A Simple Introduction To Multiparty Sessions

Franco Barbanera
Dipartimento di Matematica e Informatica, Università di Catania, Catania, Italy
franco.barbanera@unict.it
Ugo de' Liguoro
Dipartimento di Informatica, Università di Torino, Torino, Italy
ugo.deliguoro@unito.it

1 Introduction

In programming languages, *types* have been introduced as a sort of *partial specification* of program "behaviours". Coming from type theory, a branch of mathematical logic, types and type systems are used to declare the domain of program variables and the range of functions and procedures, defining constraints that can be effectively checked at compile-time. This is particularly evident in the case of functional programming languages like Haskell [1], whose type system can derive the type `Int->Bool` for, say, the program `odd` while checking that this is consistent with the actual code of the program. This typing serves two goals: first, any application of `odd` to a string, boolean, or any argument that does not qualify as an integer will not type check, as well as any use of, say, `odd`(7) in a context expecting a value that is not a boolean will be rejected by the system after the static analysis of the code, so preventing run-time errors. Second, the semantic piece of information that types convey is useful as a structuring principle for the programmer, helping, e.g. in the design of complex classes in object-oriented languages.

Type systems have been primarily used for sequential programs, whose behaviour can be described in functional terms as input-output relations. In this setting, types are computational invariants of the processes that compute function values. Exploiting type systems in the case of concurrent programs is more challenging. This is mainly because the semantics of the latter are essentially about the interaction among communicating processes, which are non-functional and can hardly be captured by the types of the exchanged messages only.

A crucial step toward using types to reason about concurrent programs has been the development of Milner's π-calculus [12, 13]. In the π-calculus, channels are themselves messages that can be passed around so that types attached to channels can be more expressive. Indeed, type systems inspired by typed dialects of the π-calculus not only enforce error-freeness properties similar to those mentioned above in the sequential case, but also may express (to a certain extent) properties like deadlock-freeness (the concurrent program cannot get stuck) and lock-freedom (no component of the program can get stuck).

Among such developments, a remarkable one was the introduction of the notion of *Session Type* [9]. A session type can be looked at as the description of an "interaction protocol" from the point of view of the single participants interacting on a two-sided communication channel. More precisely, it is a structured set of constraints that a process has to respect when interacting with a partner through its endpoint of a channel. Let us assume that we have a client connected to a server via a channel c. If the server can be used, say, to check the equality of two integers, the following type for the client's endpoint

To appear in EPTCS.

of c describes the correct sequence of the client's actions (where '?' and '!' stand, respectively, for the input and output action), and are followed by the sort of communicated value.

$$\texttt{!Int.!Int.?Bool}$$

Dually, the server has to behave on its endpoint of c as described by

$$\texttt{?Int.?Int.!Bool}$$

Roughly, type systems based on session types do ensure (a) that a channel is used exactly by two participants; (b) that the types associated to the two endpoints of the channel are "dual"; (c) that the participants' interactions on their respective sides of the channel respect what is prescribed by their type. As happens in general with types, session types provide only a partial description of the concurrent computation carried on by the client and the server, but it is nonetheless a remarkable piece of information in case there was an automatic or semi-automatic way to ensure that the exchange of data between them does conform with what the types describe. It is worth recalling that session types also manage to describe branching behaviours, similarly to what we shall see later on. The "binary" specification is used to distinguish them from their *multiparty* version: our main topic here, which we now begin to address in the sequel to this introduction.

As hinted above, in binary session types, an interaction protocol between two participants is described in terms of the communication behaviours of the single participants on their respective endpoints of the channel connecting them. The "coherence" of the two separate behaviours concerning the implicitly assumed overall protocol, is guaranteed by the notion of *duality*. It is natural, however, to try and provide also explicit descriptions of protocols from a *global* point of view, in contrast with the *local* ones previously considered. The global protocol for our simple client-server example, by assuming $\mathtt{p}$ to be the name of the client and $\mathtt{q}$ that of the server, could be represented by the expression

$$\mathtt{p} \to \mathtt{q}\texttt{:Int.}\,\mathtt{p} \to \mathtt{q}\texttt{:Int.}\,\mathtt{p} \to \mathtt{q}\texttt{:Bool}$$

