G}_2 \triangleright \{\mathtt{p} : T_1 \oplus T_2\} \uplus \Delta'}$$

$$\text{(SP-ITERATION)}\quad \frac{\{\mathtt{p} : T_1 \oplus T_2\} \uplus \Delta \vdash \mathscr{G} \triangleright \{\mathtt{p} : T_1\} \uplus \Delta}{\{\mathtt{p} : T_2\} \uplus \Delta \vdash \mathscr{G}^* \triangleright \{\mathtt{p} : T_1 \oplus T_2\} \uplus \Delta}$$

$$\text{(SP-SUBSUMPTION)}\quad \frac{\Delta \vdash \mathscr{G}' \triangleright \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G} \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \triangleright \Delta''}$$

the implementation systematically exhibits only one ($\mathtt{p} \xrightarrow{a} \mathtt{q}$) of the specified alternative behaviors. In the end, we say that Δ is a correct implementation of $\mathscr{G}$ if: first, every trace of Δ is a trace of $\mathscr{G}$ (*soundness*); second, every trace of $\mathscr{G}$ is the permutation of a trace of Δ (*completeness*). Formally:

$$\mathrm{tr}(\Delta) \subseteq \mathrm{tr}(\mathscr{G}) \subseteq \mathrm{tr}(\Delta)^\circ$$

where L° is the closure of L under arbitrary permutations of the strings in L:

$$L^\circ \stackrel{\text{def}}{=} \{\alpha_1 \cdots \alpha_n \mid \text{there exists a permutation } \sigma \text{ such that } \alpha_{\sigma(1)} \cdots \alpha_{\sigma(n)} \in L\}$$

Since these relations between languages (of traces) play a crucial role, it is convenient to define a suitable pre-order relation:

Definition 4.1 (implementation pre-order). *We let $L_1 \leqslant L_2$ if $L_1 \subseteq L_2 \subseteq L_1^\circ$ and extend it to global types and sessions in the natural way, by considering the corresponding sets of traces. Therefore, we write $\Delta \leqslant \mathscr{G}$ if $\mathrm{tr}(\Delta) \leqslant \mathrm{tr}(\mathscr{G})$.*

It is easy to see that soundness and completeness respectively formalize the notions of fitness and exhaustivity that we have outlined in the introduction. For what concerns the remaining three properties listed in the introduction (*ie*, sequentiality, alternativeness, and shuffling), they are entailed by the formalization of the semantics of a global type in terms of its traces (Definition 2.2). In particular, we have that soundness implies sequentiality and alternativeness, while completeness implies shuffling. Therefore, in the formal treatment that follows we will focus on soundness and completeness as to the only characterizing properties connecting sessions and global types. The relation $\Delta \leqslant \mathscr{G}$ summarizes the fact that Δ is both sound and complete with respect to $\mathscr{G}$, namely that Δ is a correct implementation of the specification $\mathscr{G}$.

Table 3 presents our rules to build the projections of global types. Projecting a global type basically means compiling it to an "equivalent" set of session types. Since the source language (global types) is equipped with sequential composition while the target language (session types) is not, it is convenient to parameterize projection on a

continuation, *ie*, we consider judgments of the shape:

$$\Delta \vdash \mathscr{G} \triangleright \Delta'$$

meaning that if Δ is the projection of some $\mathscr{G}'$, then Δ' is the projection of $\mathscr{G};\mathscr{G}'$. This immediately gives us the rule (SP-SEQUENCE). We say that Δ' is a *projection* of $\mathscr{G}$ with *continuation* Δ.

The projection of an *interaction* $\pi \xrightarrow{a} \mathtt{p}$ adds $\mathtt{p}!a$ in front of the session type of all the roles in π, and $\pi?a$ in front of the session type of $\mathtt{p}$ (rule (SP-ACTION)). For example we have:

$$\{\mathtt{p} : \mathsf{end}, \mathtt{q} : \mathsf{end}\} \vdash \mathtt{p} \xrightarrow{a} \mathtt{q} \triangleright \{\mathtt{p} : \mathtt{q}!a.\mathsf{end},\ \ \mathtt{q} : \mathtt{p}?a.\mathsf{end}\}$$

An *alternative* $\mathscr{G}_1 \vee \mathscr{G}_2$ (rule (SP-ALTERNATIVE)) can be projected only if there is a participant $\mathtt{p}$ that actively chooses among different behaviors by sending different messages, while all the other participants must exhibit the same behavior in both branches. The subsumption rule (SP-SUBSUMPTION) can be used to fulfil this requirement in many cases. For example we have $\Delta_0 \vdash \mathtt{p} \xrightarrow{a} \mathtt{q} \triangleright \{\mathtt{p} : \mathtt{q}!a.\mathsf{end}, \mathtt{q} : \mathtt{p}?a.\mathsf{end}\}$ and $\Delta_0 \vdash \mathtt{p} \xrightarrow{b} \mathtt{q} \triangleright \{\mathtt{p} : \mathtt{q}!b.\mathsf{end}, \mathtt{q} : \mathtt{p}?b.\mathsf{end}\}$, where $\Delta_0 = \{\mathtt{p} : \mathsf{end}, \mathtt{q} : \mathsf{end}\}$. In order to project $\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{p} \xrightarrow{b} \mathtt{q}$ with continuation Δ_0 we derive first by subsumption $\Delta_0 \vdash \mathtt{p} \xrightarrow{a} \mathtt{q} \triangleright \{\mathtt{p} : \mathtt{q}!a.\mathsf{end}\ ,\ \mathtt{q} : T\}$ and $\Delta_0 \vdash \mathtt{p} \xrightarrow{b} \mathtt{q} \triangleright \{\mathtt{p} : \mathtt{q}!b.\mathsf{end}\ ,\ \mathtt{q} : T\}$ where $T = \mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{end}$. Then we obtain

$$\Delta_0 \vdash \mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{p} \xrightarrow{b} \mathtt{q} \triangleright \{\mathtt{p} : \mathtt{q}!a.\mathsf{end} \oplus \mathtt{q}!b.\mathsf{end}\ ,\ \ \mathtt{q} : T\}$$

Notice that rule (SP-ALTERNATIVE) imposes that in alternative branches there must be one *and only one* participant that takes the decision. For instance, the global type

$$\{\mathtt{p}, \mathtt{q}\} \xrightarrow{a} \mathtt{r} \ \vee\ \{\mathtt{p}, \mathtt{q}\} \xrightarrow{b} \mathtt{r}$$

cannot be projected since we would need a covert channel for $\mathtt{p}$ to agree with $\mathtt{q}$ about whether to send to $\mathtt{r}$ the message a or b.

To project a *starred* global type we also require that one participant $\mathtt{p}$ chooses between repeating the loop or exiting by sending messages, while the session types of all other participants are unchanged. If T_1 and T_2 are the session types describing the behavior of $\mathtt{p}$ when it has respectively decided to perform one more iteration or to terminate the iteration, then $T_1 \oplus T_2$ describes the behavior of $\mathtt{p}$ before it takes the decision. The projection rule requires that one execution of $\mathscr{G}$ followed by the choice between T_1 and T_2 projects in a session with type T_1 for $\mathtt{p}$. This is possible only if T_1 is a recursive type, as expected, and it is the premise of rule (SP-ITERATION). For example if $T_1 = \mathtt{q}!a.\mathtt{rec}\, X.(\mathtt{q}!a.X \oplus \mathtt{q}!b.\mathsf{end})$, $T_2 = \mathtt{q}!b.\mathsf{end}$, and $S = \mathtt{rec}\, Y.(\mathtt{p}?a.Y + \mathtt{p}?b.\mathsf{end})$ we can derive $\{\mathtt{p} : T_1 \oplus T_2, \mathtt{q} : S\} \vdash \mathtt{p} \xrightarrow{a} \mathtt{q} \triangleright \{\mathtt{p} : T_1, \mathtt{q} : S\}$ and then

$$\{\mathtt{p} : T_2, \mathtt{q} : S\} \vdash (\mathtt{p} \xrightarrow{a} \mathtt{q})^* \triangleright \{\mathtt{p} : T_1 \oplus T_2,\ \ \mathtt{q} : S\}$$

Notably there is no rule for "$\wedge$", the *both* constructor. We deal with this constructor by observing that all interleavings of the actions in $\mathscr{G}_1$ and $\mathscr{G}_2$ give global types $\mathscr{G}$ such

that $\mathscr{G} \leqslant \mathscr{G}_1 \wedge \mathscr{G}_2$, and therefore we can use the subsumption rule to eliminate every occurrence of $\wedge$. For example, to project the global type $\mathtt{p} \xrightarrow{a} \mathtt{q} \wedge \mathtt{r} \xrightarrow{b} \mathtt{s}$ we can use $\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{b} \mathtt{s}$: since the two actions that compose both global types have disjoint participants, then the projections of these global types (as well as that of $\mathtt{r} \xrightarrow{b} \mathtt{s}; \mathtt{p} \xrightarrow{a} \mathtt{q}$) will have exactly the same set of traces.

Other interesting examples of subsumptions useful for projecting are

$$\mathtt{r} \xrightarrow{b} \mathtt{p}; \mathtt{p} \xrightarrow{a} \mathtt{q} \leqslant (\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{b} \mathtt{p}) \vee (\mathtt{r} \xrightarrow{b} \mathtt{p}; \mathtt{p} \xrightarrow{a} \mathtt{q}) \tag{4}$$

$$\mathtt{r} \xrightarrow{b} \mathtt{p}; (\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{p} \xrightarrow{b} \mathtt{q}) \leqslant (\mathtt{r} \xrightarrow{b} \mathtt{p}; \mathtt{p} \xrightarrow{a} \mathtt{q}) \vee (\mathtt{r} \xrightarrow{b} \mathtt{p}; \mathtt{p} \xrightarrow{b} \mathtt{q}) \tag{5}$$

In (4) the $\leqslant$-larger global type describes the shuffling of two interactions, therefore we can project one particular ordering still preserving completeness. In (5) we take advantage of the flat nature of traces to push the $\vee$ construct where the choice is actually being made.

We are interested in projections without continuations, that is, in judgments of the shape $\{\mathtt{p}_i : \mathsf{end} \mid \mathtt{p}_i \in \mathscr{G}\} \vdash \mathscr{G} \triangleright \Delta$ (where $\mathtt{p} \in \mathscr{G}$ means that $\mathtt{p}$ occurs in $\mathscr{G}$) which we shortly will write as

$$\vdash \mathscr{G} \triangleright \Delta$$

The mere existence of a projection does not mean that the projection behaves as specified in the global type. For example, we have

$$\vdash \mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{a} \mathtt{s} \triangleright \{\mathtt{p} : \mathtt{q}!a.\mathsf{end},\ \ \mathtt{q} : \mathtt{p}?a.\mathsf{end},\ \ \mathtt{r} : \mathtt{s}!a.\mathsf{end},\ \ \mathtt{s} : \mathtt{r}?a.\mathsf{end}\}$$

but the projection admits also the trace $\mathtt{r} \xrightarrow{a} \mathtt{s}; \mathtt{p} \xrightarrow{a} \mathtt{q}$ which is not allowed by the global type. Clearly the problem resides in the global type, which tries to impose a temporal ordering between interactions involving disjoint participants. What we want, in accordance with the traces of a global type $\mathscr{G}_1; \mathscr{G}_2$, is that no interaction in $\mathscr{G}_2$ can be completed before all the interactions in $\mathscr{G}_1$ are completed. More in details:

- an action $\pi \xrightarrow{a} \mathtt{p}$ is completed when the participant $\mathtt{p}$ has received the message a from all the participants in π;
- if $\varphi; \pi \xrightarrow{a} \mathtt{p}; \pi' \xrightarrow{b} \mathtt{p}'; \psi$ is a trace of a global type, then either the action $\pi' \xrightarrow{b} \mathtt{p}'$ cannot be completed before the action $\pi \xrightarrow{a} \mathtt{p}$ is completed, or they can be executed in any order. The first case requires $\mathtt{p}$ to be either $\mathtt{p}'$ or a member of π', in the second case the set of traces must also contain the trace $\varphi; \pi' \xrightarrow{b} \mathtt{p}'; \pi \xrightarrow{a} \mathtt{p}; \psi$.

This leads us to the following definition of well-formed global type.

Definition 4.2 (well-formed global type). *We say that a set of traces L is* well formed *if $\varphi; \pi \xrightarrow{a} \mathtt{p}; \pi' \xrightarrow{b} \mathtt{p}'; \psi \in L$ implies either $\mathtt{p} \in \pi' \cup \{\mathtt{p}'\}$ or $\varphi; \pi' \xrightarrow{b} \mathtt{p}'; \pi \xrightarrow{a} \mathtt{p}; \psi \in L$. We say that a global type $\mathscr{G}$ is* well formed *if so is* $\mathrm{tr}(\mathscr{G})$.

It is easy to decide well-formedness of an arbitrary global type $\mathscr{G}$ by building in a standard way the automaton that recognises the language of traces generated by $\mathscr{G}$.

Projectability and well-formedness must be kept separate because it is sometimes necessary to project ill-formed global types. For example, the ill-formed global type $\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{r} \xrightarrow{a} \mathtt{s}$ above is useful to project $\mathtt{p} \xrightarrow{a} \mathtt{q} \wedge \mathtt{r} \xrightarrow{a} \mathtt{s}$ which is well formed.

Clearly, if a global type is projectable (*ie*, $\vdash \mathscr{G} \triangleright \Delta$ is derivable) then well-formedness of $\mathscr{G}$ is a necessary condition for the soundness and completeness of its projection (*ie*, for $\Delta \leqslant \mathscr{G}$). It turns out that well-formedness is also a sufficient condition for having soundness and completeness of projections, as stated in the following theorem.

Theorem 4.1. *If $\mathscr{G}$ is well formed and $\vdash \mathscr{G} \triangleright \Delta$, then $\Delta \leqslant \mathscr{G}$.*

In summary, if a well-formed global type $\mathscr{G}$ is projectable, then its projection is a *live* projection (it cannot be empty since $\mathrm{tr}(\mathscr{G}) \subseteq \mathrm{tr}(\Delta)^\circ$) which is sound and complete wrt $\mathscr{G}$ and, therefore, satisfies the sequentiality, alternativeness, and shuffling properties outlined in the introduction.

We conclude this section by formally characterizing the three kinds of problematic global types we have described earlier. We start from the least severe problem and move towards the more serious ones. Let $L^{\#}$ denote the smallest well-formed set such that $L \subseteq L^{\#}$.

No sequentiality. Assuming that there is no Δ that is both sound and complete for $\mathscr{G}$, it might be the case that we can find a session whose traces are complete for $\mathscr{G}$ and sound for the global type $\mathscr{G}'$ obtained from $\mathscr{G}$ by turning some « ; »'s into « $\wedge$ »'s. This means that the original global type $\mathscr{G}$ is ill formed, namely that it specifies some sequentiality constraints that are impossible to implement. For instance, $\{\mathtt{p} : \mathtt{q}!a.\mathsf{end},\ \mathtt{q} : \mathtt{p}?a.\mathsf{end},\ \mathtt{r} : \mathtt{s}!b.\mathsf{end},\ \mathtt{s} : \mathtt{r}?b.\mathsf{end}\}$ is a complete but not sound session for the ill-formed global type $\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{r} \xrightarrow{b} \mathtt{s}$ (while it is a sound and complete session for $\mathtt{p} \xrightarrow{a} \mathtt{q} \wedge \mathtt{r} \xrightarrow{b} \mathtt{s}$). We characterize the global types $\mathscr{G}$ that present this error as:

$$\nexists \Delta : \Delta \leqslant \mathscr{G} \text{ and } \exists \Delta : \mathrm{tr}(\mathscr{G}) \subseteq \mathrm{tr}(\Delta) \subseteq \mathrm{tr}(\mathscr{G})^{\#}\,.$$

No knowledge for choice. In this case every session Δ that is complete for $\mathscr{G}$ invariably exhibits some interactions that are not allowed by $\mathscr{G}$ despite the fact that $\mathscr{G}$ is well formed. This happens when the global type specifies alternative behaviors, but some participants do not have enough information to behave consistently. For example, the global type

$$(\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{q} \xrightarrow{a} \mathtt{r};\mathtt{r} \xrightarrow{a} \mathtt{p}) \vee (\mathtt{p} \xrightarrow{b} \mathtt{q};\mathtt{q} \xrightarrow{a} \mathtt{r};\mathtt{r} \xrightarrow{b} \mathtt{p})$$

mandates that $\mathtt{r}$ should send either a or b in accordance with the message that $\mathtt{p}$ sends to $\mathtt{q}$. Unfortunately, $\mathtt{r}$ has no information as to which message $\mathtt{q}$ has received, because $\mathtt{q}$ notifies $\mathtt{r}$ with an a message in both branches. A complete implementation of this global type is

$$\begin{array}{l}\{\mathtt{p} : \mathtt{q}!a.(\mathtt{r}?a.\mathsf{end} + \mathtt{r}?b.\mathsf{end}) \oplus \mathtt{q}!b.(\mathtt{r}?a.\mathsf{end} + \mathtt{r}?b.\mathsf{end}),\\ \ \mathtt{q} : \mathtt{p}?a.\mathtt{r}!a.\mathsf{end} + \mathtt{p}?b.\mathtt{r}!a.\mathsf{end}, \mathtt{r} : \mathtt{q}?a.(\mathtt{q}!a.\mathsf{end} \oplus \mathtt{q}!b.\mathsf{end})\}\end{array}$$

which also produces the traces $\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{q} \xrightarrow{a} \mathtt{r};\mathtt{r} \xrightarrow{b} \mathtt{p}$ and $\mathtt{p} \xrightarrow{b} \mathtt{q};\mathtt{q} \xrightarrow{a} \mathtt{r};\mathtt{r} \xrightarrow{a} \mathtt{p}$. We characterize this error as:

$$\nexists \Delta : \mathrm{tr}(\mathscr{G}) \subseteq \mathrm{tr}(\Delta) \subseteq \mathrm{tr}(\mathscr{G})^{\#} \text{ and } \exists \Delta : \mathrm{tr}(\mathscr{G}) \subseteq \mathrm{tr}(\Delta)\,.$$