This linear sequence of interactions is sufficient in case the server $\mathtt{q}$ just computes the equality predicate on integers. What, however, if $\mathtt{q}$ were able to compute both the equality predicate and the factorial function? In case $\mathtt{p}$ wished to safely use both the above $\mathtt{q}$'s features, the interaction protocol should contain a branch, depending on $\mathtt{p}$'s needs. How can $\mathtt{p}$ communicate to $\mathtt{q}$ its intention to pursue one branch of the protocol rather than the other one? To this aim, one can introduce a particular sort of data – whose element we call "labels" – as a tool allowing selection among protocol branches. The protocol driving the interactions between $\mathtt{p}$ and $\mathtt{q}$ is now

$$\mathtt{p} \to \mathtt{q} \left\{ \begin{array}{l} \mathit{equal} : \mathtt{p} \to \mathtt{q}\texttt{:Int.}\,\mathtt{p} \to \mathtt{q} : \texttt{Int.}\,\mathtt{q} \to \mathtt{p}\texttt{:Bool.End} \\ \mathit{fact} : \mathtt{p} \to \mathtt{q} : \texttt{Int.}\,\mathtt{q} \to \mathtt{p}\texttt{:Int.End} \end{array} \right.$$

In actual scenarios, concurrent systems can have more than two participants. Moreover, it is reasonable to expect a process to have access to more than one channel, to interact with many other processes. Even if type systems with binary session types can control and provide guarantees on what happens on the single channels in isolation, it is not possible, however, to control the overall behaviour of several interacting processes. The actions of a process can be interwoven among the several sessions represented by private channels, making it hard, if not impossible, to predict the overall behaviour of a system from what happens in a single session and its local channels. *MultiParty Session Types* [10], MPST,

have been introduced to describe and control the communication behaviour of concurrent systems where several participants may intervene.

Type systems with multiparty session types are essentially particular *choreographic* formalisms. These sorts of formalisms are characterised by the coexistence of two different views of concurrent systems: the *global* and the *local* views. The former is a description of the overall behaviour of the system interconnecting several participants, whereas the latter is the description of the behaviours of the single participants. MPST formalisms use particular global views called *global types*, whereas the local views are "two-layered": a layer of *processes* and a layer of *local types*. The latter can be looked at as an abstract description of the former. In contrast, in the present chapter, we treat a particular multiparty-session formalism, dubbed *Simple MultiParty Sessions* [4, 2], SMPS. In such a formalism the local view is "single-layered" and the relationship between the global and the local views is approached from a somewhat orthogonal perspective than in MPST. We also propose a simplified version of standard MPST, enabling us to clarify their relationship with SMPS. Many notions of concurrency theory, when needed, are introduced in a simplified way to make the discussion as self-contained as possible.

2 Global Views as Global Types

Unlike binary session types, global types describe interaction protocols from an outer perspective. Moreover, such protocols take into account an arbitrary, though finite, number of participants. As a simple example, we now consider a simplified version of the classical *Travel Agency* protocol.

`Travel Agency`

> The system consists of three participants: c, the customer; a, the travel agency; and s, the travel service. The customer sends to the agency a string specifying where she would like to go on vacation. The agency then replies with a number (the price of the trip), according to which the client decides either to accept the offer or to reject it. In the first case the client sends a string specifying the date of his vacation to the service affiliated to the agency, whereas in the second case she communicates a number representing the fact that she did not accept the agency's offer and at the same time quantifying the quality of the service notwithstanding the offer refusal.

In the style of the examples in the Introduction, this protocol is described by the following expression:

$$\mathsf{c} \to \mathsf{a}{:}\mathtt{String}.\,\mathsf{a} \to \mathsf{c}{:}\mathtt{Int}.\,\mathsf{c} \to \mathsf{a} \begin{cases} \mathit{accept}.\,\mathsf{c} \to \mathsf{s}{:}\mathtt{String}.\,\mathtt{End} \\ \mathit{reject}.\,\mathsf{c} \to \mathsf{s}{:}\mathtt{Int}.\,\mathtt{End} \end{cases}$$

To obtain a more intelligible version of a protocol, we may decide not to distinguish among sequential and branching flow of control and use labels also when there are no branches:

$$\mathsf{c} \to \mathsf{a}{:}\mathit{req}\langle\mathtt{String}\rangle.\,\mathsf{a} \to \mathsf{c}{:}\mathit{price}\langle\mathtt{Int}\rangle.\,\mathsf{c} \to \mathsf{a} \begin{cases} \mathit{accept}.\,\mathsf{c} \to \mathsf{s}{:}\mathit{date}\langle\mathtt{String}\rangle.\mathtt{End} \\ \mathit{reject}.\,\mathsf{c} \to \mathsf{s}{:}\mathit{nodeal}\langle\mathtt{Int}\rangle.\mathtt{End} \end{cases}$$

On the other hand, if we focus on the structure of the interaction, we may disregard the (ground) types of the labels and keep just the latter, which we shall refer to as *messages*:

$$\mathsf{c}{\to}\mathsf{a}{:}\mathit{req}.\,\mathsf{a}{\to}\mathsf{c}{:}\mathit{price}.\,\mathsf{c}{\to}\mathsf{a} \begin{cases} \mathit{accept}.\,\mathsf{c}{\to}\mathsf{s}{:}\mathit{date}.\mathsf{End} \\ \mathit{reject}.\,\mathsf{c}{\to}\mathsf{s}{:}\mathit{nodeal}.\mathsf{End} \end{cases} \tag{1}$$

Abstracting from the message types strengthens the uniformity we intend to pursue, since now there is no difference between labels like *accept* and *reject*, used for denoting different branches in a protocol and labels like *price*, used for naming data items.