No knowledge, no choice. In this case we cannot find a complete session Δ for $\mathscr{G}$. This typically means that $\mathscr{G}$ specifies some combination of incompatible behaviors. For example, the global type $\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{q} \xrightarrow{a} \mathtt{p}$ implies an agreement between $\mathtt{p}$ and $\mathtt{q}$ for establishing who is entitled to send the a message. In a distributed environment, however, there can be no agreement without a previous message exchange. Therefore, we can either have a sound but not complete session that implements just one of the two branches (for example, $\{\mathtt{p} : \mathtt{q}!a.\mathsf{end}, \mathtt{q} : \mathtt{p}?a.\mathsf{end}\}$) or a session like $\{\mathtt{p} : \mathtt{q}!a.\mathtt{q}?a.\mathsf{end}, \mathtt{q} : \mathtt{p}?a.\mathtt{p}!a.\mathsf{end}\}$ where both $\mathtt{p}$ and $\mathtt{q}$ send their message but which is neither sound nor complete. We characterize this error as:

$$\nexists \Delta : \mathrm{tr}(\mathscr{G}) \subseteq \mathrm{tr}(\Delta)\,.$$

5 Algorithmic projection

We now attack the problem of *computing* the projection of a global type. We are looking for an algorithm that "implements" the projection rules of Section 4, that is, that given a session continuation Δ and a global type $\mathscr{G}$, produces a projection Δ' such that $\Delta \vdash \mathscr{G} : \Delta'$. In other terms this algorithm must be sound with respect to the semantic projection (completeness, that is, returning a projection for every global type that is semantically projectable, seems out of reach, yet).

The deduction system in Table 3 is not algorithmic because of two rules: the rule (SP-ITERATION) that does not satisfy the subformula property since the context Δ used in the premises is the result of the conclusion; the rule (SP-SUBSUMPTION) since it is neither syntax-directed (it is defined for a generic $\mathscr{G}$) nor does it satisfy the subformula property (the $\mathscr{G}'$ and Δ'' in the premises are not uniquely determined).[3] The latter rule can be expressed as the composition of the two rules

$$\text{(SP-SUBSUMPTIONG)}\ \frac{\Delta \vdash \mathscr{G}' \triangleright \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G}}{\Delta \vdash \mathscr{G} \triangleright \Delta'} \qquad \text{(SP-SUBSUMPTIONS)}\ \frac{\Delta \vdash \mathscr{G} \triangleright \Delta' \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \triangleright \Delta''}$$

Splitting (SP-SUBSUMPTION) into (SP-SUBSUMPTIONG) and (SP-SUBSUMPTIONS) is useful to explain the following problems we have to tackle to define an algorithm:

1. How to eliminate (SP-SUBSUMPTIONS), the subsumption rule for sessions.
2. How to define an algorithmic version of (SP-ITERATION), the rule for Kleene star.
3. How to eliminate (SP-SUBSUMPTIONG), the subsumption rule for global types.

We address each problem in order and discuss the related rule in the next sections.

5.1 Session subsumption

Rule (SP-SUBSUMPTIONS) is needed to project alternative branches and iterations (a loop is an unbound repetition of alternatives, each one starting with the choice of

[3] The rule (SP-ALTERNATIVE) is algorithmic: in fact there is a finite number of participants in the two sessions of the premises and at most one of them can have different session types starting with outputs.

whether to enter the loop or to skip it): each participant different from the one that actively chooses must behave according to the same session type in both branches. More precisely, to project $\mathscr{G}_1 \vee \mathscr{G}_2$ the rule (SP-ALTERNATIVE) requires to deduce for $\mathscr{G}_1$ and $\mathscr{G}_2$ the same projection: if different projections are deduced, then they must be previously subsumed to a common lower bound. The algorithmic projection of an alternative (see the corresponding rule in Table 4) allows premises with two different sessions, but then *merges* them. Of course not every pair of projections is mergeable. Intuitively, two projections are mergeable if so are the behaviors of each participant. This requires participants to respect a precise behavior: as long as a participant cannot determine in which branch (*ie*, projection) it is, then it must do the same actions in all branches (*ie*, projections). For example, to project $\mathscr{G} = (\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{r} \xrightarrow{c} \mathtt{q};\dots) \vee (\mathtt{p} \xrightarrow{b} \mathtt{q};\mathtt{r} \xrightarrow{c} \mathtt{q};\dots)$ we project each branch separately obtaining $\Delta_1 = \{\mathtt{p} : \mathtt{q}!a\dots, \mathtt{q} : \mathtt{p}?a.\mathtt{r}?c\dots, \mathtt{r} : \mathtt{q}!c\dots\}$ and $\Delta_2 = \{\mathtt{p} : \mathtt{q}!b\dots, \mathtt{q} : \mathtt{p}?b.\mathtt{r}?c\dots, \mathtt{r} : \mathtt{q}!c\dots\}$. Since p performs the choice, in the projection of $\mathscr{G}$ we obtain $\mathtt{p} : \mathtt{q}!a\dots \oplus \mathtt{q}!b\dots$ and we must merge $\{\mathtt{q} : \mathtt{p}?a.\mathtt{r}?c\dots, \mathtt{r} : \mathtt{q}!c\dots\}$ with $\{\mathtt{q} : \mathtt{p}?b.\mathtt{r}?c\dots, \mathtt{r} : \mathtt{q}!c\dots\}$. Regarding q, observe that it is the receiver of the message from p, therefore it becomes aware of the choice and can behave differently right after the first input operation. Merging its behaviors yields $\mathtt{q} : \mathtt{p}?a.\mathtt{r}?c\dots + \mathtt{p}?b.\mathtt{r}?c\dots$. Regarding r, it has no information as to which choice has been made by p, therefore it must have the same behavior in both branches, as is the case. Since merging is idempotent, we obtain $\mathtt{r} : \mathtt{q}!c\dots$. In summary, *mergeability* of two branches of an "$\vee$" corresponds to the "awareness" of the choice made when branching (see the discussion in Section 4 about the "No knowledge for choice" error), and it is possible when, roughly, each participant performs the same internal choices and disjoint external choices in the two sessions.

Special care must be taken when merging external choices to avoid unexpected interactions that may invalidate the correctness of the projection. To illustrate the problem consider the session types $T = \mathtt{p}?a.\mathtt{q}?b.\mathsf{end}$ and $S = \mathtt{q}?b.\mathsf{end}$ describing the behavior of a participant r. If we let r behave according to the merge of T and S, which intuitively is the external choice $\mathtt{p}?a.\mathtt{q}?b.\mathsf{end} + \mathtt{q}?b.\mathsf{end}$, it may be possible that the message b from q is read *before* the message a from p arrives. Therefore, r may mistakenly think that it should no longer participate to the session, while there is still a message targeted to r that will never be read. Therefore, T and S are *incompatible* and it is not possible to merge them safely. On the contrary, $\mathtt{p}?a.\mathtt{p}?b.\mathsf{end}$ and $\mathtt{p}?b.\mathsf{end}$ are compatible and can be merged to $\mathtt{p}?a.\mathtt{p}?b.\mathsf{end} + \mathtt{p}?b.\mathsf{end}$. In this case, since the order of messages coming from the same sender is preserved, it is not possible for r to read the b message coming from p before the a message, assuming that p sent both. More formally:

Definition 5.1 (compatibility). *We say that an input* $\mathtt{p}?a$ *is* compatible *with a session type T if either (i)* $\mathtt{p}?a$ *does not occur in T, or (ii) $T = \bigoplus_{i\in I} \mathtt{p}_i!a_i.T_i$ and* $\mathtt{p}?a$ *is compatible with T_i for all $i \in I$, or (iii) $T = \sum_{i\in I} \pi_i?a_i.T_i$ and for all $i \in I$ either $\mathtt{p} \in \pi_i$ and $a \neq a_i$ or $\mathtt{p} \notin \pi_i$ and* $\mathtt{p}?a$ *is compatible with T_i.*

We say that an input $\pi?a$ *is* compatible *with a session type T if* $\mathtt{p}?a$ *is compatible with T for some* $\mathtt{p} \in \pi$.

Finally, $T = \sum_{i\in I} \pi_i?a_i.T_i + \sum_{j\in J} \pi_j?a_j.T_j$ and $S = \sum_{i\in I} \pi_i?a_i.S_i + \sum_{h\in H} \pi_h?a_h.S_h$ are compatible *if $\pi_j?a_j$ is compatible with S for all $j \in J$ and $\pi_h?a_h$ is compatible with T for all $h \in H$.*

Table 4. Rules for algorithmic projection.

$$\text{(AP-SKIP)}\quad \Delta \vdash_{\mathsf{a}} \mathsf{skip} \triangleright \Delta$$

$$\text{(AP-ACTION)}\quad \{\mathtt{p}_i : T_i\}_{i\in I} \uplus \{\mathtt{p} : T\} \uplus \Delta \vdash_{\mathsf{a}} \{\mathtt{p}_i\}_{i\in I} \xrightarrow{a} \mathtt{p} \triangleright \{\mathtt{p}_i : \mathtt{p}!a.T_i\}_{i\in I} \uplus \{\mathtt{p} : \{\mathtt{p}_i\}_{i\in I}?a.T\} \uplus \Delta$$

$$\text{(AP-SEQUENCE)}\quad \frac{\Delta \vdash_{\mathsf{a}} \mathscr{G}_2 \triangleright \Delta' \qquad \Delta' \vdash_{\mathsf{a}} \mathscr{G}_1 \triangleright \Delta''}{\Delta \vdash_{\mathsf{a}} \mathscr{G}_1;\mathscr{G}_2 \triangleright \Delta''}$$

$$\text{(AP-ALTERNATIVE)}\quad \frac{\Delta \vdash_{\mathsf{a}} \mathscr{G}_1 \triangleright \{\mathtt{p} : T_1\} \uplus \Delta_1 \qquad \Delta \vdash_{\mathsf{a}} \mathscr{G}_2 \triangleright \{\mathtt{p} : T_2\} \uplus \Delta_2}{\Delta \vdash_{\mathsf{a}} \mathscr{G}_1 \vee \mathscr{G}_2 \triangleright \{\mathtt{p} : T_1 \oplus T_2\} \uplus (\Delta_1 \between \Delta_2)}$$

$$\text{(AP-ITERATION)}\quad \frac{\{\mathtt{p} : X\} \uplus \{\mathtt{p}_i : X_i\}_{i\in I} \vdash_{\mathsf{a}} \mathscr{G} \triangleright \{\mathtt{p} : S\} \uplus \{\mathtt{p}_i : S_i\}_{i\in I}}{\{\mathtt{p} : T\} \uplus \{\mathtt{p}_i : T_i\}_{i\in I} \uplus \Delta \vdash_{\mathsf{a}} \mathscr{G}^* \triangleright \{\mathtt{p} : \mathtt{rec}\, X.(T \oplus S)\} \uplus \{\mathtt{p}_i : \mathtt{rec}\, X_i.(T_i \between S_i)\}_{i\in I} \uplus \Delta}$$

The merge operator just connects sessions with the *same* output guards by internal choices and with *compatible* input guards by external choices:

Definition 5.2 (merge). *The* merge *of T and S, written $T \between S$, is defined coinductively and by cases on the structure of T and S thus:*

- *if $T = S = \mathsf{end}$, then $T \between S = \mathsf{end}$;*
- *if $T = \bigoplus_{i\in I} \mathtt{p}_i!a_i.T_i$ and $S = \bigoplus_{i\in I} \mathtt{p}_i!a_i.S_i$, then $T \between S = \bigoplus_{i\in I} \mathtt{p}_i!a_i.(T_i \between S_i)$;*
- *if $T = \sum_{i\in I} \pi_i?a_i.T_i + \sum_{j\in J} \pi_j?a_j.T_j$ and $S = \sum_{i\in I} \pi_i?a_i.S_i + \sum_{h\in H} \pi_h?a_h.S_h$ are compatible, then $T \between S = \sum_{i\in I} \pi_i?a_i.(T_i \between S_i) + \sum_{j\in J} \pi_j?a_j.T_j + \sum_{h\in H} \pi_h?a_h.S_h$.*

We extend merging to sessions so that $\Delta \between \Delta' = \{\mathtt{p} : T \between S \mid \mathtt{p} : T \in \Delta \;\&\; \mathtt{p} : S \in \Delta'\}$.

Rules (AP-ALTERNATIVE) and (AP-ITERATION) of Table 4 are the algorithmic versions of (SP-ALTERNATIVE) and (SP-ITERATION), but instead of relying on subsumption they use the merge operator to compute common behaviors.

The merge operation is a sound but incomplete approximation of session subsumption insofar as the merge of two sessions can be undefined even though the two sessions completed with the participant that makes the decision have a common lower bound according to $\leqslant$. This implies that there are global types which can be semantically but not algorithmically projected. Take for example $\mathscr{G}_1 \vee \mathscr{G}_2$ where $\mathscr{G}_1 = \mathtt{p} \xrightarrow{a} \mathtt{r};\mathtt{r} \xrightarrow{a} \mathtt{p};\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{q} \xrightarrow{b} \mathtt{r}$ and $\mathscr{G}_2 = \mathtt{p} \xrightarrow{b} \mathtt{q};\mathtt{q} \xrightarrow{b} \mathtt{r}$. The behavior of $\mathtt{r}$ in $\mathscr{G}_1$ and $\mathscr{G}_2$ respectively is $T = \mathtt{p}?a.\mathtt{p}!a.\mathtt{q}?b.\mathsf{end}$ and $S = \mathtt{q}?b$. Then we see that $\mathscr{G}_1 \vee \mathscr{G}_2$ is semantically projectable, for instance by inferring the behavior $T + S$ for $\mathtt{r}$. However, T and S are incompatible and $\mathscr{G}_1 \vee \mathscr{G}_2$ is not algorithmically projectable. The point is that the $\leqslant$ relation on projections has a comprehensive perspective of the *whole* session and "realizes" that, if $\mathtt{p}$ initially chooses to send a, then $\mathtt{r}$ will not receive a b message coming from $\mathtt{q}$ until $\mathtt{r}$ has sent a to $\mathtt{p}$. The merge operator, on the other hand, is defined locally on pairs of session types and ignores that the a message that $\mathtt{r}$ sends to $\mathtt{p}$ is used to enforce the arrival of the b message from $\mathtt{q}$ to $\mathtt{r}$ only afterwards. For this reason it conservatively declares T and S incompatible, making $\mathscr{G}_1 \vee \mathscr{G}_2$ impossible to project algorithmically.

5.2 Projection of Kleene star

Since an iteration $\mathscr{G}^*$ is intuitively equivalent to $\mathsf{skip} \vee \mathscr{G};\mathscr{G}^*$ it comes as no surprise that the algorithmic rule (AP-ITERATION) uses the merge operator. The use of recursion variables for continuations is also natural: in the premise we project $\mathscr{G}$ taking recursion variables as session types in the continuation; the conclusion projects $\mathscr{G}^*$ as the choice between exiting and entering the loop. There is, however, a subtle point in this rule that may go unnoticed: although in the premises of (AP-ITERATION) the only actions and roles taken into account are those occurring in $\mathscr{G}$, in its conclusion the projection of $\mathscr{G}^*$ may require a continuation that includes actions and roles that precede $\mathscr{G}^*$. The point can be illustrated by the global type

$$(\mathtt{p} \xrightarrow{a} \mathtt{q};(\mathtt{p} \xrightarrow{b} \mathtt{q})^*)^*;\mathtt{p} \xrightarrow{c} \mathtt{q}$$

where $\mathtt{p}$ initially decides whether to enter the outermost iteration (by sending a) or not (by sending c). If it enters the iteration, then it eventually decides whether to also enter the innermost iteration (by sending b), whether to repeat the outermost one (by sending a), or to exit both (by sending c). Therefore, when we project $(\mathtt{p} \xrightarrow{b} \mathtt{q})^*$, we must do it in a context in which both $\mathtt{p} \xrightarrow{c} \mathtt{q}$ and $\mathtt{p} \xrightarrow{a} \mathtt{q}$ are possible, that is a continuation of the form $\{\mathtt{p} : \mathtt{q}!a \ldots \oplus \mathtt{q}!c.\mathsf{end}\}$ even though no a is sent by an action (syntactically) following $(\mathtt{p} \xrightarrow{b} \mathtt{q})^*$. For the same reason, the projection of $(\mathtt{p} \xrightarrow{b} \mathtt{q})^*$ in $(\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{p} \xrightarrow{a} \mathtt{r};(\mathtt{p} \xrightarrow{b} \mathtt{q})^*)^*;\mathtt{p} \xrightarrow{c} \mathtt{q};\mathtt{q} \xrightarrow{c} \mathtt{r}$ will need a recursive session type for $\mathtt{r}$ in the continuation.