It is worth remarking that usually, the representation of a protocol is independent of the synchronization model (synchronous or asynchronous) of the communications among the participants related to the protocol. We shall briefly discuss synchronous and asynchronous communication models later on in Section 3.

Infinite protocols It is natural to expect concurrent systems to have unbounded behaviours of potentially infinite length. In the above example, c might ask for the price of another destination in case the price is not accepted. Hence a should be able to answer an unpredictable number of c's requests. As in actual client/server systems, c should also explicitly communicate to a her intention to quit the interaction. The protocol should hence look as follows:

$$\mathtt{c}\to\mathtt{a}{:}\mathit{req}.\,\mathtt{a}\to\mathtt{c}{:}\mathit{price}.\,\mathtt{c}\to\mathtt{a}\begin{cases}\mathit{accept}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{date}.\mathsf{End}\\ \mathit{reject}.\,\mathtt{c}\to\mathtt{a}{:}\mathit{req}.\,\mathtt{a}\to\mathtt{c}{:}\mathit{price}.\,\mathtt{c}\to\mathtt{a}\begin{cases}\mathit{accept}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{date}.\mathsf{End}\\ \mathit{reject}.\,\mathtt{c}\to\mathtt{a}{:}\mathit{req}\ldots\mathtt{etc.}\\ \mathit{quit}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{nodeal}.\mathsf{End}\end{cases}\\ \mathit{quit}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{nodeal}.\mathsf{End}\end{cases}$$

This protocol looks like a tree with an infinite branch (the one representing the case of c keeping on infinitely rejecting offers and making new requests). Of course infinite trees[1] makes perfect sense from a mathematical point of view. In actual implementations of concurrent systems what we can handle, however, are only finite representations of them. Such finite representations can easily be devised in cases such as the one above, since the tree is *regular*. A regular tree is a possibly infinite tree with only a finite number of subtrees. There is a plethora of formalisms for providing finite representations of regular trees. Our example can be represented as a recursive expression, like

$$\mu\mathtt{t}.\,\mathtt{c}\to\mathtt{a}{:}\mathit{req}.\,\mathtt{a}\to\mathtt{c}{:}\mathit{price}.\,\mathtt{c}\to\mathtt{a}\begin{cases}\mathit{accept}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{date}.\mathsf{End}\\ \mathit{reject}.\,\mathtt{t}\\ \mathit{quit}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{nodeal}.\mathsf{End}\end{cases}$$

or as the solution of the following recursive equation,

$$\mathsf{G} = \mathtt{c}\to\mathtt{a}{:}\mathit{req}.\,\mathtt{a}\to\mathtt{c}{:}\mathit{price}.\,\mathtt{c}\to\mathtt{a}\begin{cases}\mathit{accept}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{date}.\mathsf{End}\\ \mathit{reject}.\,\mathsf{G}\\ \mathit{quit}.\,\mathtt{c}\to\mathtt{s}{:}\mathit{nodeal}.\mathsf{End}\end{cases}$$

or also by using an automaton.

Since we are interested in a theoretical investigation of the relationship between *global* and *local* views of systems of concurrent processes, we choose to look at protocols in the most abstract way, so dealing directly with them as possibly infinite objects. What we need now is a formal mathematical tool for defining them and proving their properties.

[1] A tree can be infinite either because it possesses branches of infinite length, or because it has nodes with an infinite number of children. In our setting we assume the number of possible messages to be finite, so we do consider the former possibility.

Coinduction The infinite objects we need to handle are essentially infinite trees, used for the representation of protocols, and infinite lists (called traces), used for the representation of possible evolutions of multiparty sessions (parallel compositions of abstract communicating processes).

COMMUNICATIONS

Let us consider triples of the form $(\mathsf{p},\lambda,\mathsf{q})$, where p and q are participants' names and λ is a message, that we represent as $\mathsf{p}\lambda\mathsf{q}$ for short. We use these triples to describe the exchange of message λ from participant p to participant q and call them *communications* belonging to the set $\mathscr{C}$. On these, we impose the natural condition $\mathsf{p} \neq \mathsf{q}$. Let us now assume that in a system we can observe only *events* that are communications, i.e. we abstract away anything but message exchanges since we are interested in investigating only the communication properties of systems. Let us also assume that we can observe only one event at a time. So that, even if two events happens simultaneously, we can look at them as if they happened in sequence. The description of what happens in a concurrent system through such sequences of observations is usually called *interleaving* operational semantics[2].