5.3 Global type subsumption

Elimination of global type subsumption is the most difficult problem when defining the projection algorithm. While in the case of sessions the definition of the merge operator gives us a sound—though not complete—tool that replaces session subsumption in very specific places, we do not have such a tool for global type containment. This is unfortunate since global type subsumption is necessary to project several usage patterns (see for example the inequations (4) and (5)), but most importantly it is the only way to eliminate $\wedge$-types (neither the semantic nor the algorithmic deduction systems have projection rules for $\wedge$). The minimal facility that a projection algorithm should provide is to feed the algorithmic rules with all the variants of a global type obtained by replacing occurrences of $\mathscr{G}_1 \wedge \mathscr{G}_2$ by either $\mathscr{G}_1;\mathscr{G}_2$ or $\mathscr{G}_2;\mathscr{G}_1$. Unfortunately, this is not enough to cover all the occurrences in which rule (SP-SUBSUMPTIONG) is necessary. Indeed, while $\mathscr{G}_1;\mathscr{G}_2$ and $\mathscr{G}_2;\mathscr{G}_1$ are in many cases projectable (for instance, when $\mathscr{G}_1$ and $\mathscr{G}_2$ have distinct roles and are both projectable), there exist $\mathscr{G}_1$ and $\mathscr{G}_2$ such that $\mathscr{G}_1 \wedge \mathscr{G}_2$ is projectable only by considering a clever interleaving of the actions occurring in them. Consider for instance $\mathscr{G}_1 = (\mathtt{p} \xrightarrow{a} \mathtt{q};\mathtt{q} \xrightarrow{c} \mathtt{s};\mathtt{s} \xrightarrow{e} \mathtt{q}) \vee (\mathtt{p} \xrightarrow{b} \mathtt{r};\mathtt{r} \xrightarrow{d} \mathtt{s};\mathtt{s} \xrightarrow{f} \mathtt{r})$ and $\mathscr{G}_2 = \mathtt{r} \xrightarrow{g} \mathtt{s};\mathtt{s} \xrightarrow{h} \mathtt{r};\mathtt{s} \xrightarrow{i} \mathtt{q}$. The projection of $\mathscr{G}_1 \wedge \mathscr{G}_2$ from the environment $\{\mathtt{q} : \mathtt{p}!a.\mathsf{end}, \mathtt{r} : \mathtt{p}!b.\mathsf{end}\}$ can be obtained only from the interleaving $\mathtt{r} \xrightarrow{g} \mathtt{s};\mathscr{G}_1;\mathtt{s} \xrightarrow{h} \mathtt{r};\mathtt{s} \xrightarrow{i} \mathtt{q}$. The reason is that $\mathtt{q}$ and $\mathtt{r}$ receive messages only in one of the two branches

of the $\vee$, so we need to compute the $\between$ of their types in these branches with their types in the continuations. The example shows that to project $\mathscr{G}_1 \wedge \mathscr{G}_2$ it may be necessary to arbitrarily decompose one or both of $\mathscr{G}_1$ and $\mathscr{G}_2$ to find the particular interleaving of actions that can be projected. As long as $\mathscr{G}_1$ and $\mathscr{G}_2$ are finite (no non-trivial iteration occurs in them), we can use a brute force approach and try to project all the elements in their shuffle, since there are only finitely many of them. In general —*ie*, in presence of iteration— this is not an effective solution. However, we conjecture that even in the presence of infinitely many traces one may always resort to the finite case by considering only zero, one, and two unfoldings of starred global types. To give a rough idea of the intuition supporting this conjecture consider the global type $\mathscr{G}^* \wedge \mathscr{G}'$: its projectability requires the projectability of $\mathscr{G}'$ (since $\mathscr{G}$ can be iterated zero times), of $\mathscr{G} \wedge \mathscr{G}'$ (since $\mathscr{G}$ can occur only once) and of $\mathscr{G};\mathscr{G}$ (since the number of occurences of $\mathscr{G}$ is unbounded). It is enough to require also that either $\mathscr{G};(\mathscr{G} \wedge \mathscr{G}')$ or $(\mathscr{G} \wedge \mathscr{G}');\mathscr{G}$ can be projected, since then the projectability of either $\mathscr{G}^n;(\mathscr{G} \wedge \mathscr{G}')$ or $(\mathscr{G} \wedge \mathscr{G}');\mathscr{G}^n$ for an arbitrary n follows (see the appendix in the extended version).

So we can —or, conjecture we can— get rid of all occurences of $\wedge$ operators automatically, without loosing in projectability. However, examples (4) and (5) in Section 4 show that rule (SP-SUBSUMPTIONG) is useful to project also global types in which the $\wedge$-constructor does not occur. A fully automated approach may consider (4) and (5) as right-to-left rewriting rules that, in conjunction with some other rules, form a rewriting system generating a set of global types to be fed to the algorithm of Table 4. The choice of such rewriting rules must rely on a more thorough study to formally characterize the sensible classes of approximations to be used in the algorithms. An alternative approach is to consider a global type $\mathscr{G}$ as somewhat underspecified, in that it may allow for a large number of *different* implementations (exhibiting *different* sets of traces) that are sound and complete. Therefore, rule (SP-SUBSUMPTIONG) may be interpreted as a human-assisted refinement process where the designer of a system proposes one particular implementation $\mathscr{G} \leqslant \mathscr{G}'$ of a system described by $\mathscr{G}'$. In this respect it is interesting to observe that checking whether $L_1 \leqslant L_2$ when L_1 and L_2 are regular is decidable, since this is a direct consequence of the decidability of the Parikh equivalence on regular languages [18].

5.4 Properties of the algorithmic rules

Every deduction of the algorithmic system given in Table 4, possibly preceeded by the elimination of $\wedge$ and other possible sources of failures by applying the rewritings/heuristics outlined in the previous subsection, induces a similar deduction using the rules for semantic projection (Table 3).

Theorem 5.1. *If* $\vdash_{\mathsf{a}} \mathscr{G} \triangleright \Delta$, *then* $\vdash \mathscr{G} \triangleright \Delta$.

As a corollary of Theorems 4.1 and 5.1, we immediately obtain that the projection Δ of $\mathscr{G}$ obtained through the algorithm is sound and complete with respect to $\mathscr{G}$.

6 k-Exit Iterations

The syntax of global types (Table 1) includes that of regular expressions and therefore is expressive enough for describing any protocol that follows a regular pattern. Nonetheless, the simple Kleene star prevents us from projecting some useful protocols. To illustrate the point, suppose we want to describe an interaction where two participants $\mathtt{p}$ and $\mathtt{q}$ alternate in a negotiation in which each of them may decide to bail out. On $\mathtt{p}$'s turn, $\mathtt{p}$ sends either a *bailout* message or a *handover* message to $\mathtt{q}$; if a *bailout* message is sent, the negotiation ends, otherwise it continues with $\mathtt{q}$ that behaves in a symmetric way. The global type

$$(\mathtt{p} \overset{\mathit{handover}}{\longrightarrow} \mathtt{q};\mathtt{q} \overset{\mathit{handover}}{\longrightarrow} \mathtt{p})^*;(\mathtt{p} \overset{\mathit{bailout}}{\longrightarrow} \mathtt{q} \vee \mathtt{p} \overset{\mathit{handover}}{\longrightarrow} \mathtt{q};\mathtt{q} \overset{\mathit{bailout}}{\longrightarrow} \mathtt{p})$$

describes this protocol as an arbitrarily long negotiation that may end in two possible ways, according to the participant that chooses to bail out. This global type cannot be projected because of the two occurrences of the interaction $\mathtt{p} \overset{\mathit{handover}}{\longrightarrow} \mathtt{q}$, which make it ambiguous whether $\mathtt{p}$ actually chooses to bail out or to continue the negotiation. In general, our projection rules (SP-ITERATION) and (AP-ITERATION) make the assumption that an iteration can be exited in one way only, while in this case there are two possibilities according to which role bails out. This lack of expressiveness of the simple Kleene star used in a nondeterministic setting [17] led researchers to seek for alternative iteration constructs. One proposal is the *k-exit iteration* [2], which is a generalization of the binary Kleene star and has the form

$$(\mathscr{G}_1,\ldots,\mathscr{G}_k)^{\,k*}\,(\mathscr{G}'_1,\ldots,\mathscr{G}'_k)$$

indicating a loop consisting of k subsequent phases $\mathscr{G}_1,\ldots,\mathscr{G}_k$. The loop can be exited just before each phase through the corresponding $\mathscr{G}'_i$. Formally, the traces of the k-exit iteration can be expressed thus:

$$\mathrm{tr}((\mathscr{G}_1,\ldots,\mathscr{G}_k)^{\,k*}\,(\mathscr{G}'_1,\ldots,\mathscr{G}'_k)) \overset{\text{def}}{=} \mathrm{tr}((\mathscr{G}_1;\ldots;\mathscr{G}_k)^*;(\mathscr{G}'_1 \vee \mathscr{G}_1;\mathscr{G}'_2 \vee \cdots \vee \mathscr{G}_1;\ldots;\mathscr{G}_{k-1};\mathscr{G}'_k))$$

and, for example, the negotiation above can be represented as the global type

$$(\mathtt{p} \overset{\mathit{handover}}{\longrightarrow} \mathtt{q},\mathtt{q} \overset{\mathit{handover}}{\longrightarrow} \mathtt{p})^{\,2*}\,(\mathtt{p} \overset{\mathit{bailout}}{\longrightarrow} \mathtt{q},\mathtt{q} \overset{\mathit{bailout}}{\longrightarrow} \mathtt{p}) \tag{6}$$

while the unary Kleene star $\mathscr{G}^*$ can be encoded as $(\mathscr{G})^{\,1*}\,(\mathsf{skip})$.

In our setting, the advantage of the k-exit iteration over the Kleene star is that it syntactically identifies the k points in which a decision is made by a participant of a multi-party session and, in this way, it enables more sophisticated projection rules such as that in Table 5. Albeit intimidating, rule (SP-k-EXIT ITERATION) is just a generalization of rule (SP-ITERATION). For each phase i a (distinct) participant $\mathtt{p}_i$ is identified: the participant may decide to exit the loop behaving as S_i or to continue the iteration behaving as T_i. While projecting each phase $\mathscr{G}_i$, the participant $\mathtt{p}_{(i \bmod k)+1}$ that will decide at the next turn is given the continuation $T_{(i \bmod k)+1} \oplus S_{(i \bmod k)+1}$, while the others must behave according to some R_i that is the same for every phase in which

Table 5. Semantic projection of k-exit iteration.

(SP-k-EXIT ITERATION)

$$\frac{\begin{array}{c}\Delta \vdash \mathscr{G}_i' \triangleright \{\mathtt{p}_i : S_i\} \uplus \{\mathtt{p}_j : R_j\}_{j=1,\ldots,i-1,i+1,\ldots,k} \uplus \Delta' \ {\scriptstyle (i\in\{1,\ldots,k\})} \\ \{\mathtt{p}_2 : T_2 \oplus S_2\} \uplus \{\mathtt{p}_i : R_i\}_{i=1,3,\ldots,k} \uplus \Delta' \vdash \mathscr{G}_1 \triangleright \{\mathtt{p}_1 : T_1\} \uplus \{\mathtt{p}_i : R_i\}_{i=2,\ldots,k} \uplus \Delta' \\ \{\mathtt{p}_3 : T_3 \oplus S_3\} \uplus \{\mathtt{p}_i : R_i\}_{i=1,2,4,\ldots,k} \uplus \Delta' \vdash \mathscr{G}_2 \triangleright \{\mathtt{p}_2 : T_2\} \uplus \{\mathtt{p}_i : R_i\}_{i=1,3,\ldots,k} \uplus \Delta' \\ \vdots \\ \{\mathtt{p}_1 : T_1 \oplus S_1\} \uplus \{\mathtt{p}_i : R_i\}_{i=2,\ldots,k} \uplus \Delta' \vdash \mathscr{G}_k \triangleright \{\mathtt{p}_k : T_k\} \uplus \{\mathtt{p}_i : R_i\}_{i=1,\ldots,k-1} \uplus \Delta'\end{array}}{\Delta \vdash (\mathscr{G}_1,\ldots,\mathscr{G}_k)^{k*} (\mathscr{G}_1',\ldots,\mathscr{G}_k') \triangleright \{\mathtt{p}_1 : T_1 \oplus S_1\} \uplus \{\mathtt{p}_i : R_i\}_{i=2,\ldots,k} \uplus \Delta'}$$

they play no active role. Once again, rule (SP-SUBSUMPTION) is required in order to synthesize these behaviors. For example, the global type (6) is projected to

$$\begin{array}{l}\{\mathtt{p} : \mathtt{rec}\ X.(\mathtt{q}!\mathit{handover}.(\mathtt{q}?\mathit{handover}.X + \mathtt{q}?\mathit{bailout}.\mathsf{end}) \oplus \mathtt{q}!\mathit{bailout}.\mathsf{end}), \\ \ \mathtt{q} : \mathtt{rec}\ Y.(\mathtt{p}?\mathit{handover}.(\mathtt{p}!\mathit{handover}.Y \oplus \mathtt{p}!\mathit{bailout}.\mathsf{end}) + \mathtt{p}?\mathit{bailout}.\mathsf{end})\}\end{array}$$

as one expects.

7 Related work

The formalization and analysis of the relation between a global description of a distributed system and a more machine-oriented description of a set of components that implements it, is a problem that has been studied in several contexts and by different communities. In this context, important properties that are consider are the *verification* that an implementation satisfies the specification, the *implementability* of the specification by automatically producing an implementation from it, and the study of different properties on the specification that can then be transposed to every (possibly automatically produced) implementation satisfying it. In this work we concentrated on the implementability problem, and we tackled it from the "Web service coordination" angle developed by the community that focuses on behavioral types and process algebrae. We are just the latest to attack this problem. So many other communities have been considering it before us that even a sketchy survey has no chance to be exhaustive.

In what follows we compare the "behavioral types/process algebra" approach we adopted, with two alternative approaches studied by important communities with a large amount of different and important contributions, namely the "automata" and "cryptographic protocols" approaches. In the full version of this article the reader will find a deeper survey of these two approaches along with a more complete comparison. In a nutshell, the "automata/model checking" community has probably done the most extensive research on the problem. The paradigmatic global descriptions language they usually refer to are *Message Sequence Charts* (MSC, ITU Z.120 standard) enriched with branching and iteration (which are then called *Message Sequence Graphs* or, as in the Z.120 standard, *High-Level Message Sequence Charts*) and they are usually projected into *Communicating Finite State Machines* (CFM) which form the theoretical core of the *Specification and Description Language* (SDL ITU Z.100 standard). This community has investigated the expressive power of the two formalisms and their properties,

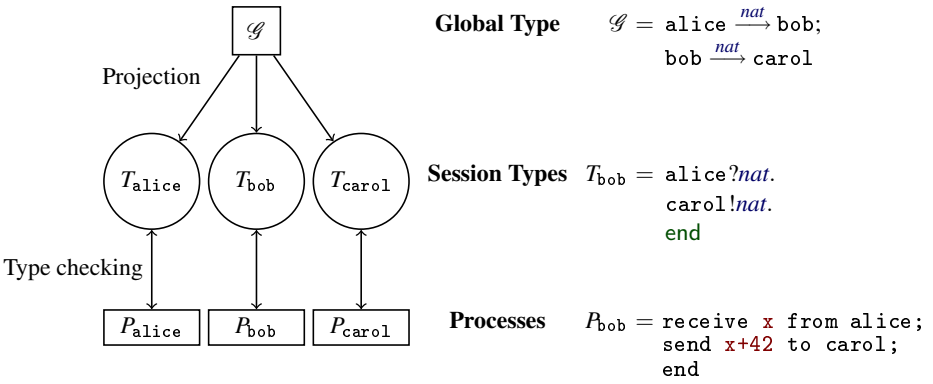

Fig. 1. Global types and multiparty sessions in a nutshell.

studied different notions of implementability (but not the notion we studied here which, as far as we know, is original to our work), and several variants of these formalisms especially to deal with the decidability or tractability of the verification of properties, in particular model-checking. The community that works on the formal verification of cryptographic protocols uses MSC as global descriptions, as well, though they are of different nature from the previous ones. In particular, for cryptographic protocols much less emphasis is put on control (branching and iteration have a secondary role) and expressivity, while much more effort is devoted to the description of the messages (these include cryptographic primitives, at least), of properties of the participants, and of local treatment of the messages. The global descriptions are then projected into local descriptions that are more detailed than in the automata approach since they precisely track single values and the manipulations thereof. The verification of properties is finer grained and covers execution scenari that fall outside the global description since the roles described in the global type can be concurrently played by different participants, messages can be intercepted, read, destroyed, and forged and, generally, the communication topology may be changed. Furthermore different executions of the protocol may be not independent as attackers can store and detour information in one execution to use it in a later execution.