TRACES

A possible sequence of communication events is called a *trace*. The fact that a system can follow an infinite protocol entails that we have to consider also infinite traces, that is infinite sequences of communications. As far as traces are concerned, such an informal definition of trace suffices. But what in the case of more structured infinite objects? We can recur to formal *coinductive definitions*. These come in several possible forms. We briefly sketch coinductive definitions via formal systems, a method called *rule coinduction*[3]. The set **Tr** of traces can hence be formally defined by

$$\frac{}{\varepsilon \in \mathbf{Tr}} \qquad \frac{\mathsf{p}\lambda\mathsf{q} \in \mathscr{C} \quad t \in \mathbf{Tr}}{\mathsf{p}\lambda\mathsf{q} \cdot t \in \mathbf{Tr}}$$

where "$\cdot$" is the sequence concatenation operator.
These rules can be read in two ways: top-down or bottom-up, or more precisely: inductively or coinductively.

The **inductive** reading goes forward from the premises to the conclusion; in the case of **Tr** the above rules specify how its elements are constructed starting with ε (representing the empty sequence), which belongs to **Tr** unconditionally since the first rule has no premises (it is an *axiom*). The second rule states that if we have previously constructed some $t \in \mathbf{Tr}$, then by choosing any $\mathsf{p}\lambda\mathsf{q} \in \mathscr{C}$ we can construct the new trace $\mathsf{p}\lambda\mathsf{q} \cdot t$ and conclude that $\mathsf{p}\lambda\mathsf{q} \cdot t \in \mathbf{Tr}$. In any case, the statement $t \in \mathbf{Tr}$ labels the root of a derivation which is a finite tree, and indeed the trace t itself is finite. By saying that **Tr** is *inductively* defined by the above rules, we are defining **Tr** as the set of all finite sequences of communications.

The **coinductive** reading of these rules proceeds backward from the conclusion to the premises, specifying what we can observe of a trace in one step. This becomes apparent with the second rule, which now tells that what we can observe of a trace $\mathsf{p}\lambda\mathsf{q} \cdot t \in \mathbf{Tr}$, different than ε, is that it begins with a communication $\mathsf{p}\lambda\mathsf{q} \in \mathscr{C}$ and continues with some $t \in \mathbf{Tr}$ that can be observed using at least one of the same rules. Since, in particular, there is no bound to the number of times in which we can use backward the second rule above, the derivation of a statement $t \in \mathbf{Tr}$ can be infinite, in which case the trace t is infinite as well. Therefore, by saying that **Tr** is *coinductively* defined by the above rules, we are formally defining **Tr** as the set of all finite and infinite sequences of communications.

[2] A concurrent process can be given other kinds of semantics. An instance is the *true concurrency* semantics, where several events can be observed simultaneously.

[3] Rule coinduction is a particular case of definition via the greatest fixed point of a suitable functor; for uniformity, we shall use rule coinduction only, even when it would be more natural to use the general principle. The interested reader is referred to [14, Ch.2] and the literature cited there.

As a notational convention, rules that we wish to be coinductively interpreted, are represented with double lines, like

$$\frac{}{\varepsilon \in \mathbf{Tr}} \qquad \frac{\mathsf{p}\lambda\mathsf{q} \in \mathscr{C} \quad t \in \mathbf{Tr}}{\mathsf{p}\lambda\mathsf{q}\cdot t \in \mathbf{Tr}}$$

Rules can also be presented in a form that is more familiar to computer scientists, such as productions of a Chomsky grammar. In particular, the above rules can be presented by

$$\mathbf{Tr} ::= \varepsilon \mid \mathsf{p}\lambda\mathsf{q}\cdot\mathbf{Tr} \qquad \text{or} \qquad \mathbf{Tr} ::=^{coind} \varepsilon \mid \mathsf{p}\lambda\mathsf{q}\cdot\mathbf{Tr}$$

according to whether the inductive or the coinductive interpretation is intended. We shall later provide the meaning of global types in terms of coinductively defined traces.

Given a system of communicating processes (a notion that we shall make formal later on), we can say that their interleaving semantics is a subset of the coinductively defined set **Tr**. Let us now precisely define the set of global types, namely the formalisation of our "protocols". We assume to have the following denumerable sets: *messages* (ranged over by $\lambda, \lambda', \lambda_1, \ldots$), *participant names* (ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s} \ldots$) and *indexes* (ranged over by i, j, l). Besides, we use $I, J, \ldots$ to range over finite sets of indices.