Our work springs from the research done to formally describe and verify compositions of Web services. This research has mainly centered on using process algebras to describe and verify visible local behavior of services and just recently (all the references date of the last five years) has started to consider global *choreographic* descriptions of multiple services and the problem of their projection. This yielded the three layered structure depicted in Figure 1 where a global type describing the choreography is projected into a set of session types that are then used to type-check the processes that implement it (as well as guide their implementation). The study thus focuses on defining the relation between the different layers. Implementability is the relation between the first and second layer. Here the important properties are that projection produces systems that are sound and complete with respect to the global description (in the sense stated by Theorem 4.1) and deadlock free (*eg*, we bar out specifications as

$\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{p} \xrightarrow{a} \mathtt{r}$ when it has no continuation, since whatever the choice either q or r will be stuck). Typeability is the relation between the second and third layer. Here the important properties are subject reduction (well-typed processes reduce only to well-typed processes) and progress (which in this context implies deadlock freedom).

Although in this work we disregarded the lower layer of processes, it is nevertheless an essential component of this research. In particular, it explains the nature of the messages that characterize this approach, which are *types*. One of the principal aims of this research, thus, is to find the right level of abstraction that must be expressed by types and session types. Consider again Figure 1. The process layer clearly shows the relation between the message received by bob and the one it sends to carol, but this relation (actually, any relation) is abstracted out both in the session and the global type layers. The level of abstraction is greater than that of cryptographic protocols since values are not tracked by global descriptions. Although tracking of values could be partially recovered by resorting to singleton types, there is a particular class of values that deserves special care and whose handling is one of the main future challenges of this research, that is, *channels*. The goal is to include higher order types in global specifications thus enabling the transmission of session channels and therefore the reification of dynamic reconfiguration of session topology. We thus aim at defining reconfiguration in the specification itself, as opposed to the case of cryptographic protocols where the reconfiguration of the communication topology is considered at meta-level for verification purposes. As a matter of fact, this feature has already been studied in the literature. For instance, the extension of WS-CDL [1] with channel passing is studied in [11] (as the automata approach has the MSC as their reference standard, so the Web service-oriented research refers to the WS-CDL standard whose implementability has been studied in [19]); the paper that first introduced global descriptions for session types [9] explicitly mentions channels in messages that can be sent to other participants to open new sessions on them. In our opinion the existing works on session types are deeply syntactic in nature and suffer from the fact that their global types are defined in function of the languages used to define processes and session types. The consequence is that the design choices done in defining session types are amplified in the passage to global types yielding a somewhat unnatural syntax for global types and restrictive conditions devoid of semantic characterizations. Here we preferred to take a step back and to start by defining global descriptions whose restrictions are semantically justified. So we favored a less rich language with few semantically justified features and leave the addition of more advanced features for a later time.

Coming back to the comparison of the three approaches, the Web service-oriented approach shares several features in common with the other two. As for the automata approach we (in the sense of the Web service-oriented research) focus on the expressiveness of the control, the possibility of branching and iterate, and the effective implementability into deadlock-free local descriptions. However the tendency for Web services is to impose syntactic restrictions from the beginning rather than study the general case and then devise appropriate restrictions with the sought properties (in this respects our work and those of Bravetti, Zavattaro and Lanese [6, 7, 5] are few exceptions in the panorama of the Web service approach). Commonalities with the cryptographic protocol approach are more technical. In particular we share the dynamism of the com-

munication topology (with the caveat about whether this dynamism is performed at the linguistic or meta-linguistic level) and the robustness with respect to reconfiguration (the projected session types should ensure that well-typed process will be deadlock free even in the presence of multiple interleaved sessions and session delegation, though few works actually enforce this property [3, 13]). As for cryptographic protocols, this dynamism is also accounted at level of participants since recent work in session types studies global descriptions of roles that can then be implemented by several different agents [12]. Finally, there are some characteristics specific to our approach such as the exploration of new linguistic features (for instance in this work we introduced actions with multi-senders and encoded multi-receivers) and a pervasive use of compositional deduction systems that we inherit from type theory. We conclude this section with a more in-depth description of the main references in this specific area so as to give a more detailed comparison with our work.

Multiparty session types. Global types were introduced in [9] for dyadic sessions and in [16] for multi-party sessions. Channels are present in the global types of both [9] and [16] while the first also allows messages to have a complex structure. Their presence, however, requires the definition of syntactic restrictions that ensure projectability: channels need to be "well-threaded" (to avoid that the use of different channels disrupts the sequentiality constraints of the specification) and message structures must be used "coherently" in different threads (to assure that a fixed server offers the same services to different clients). We did not include such features in our treatment since we wanted to study the problems of sequentiality (which yielded Definition 4.2 of well-formed global type) and of coherence (which is embodied by the subsession relation whose algorithmic counterpart is the $\between$ merge operator) in the simplest setting without further complexity induced by extra features. As a consequence of this choice, our merge between session types is a generalization of the merge in [21, 12] since we allow inputs from different senders (this is the reason why our compatibility is more demanding than the corresponding notion in [21]).

Another feature we disregarded in the present work is *delegation*. This was introduced in [16] for multi-party sessions and is directly inherited from that of dyadic sessions [15]. A participant can delegate another agent to play his role in a session. This delegation is transparent for all the remaining participant of the session. Delegation is implemented by exchanging channels, *ie*, by allowing higher-order channels. In this way the topology of communications may dynamically evolve.

Our crusade for simplification did not restrict itself to exclude features that seemed inessential or too syntax dependent, but it also used simpler forms of existing constructs. In particular an important design choice was to use Kleene star instead of more expressive recursive global types used in [15, 9, 16, 12]. Replacing the star for the recursion gives us a fair implementation of the projected specification almost for free. Fairness seems to us an important —though neglected by current literature— requirement for multi-party sessions. Without it a session in which a part continuously interacts leaving a second one to starve is perfectly acceptable. This is what happens in all the papers referred in this subsection. Without Kleene star, fairness would be more difficult to en-

force. Clearly recursion is more expressive than iteration, even though we can partially bridge this gap using k-exit iterations (Section 6).

Finally, although we aimed at simplifying as much as possible, we still imposed few restrictions that seemed unavoidable. Foremost, the sequentiality condition of Section 4, that is, that any two actions that are bound by a semicolon must always appear in the same order in all traces of (sound and complete) implementations. Surprisingly, in all current literature of multi-party session types we are aware of, just one work [9] enforces the sequential semantics of "$;$". In [9] the sequentiality condition, called *connectedness* is introduced (albeit in a simplified setting since—as in [15, 16]— instead of the "$;$" the authors consider the simpler case of prefixed actions) and identified as one of three basic principles for global descriptions under which a sound and complete implementation can be defined. All others (even later) works admit to project, say, $\mathtt{q} \xrightarrow{a} \mathtt{p}; \mathtt{r} \xrightarrow{a} \mathtt{p}$ in implementations in which $\mathtt{p}$ reads from $\mathtt{r}$ before having read from $\mathtt{q}$. While the technical interest of relaxing the sequentiality constraint in the interpretation of the "$;$" operator is clear —it greatly simplifies projectability— we really cannot see any semantically plausible reason to do it.

Of course all this effort of simplification is worth only if it brings clear advantages. First and foremost, our simpler setting allows us to give a semantic justification of the formalism and of the restrictions and the operators we introduced in it. For these reasons many restrictions that are present in other formalisms are pointless in our framework. For instance, two global types whose actions can be interleaved in an arbitrary way (*ie*, composed by $\wedge$ in our calculus) can share common participants in our global types, while in the work of [9] and [16] (which use the parallel operator for $\wedge$) this is forbidden. So these works fail to project (actually, they reject) protocols as simple as the first line of the example given in the specification (1) in the introduction. Likewise we can have different receivers in a choice like, for example, the case in which two associated buyers wait for a price from a given seller:

$$\mathtt{seller} \xrightarrow{price} \mathtt{buyer1}; \mathtt{buyer1} \xrightarrow{price} \mathtt{buyer2} \vee \mathtt{seller} \xrightarrow{price} \mathtt{buyer2}; \mathtt{buyer2} \xrightarrow{price} \mathtt{buyer1}$$

while such a situation is forbidden in [9, 16].