Definition 2.1 (Global types) Global types *are defined by:*

$$\mathsf{G} ::=^{coind} \mathsf{End} \mid \mathsf{p}\to\mathsf{q}:\{\lambda_i.\mathsf{G}_i\}_{i\in I}$$

where $I \neq \emptyset$ and $\lambda_i \neq \lambda_j$ for $i, j \in I$ and $i \neq j$. We restrict the set of global types to the regular ones, and we call $\mathscr{G}$ the set of global types.

We are not considering specific indices. We could use, for instance, natural numbers as indices. In that case, if we had $I = \{1, 2, \ldots, n\}$, $\mathsf{p}\to\mathsf{q}:\{\lambda_i.\mathsf{G}_i\}_{i\in I}$ would stand for the expression $\mathsf{p}\to\mathsf{q}:\{\lambda_1.\mathsf{G}_1, \lambda_2.\mathsf{G}_2, \ldots, \lambda_n.\mathsf{G}_n\}$. We use the set notation to express the fact that the order of the elements inside the curly bracket is inessential, i.e.

$$\mathsf{p}\to\mathsf{q}:\{cucu.\mathsf{End}, settete.\mathsf{End}\} \quad \text{and} \quad \mathsf{p}\to\mathsf{q}:\{settete.\mathsf{End}, cucu.\mathsf{End}\}$$

have to be considered as the *very same* global type. The symbol End is used to denote the protocol that does not require anything. We shall omit writing trailing End's in global types whenever the reading is so enhanced, and denote $\mathsf{p}\to\mathsf{q}:\{\lambda.\mathsf{G}\}$ by $\mathsf{p}\to\mathsf{q}:\lambda.\mathsf{G}$.

Definition 2.2 (Participants) *Given $\mathsf{G} \in \mathscr{G}$ we define the set of* participants *of* G*, written* $\mathsf{prt}(\mathsf{G})$ *by the coinductive rules deriving judgments of the form $\mathsf{p} \in \mathsf{prt}(\mathsf{G})$:*

$$\frac{\mathsf{p} \in \{\mathsf{r}, \mathsf{s}\}}{\mathsf{p} \in \mathsf{prt}(\mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i\}_{i\in I})} \qquad \frac{k \in I \quad \mathsf{p} \in \mathsf{prt}(\mathsf{G}_k)}{\mathsf{p} \in \mathsf{prt}(\mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i\}_{i\in I})}$$

We define $\mathsf{prt}(\mathsf{G})$ *as the set of all* p *such that* $\mathsf{p} \in \mathsf{prt}(\mathsf{G})$ *is derivable.*

Exercise 1 *Show that* $\mathsf{prt}(\mathsf{G})$ *is finite for any* $\mathsf{G} \in \mathscr{G}$ *(hint: use the fact that* G *is regular).*

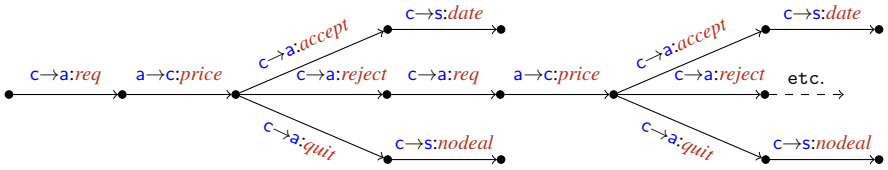
Correctness and Completeness

Let us now try and better understand what we meant when we wrote that a protocol (i.e. a global type) is a structured set of constraints that some given processes are expected to respect during their interaction.

- A global type is a **correct** description of a system behaviour when it represents *at most* all the traces of the semantics of the system.
- A global type is a **complete** description of a system behaviour when it represents *at least* all the traces of the semantics of the system.

In choreographic formalisms, one usually aims at having a global view to be a correct and complete description of the overall behaviour of the corresponding local views.

What traces does a global type represent? A natural answer to this question is to take the sequences of messages corresponding to the branches of the (finite or infinite) syntactic tree of the type. In the case of the `Travel Agency` example, such a tree is the following:



The strings corresponding to all the finite branches above can be described via the following regular expression.

$$(c\mathit{req}\mathsf{a}\cdot a\mathit{price}\mathsf{c})(c\mathit{reject}\mathsf{a}\cdot c\mathit{req}\mathsf{a}\cdot a\mathit{price}\mathsf{c})^*(c\mathit{accept}\mathsf{a}\cdot c\mathit{date}\mathsf{s}\cdot\mathsf{End}+c\mathit{quit}\mathsf{a}\cdot c\mathit{nodeal}\mathsf{s}\cdot\mathsf{End})$$

To these we have to add the infinite string $(c\mathit{req}\mathsf{a}\cdot a\mathit{price}\mathsf{c}\cdot c\mathit{reject}\mathsf{a})^{\omega}$ corresponding to the only infinite branch of this tree[4].

However, this interpretation seems unreasonable in general. Indeed, while in the sequence

$$c\mathit{req}\mathsf{a}\cdot a\mathit{price}\mathsf{c}\cdot c\mathit{accept}\mathsf{a}\cdot c\mathit{date}\mathsf{s}\cdot\mathsf{End}$$

we can easily recognize that there is a causal dependency among the interactions (as each of them – but the first – involves a participant present in the previous one), this is not the case, for example, in the following global type:

$$\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{r}\to\mathsf{s}{:}\lambda'.\mathsf{End}$$

In fact, under the given interpretation of types, the single trace

$$\mathsf{p}\lambda\mathsf{q}\cdot\mathsf{r}\lambda'\mathsf{s}$$

would not be a correct description of the behaviour of a concurrent system formed by the interacting participants p, q, r, s such that p and q exchange the message λ while r and s the message λ', because in a decentralized system, as it is, nothing might force any of these interactions to happen before the other one.

As shown by the examples we have just seen, the issue is not fixed by postulating that the operator "$\cdot$" on traces is commutative, as this is not the case when there are causal dependencies. Rather, we should consider "$\cdot$" as representing a sequentialisation constraint only for communications sharing at least one participant. Let us consider again $\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{r}\to\mathsf{s}{:}\lambda'.\mathsf{End}$. Apart from the order of the communications, such a global type tells of a system where p sends λ to q and r sends λ' to s. Such communications are clearly independent from each other. So the sequence $\mathsf{p}\lambda\mathsf{q}\cdot\mathsf{r}\lambda'\mathsf{s}$ just represents one possible interleaving of them. How can we make also the sequence $\mathsf{r}\lambda'\mathsf{s}\cdot\mathsf{p}\lambda\mathsf{q}$ be included in the meaning of $\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{r}\to\mathsf{s}{:}\lambda'.\mathsf{End}$?

Labelled Transition Systems

The technical tool for achieving our goal, which is common in the theory of concurrent processes, is the notion of *labelled transition systems* (LTS). An LTS over a set X of states, and with respect to a set Act of observable events, often called *actions*, is a ternary relation $\to\subseteq X\times Act\times X$; writing $x\xrightarrow{a}y$ for

[4] The writing s^{ω} is that of denumerably many copies of s sequentially concatenated.

$(x,a,y) \in \rightarrow$, we mean that a system in the state x may perform the action a entering a new state y. This can be extended to finite sequences $a_1 \cdots a_n \in Act^*$ by setting

$$x \xrightarrow{a_1 \cdots a_n} y \qquad \text{if} \qquad \exists x_1, \ldots, x_{n-1} \in X.\, x \xrightarrow{a_1} x_1 \cdots x_{n-1} \xrightarrow{a_n} y$$

where if $n = 0$ then we assume that $a_1 \cdots a_n$ is the empty sequence ε and $y = x$. We also write $x \rightarrow^* y$ for $x \xrightarrow{s} y$ for some (finite) s.

Exercise 2 *Turn the informal definition of $x \xrightarrow{s} y$ for $s \in Act^*$ into an inductive definition.*

We can associate to each $x \in X$ the set of the finite traces out of it by setting:

$$\mathbf{Tr}^{fin}(x) = \{s \in Act^* \mid \exists y \in X.\, x \xrightarrow{s} y\}$$

To this definition, we have to add something to treat the case of infinite sequences out of x. Let Act^ω be the set of infinite sequences $t = a_1 \cdots a_n \cdots$ of denumerably many actions from Act; then we define $Act^\infty = Act^* \cup Act^\omega$ to be the set of finite and infinite sequences over Act. For any $a \in Act$ and $s \in Act^\infty$ the sequence $a \cdot s \in Act^\infty$ begins with a and continues with s.