Another situation possible in our setting but forbidden in [9, 16, 12] is to have different sets of participants for alternatives, such as in the following case where a buyer is notified about a price by the broker or directly by the seller, but in both cases gives an answer to the broker:

$$(\mathtt{seller} \xrightarrow{agency} \mathtt{broker}; \mathtt{broker} \xrightarrow{price} \mathtt{buyer} \vee \mathtt{seller} \xrightarrow{price} \mathtt{buyer}); \mathtt{buyer} \xrightarrow{answer} \mathtt{broker}$$

A similar situation may arise when choosing between repeating or exiting a loop:

$$\begin{array}{l} \mathtt{seller} \xrightarrow{agency} \mathtt{broker}; (\mathtt{broker} \xrightarrow{offer} \mathtt{buyer}; \mathtt{buyer} \xrightarrow{counteroffer} \mathtt{broker})^*; \\ (\mathtt{broker} \xrightarrow{result} \mathtt{seller} \wedge \mathtt{broker} \xrightarrow{result} \mathtt{buyer}) \end{array}$$

which is again forbidden in [9, 16, 12].

The fact of focusing on a core calculus did not stop us from experimenting. On the contrary, having core definitions for global and session types allowed us to explore new

linguistic and communication primitives. In particular, an original contribution of our work is the addition of actions with multiple senders and encoded multiple receivers (as explained at the beginning of Section 2). This allows us to express both joins and forks of interactions as shown by the specification (3) given in Section 2.

Choreographies. Global types can be seen as choreographies [1] describing the interaction of some distributed processes connected through a private multiparty session. Therefore, there is a close relationship between our work and [6, 7, 5], which concern the projection of choreographies into the contracts of their participants. The choreography language in these works essentially coincides with our language of global types and, just like in our case, a choreography is correct if it preserves the possibility to reach a state where all of the involved Web services have successfully terminated. There are some relevant differences though, starting from choreographic interactions that invariably involve exactly one sender and one receiver, while in the present work we allow for multiple senders and we show that this strictly improves the expressiveness of the formalism, which is thus capable of specifying the join of independent activities. Other differences concern the communication model and the projection procedure. In particular, the communication model is synchronous in [6] and based on FIFO buffers associated with each participant of a choreography in [7]. In our model (Section 3) we have a single buffer and we add the possibility for a receiver to specify the participant from which a message is expected. In [6, 7, 5] the projection procedure is basically an homomorphism from choreographies to the behavior of their participants, which is described by a contract language equipped with parallel composition, while our session types are purely sequential. [6, 7] give no conditions to establish which choreographies produce correct projects. In contrast, [5] defines three *connectedness conditions* that guarantee correctness of the projection. The interesting aspect is that these conditions are solely stated on properties of the set of traces of the choreography, while we need the combination of projectability (Table 3) and well-formedness (Definition 4.2). However, the connectedness conditions in [5] impose stricter constraints on alternative choreographies by requiring that the roles in both branches be the same. This forbids the definition of the two global types described just above that involve the `broker` participant. In addition, they need a causal dependency between actions involving the same operation which immediately prevents the projection of recursive behaviors (the choreography language in [5] lacks iteration and thus can only express finite interactions).

Finally, in discussing MSG in the long version of this work we argue that requiring the specification and its projection produce the same set of traces (called *standard implementation* in [14]) seemed overly constraining and advocated a more flexible solution such as the definitions of soundness and completeness introduced here. However it is interesting that Bravetti and Zavattaro in [5] take the opposite viewpoint, and make this relation even more constraining by requiring the relation between a choreography and its projection to be a strong bisimulation.

Other calculi. In this brief overview we focused on works that study the relation between global specifications and local machine-oriented implementations. However in the literature there is an important effort to devise new description paradigms for ei-

ther global descriptions or local descriptions. In the latter category we want to cite [15, 4], while [10] seems a natural candidate in which to project an eventual higher order extension of our global types. For what concerns global descriptions, the Conversation Calculus [8] stands out for the originality of its approach.

8 Conclusion

We think that the design-by-contract approach advocated in [9, 16] and expanded in later works is a very reasonable way to implement distributed systems that are correct by construction. In this work we have presented a theory of global types in an attempt of better understanding their properties and their relationship with multi-party session types. We summarize the results of our investigations in the remaining few lines. First of all, we have defined a proper algebra of global types whose operators have a clear meaning. In particular, we distinguish between sequential composition, which models a strictly sequential execution of interactions, and unconstrained composition, which allows the designer to underspecify the order of possibly dependent interactions. The semantics of gobal types is expressed in terms of regular languages. Aside from providing an accessible intuition on the behavior of the system being specified, the most significant consequence is to induce a *fair* theory of multi-party session types where correct sessions preserve the ability to reach a state in which all the participants have successfully terminated. This property is stronger than the usual progress property within the same session that is guaranteed in other works. We claim that eventual termination is both desirable in practice and also technically convenient, because it allows us to easily express the fact that every participant of a session makes progress (this is non-trivial, especially in an asynchronous setting). We have defined two projection methods from global to session types, a semantic and an algorithmic one. The former allows us to reason about *which* are the global types that can be projected, the latter about *how* these types are projected. This allowed us to define three classes of flawed global types and to suggest if and how they can be amended. Most notably, we have characterized the absence of sequentiality solely in terms of the traces of global types, while we have not been able to provide similar trace-based characterizations for the other flaws. Finally, we have defined a notion of completeness relating a global type and its implementation which is original to the best of our knowledge. In other theories we are aware of, this property is either completely neglected or it is stricter, by requiring the equivalence between the traces of the global type and those of the corresponding implementation.

Acknowledgments. We are indebted to several people from the LIAFA lab: Ahmed Bouajjani introduced us to Parikh's equivalence, Olivier Carton explained us subtle aspects of the shuffle operator, Mihaela Sighireanu pointed us several references to global specification formalisms, while Wiesław Zielonka helped us with references on trace semantics. Anca Muscholl helped us on surveying MSC and Martín Abadi and Roberto Amadio with the literature on security protocols (see the long version). Finally, Roberto Bruni gave us serveral useful suggestions to improve the final version of this work. This work was partially supported by the ANR Codex project, by the MIUR Project IPODS, by a visiting researcher grant of the "Fondation Sciences Mathématiques de Paris", and by a visiting professor position of the Université Paris Diderot.

References

1. Web services choreography description language version 1.0. W3C Candidate Recommendation, available at `http://www.w3.org/TR/ws-cdl-10/`, 2005.
2. J. A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration. Technical Report Report CS-R9314, Programming Research Group, University of Amsterdam, 1993.
3. L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *Proceedings of CONCUR'08*, LNCS 5201, pages 418–433. Springer, 2008.
4. M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *Proceedings of FMOODS'08*, LNCS 5051, pages 19–38. Springer, 2008.
5. M. Bravetti, I. Lanese, and G. Zavattaro. Contract-driven implementation of choreographies. In *Proceedings of TGC'09*, LNCS 5474, pages 1–18. Springer, 2008.
6. M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Proceedings of SC'07*, LNCS 4829, pages 34–50. Springer, 2007.
7. M. Bravetti and G. Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *Proceedings of WS-FM'08*, LNCS 5387, pages 37–54. Springer, 2008.
8. L. Caires and H. T. Vieira. Conversation types. In *Proceedings of ESOP'09*, LNCS 5502, pages 285–300. Springer, 2009.
9. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *Proceedings of ESOP'07*, LNCS 4421, pages 2–17. Springer, 2007.
10. G. Castagna and L. Padovani. Contracts for mobile processes. In *Proceedings of CONCUR'09*, LNCS 5710, pages 211–228. Springer, 2009.
11. C. Chao and Q. Zongyan. An approach to check choreography with channel passing in WS-CDL. In *Proceedings of ICWS'08*, pages 700–707. IEEE Computer Society, 2008.
12. P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *Proceedings of POPL'11*, pages 435–446, 2011.
13. M. Dezani-Ciancaglini, U. de' Liguoro, and N. Yoshida. On progress for structured communications. In *Proceedings of TGC'07*, LNCS 4912, pages 257–275. Springer, 2008.
14. B. Genest, A. Muscholl, and D. Peled. Message sequence charts. In *Lectures on Concurrency and Petri Nets*, LNCS 3098, pages 537–558, 2003.
15. K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proceedings of ESOP'98*, LNCS 1381, pages 22–138. Springer, 1998.
16. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proceedings of POPL'08*, pages 273–284. ACM, 2008.
17. R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28(3):439–466, 1984.
18. R. J. Parikh. On context-free languages. *Journal of the Association for Computing Machinery*, 13(4):570–581, 1966.
19. Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *Proceedings WWW'07*, pages 973–982. ACM, 2007.
20. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *Proceedings of PARLE'94*, LNCS 817, pages 398–413. Springer, 1994.
21. N. Yoshida, P.-M. Deniélou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *Proceedings of FOSSACS'10*, LNCS 6014, pages 128–145, 2010.