Definition 2.3 *Let $(X, Act, \rightarrow)$ be an LTS and $x \in X$, then the set $\mathbf{Tr}(x)$ of* finite and infinite traces out of *x is coinductively defined by the rules*

$$\frac{}{\varepsilon \in \mathbf{Tr}(x)} \qquad \frac{a \in Act \quad x \xrightarrow{a} y \quad s \in \mathbf{Tr}(y)}{a \cdot s \in \mathbf{Tr}(x)}$$

Exercise 3 *Suppose that both $x \xrightarrow{a} y$ and $y \xrightarrow{b} x$ hold; show that both the finite sequence $abab$ and the infinite sequence $abab\cdots$ are in $\mathbf{Tr}(x)$. What if we drop the axiom $\varepsilon \in \mathbf{Tr}(x)$ from Definition 2.3?*

Exercise 4 *We say that the LTS $(X, Act, \rightarrow)$ is* finitely branching *if for all $x \in X$ the set $\{y \in Act \mid \exists a \in Act.\, x \xrightarrow{a} y\}$ is finite.*

Given a finite sequence $s \in Act^*$ and a finite or infinite sequence $t \in Act^\infty$ we say that s is a prefix *of t if $t = s \cdot t'$ for some $t' \in Act^\infty$. Show that if the LTS is finitely branching then*

$$\mathbf{Tr}(x) = \{t \in Act^\infty \mid \forall s \text{ prefix of } t\, \exists y \in X.\, x \xrightarrow{s} y\} \tag{2}$$

Explain why the hypothesis that the LTS is finitely branching is essential for the above to hold (hint: consider the König's Lemma from set theory, which states that a finitary tree – namely, finitely branching – is infinite if and only if it has an infinite branch).

Coming back to the problem of defining the semantics of global types, we set $X = \mathscr{G}$ and $Act = \mathscr{C}$. Then we expect that

$$\mathsf{p}{\rightarrow}\mathsf{q}{:}\lambda.\mathsf{r}{\rightarrow}\mathsf{s}{:}\lambda'.\mathsf{End} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{r}{\rightarrow}\mathsf{s}{:}\lambda'.\mathsf{End}$$

since we wish that $\mathsf{p}\lambda\mathsf{q}$ can be the first communication we observe in a system whose overall behaviour is described by this global type. Nonetheless, we should also have

$$\mathsf{p}{\rightarrow}\mathsf{q}{:}\lambda.\mathsf{r}{\rightarrow}\mathsf{s}{:}\lambda'.\mathsf{End} \xrightarrow{\mathsf{r}\lambda'\mathsf{s}} \mathsf{p}{\rightarrow}\mathsf{q}{:}\lambda.\mathsf{End}$$

since $\mathsf{r}\lambda'\mathsf{s}$ could be observed first, being its participants different from both p and q. This motivates the two rules in the definition below.

Definition 2.4 (Inductive LTS for global types)

$$[\text{G-EComm}]\ \frac{}{\mathsf{p}\to\mathsf{q}:\{\lambda_i.\mathsf{G}_i\}_{i\in I}\xrightarrow{\mathsf{p}\lambda_j\mathsf{q}}\mathsf{G}_j}\ (j\in I)$$

$$[\text{G-IComm}]\ \frac{\mathsf{G}_i\xrightarrow{\mathsf{p}\lambda\mathsf{q}}\mathsf{G}_i'\quad\forall i\in I\qquad\{\mathsf{p},\mathsf{q}\}\cap\{\mathsf{r},\mathsf{s}\}=\emptyset}{\mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i\}_{i\in I}\xrightarrow{\mathsf{p}\lambda\mathsf{q}}\mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i'\}_{i\in I}}$$

We can formally show that $\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{r}\to\mathsf{s}{:}\lambda'.\mathsf{End}\xrightarrow{\mathsf{r}\lambda'\mathsf{s}}\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{End}$ holds. In fact we obtain the following derivation by means of the above rules:

$$\frac{\dfrac{}{\mathsf{r}\to\mathsf{s}{:}\lambda'.\mathsf{End}\xrightarrow{\mathsf{r}\lambda'\mathsf{s}}\mathsf{End}}}{\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{r}\to\mathsf{s}{:}\lambda'.\mathsf{End}\xrightarrow{\mathsf{r}\lambda'\mathsf{s}}\mathsf{p}\to\mathsf{q}{:}\lambda.\mathsf{End}}$$

The interpretation of Rule [G-EComm] is immediate. Rule [G-IComm], instead, specifies that in order to show that the event $\mathsf{p}\lambda\mathsf{q}$ is observable before any event $\mathsf{r}\lambda_i\mathsf{s}$, it should hold that

- the interacting participants $\{\mathsf{p},\mathsf{q}\}$ must be disjoint from the interacting participants $\{\mathsf{r},\mathsf{s}\}$ (i.e. $\{\mathsf{p},\mathsf{q}\}\cap\{\mathsf{r},\mathsf{s}\}=\emptyset$); and
- $\mathsf{p}\lambda\mathsf{q}$ should be observable as first also in case it could be observed as the first action out of any of the G_i, namely $\mathsf{G}_i\xrightarrow{\mathsf{p}\lambda\mathsf{q}}\mathsf{G}_i'$ for all $i\in I$.

Such conditions are indeed a characterization of the fact that $\mathsf{p}\lambda\mathsf{q}$ and all the $\mathsf{r}\lambda_i\mathsf{s}$ are interleaved (as we are assuming in our setting).

Given a sequence $\sigma=\mathsf{p}_1\lambda_1\mathsf{q}_1\cdot\ldots\cdot\mathsf{p}_n\lambda_n\mathsf{q}_n$ of communications, we write $\mathsf{G}\xrightarrow{\sigma}$ whenever there exists $n\geq 1$ and $\mathsf{G}_1,\ldots,\mathsf{G}_n$ such that $\mathsf{G}\xrightarrow{\mathsf{p}_1\lambda_1\mathsf{q}_1}\mathsf{G}_1\xrightarrow{\mathsf{p}_1\lambda_2\mathsf{q}_2}\ldots\xrightarrow{\mathsf{p}_n\lambda_n\mathsf{q}_n}\mathsf{G}_n$. This definition naturally extends to the case of infinite σ as a particular case of (2) as the LTS in Definition 2.4 is finitely branchnig, so obtaining the definition of $\mathbf{Tr}(\mathsf{G})$, i.e. the set of traces of the global type G that we consider as the formal meaning of G.

The definition 2.4 of the LTS for global types is inductive. Even if reasonable, it does not manage to express some communications that should be observable out of some infinite global types. Let us consider the following protocol.

`Love and Greeting`

> Alice (a) and Bob (b) are steadily exchanging the message `love` until, possibly, deciding to issue *bye* In the meantime, Carl (c) is willing to greet Daisy (d) by sending her the message *hello*.

This protocol corresponds to the following global type:

$$\mathsf{G}_{\mathtt{LG}}=\mathsf{a}\to\mathsf{b}:\begin{cases}\mathit{love}.\ \mathsf{G}_{\mathtt{LG}}\\ \mathit{bye}.\ \mathsf{c}\to\mathsf{d}{:}\mathit{hello}\end{cases}$$

According to the discussion above, we expect that the communications c*hello*d and a*love*b are independent events and that Carl and Daisy should be able to greet each other at any moment, that is c*hello*d

should be observable after any number of a*love*b observations. It is however possible to check that, due to the inductive nature of the definition of LTS for global types, it is not even possible to have $\mathsf{G}_{\mathtt{LG}} \xrightarrow{\mathsf{c}\mathit{hello}\mathsf{d}} \mathsf{G}'$, for any G'.

Exercise 5 *Provide a justification of the previous statement, i.e. it is not possible to have* $\mathsf{G}_{\mathtt{LG}} \xrightarrow{\mathsf{c}\mathit{hello}\mathsf{d}} \mathsf{G}'$ *in the LTS of Definition 2.4.* Hint: *we could provide otherwise a* finite *derivation with that conclusion.*

FAIRNESS

What's missing to our LTS? In a sense we would like our LTS to represent a **fair** way of observing a system. Namely, if in a system
- we can observe an event a an infinite number of times and
- there is an event b independent from a and that can occur at any time

then it is not possible to have an infinite sequence of observations made of a only, and b must eventually occur[5]. We wish hence to define an LTS for global types allowing for fair observations. A natural way to get such a definition could simply be

$$\frac{}{\mathsf{p}\to\mathsf{q}:\{\lambda_i.\mathsf{G}_i\}_{i\in I} \xrightarrow{\mathsf{p}\lambda_j\mathsf{q}} \mathsf{G}_j}\ (j\in I) \qquad \frac{\mathsf{G}_i \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}_i' \quad \forall i\in I \qquad \{\mathsf{p},\mathsf{q}\}\cap\{\mathsf{r},\mathsf{s}\}=\emptyset}{\mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i\}_{i\in I} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i'\}_{i\in I}}$$

This however would not work. In fact, besides enabling $\mathsf{G}_{\mathtt{LG}} \xrightarrow{\mathsf{c}\mathit{hello}\mathsf{d}}$, we would also get some unwanted and unreasonable effects for some reasonable global types. For instance, let us consider a protocol where only Alice and Bob are present and in which they keep on exchanging a *love* message for ever. Namely $\mathsf{G}=\mathsf{a}\to\mathsf{b}$:*love*. G. By the above coinductive definition, we could also get

$$\mathsf{G} \xrightarrow{\mathsf{c}\mathit{cucu}\mathsf{d}} \mathsf{G}$$

We have the following infinite derivation made of occurrences of the second rule only, which applies vacuously under the assumption that $\{\mathsf{c},\mathsf{d}\}\cap\{\mathsf{a},\mathsf{b}\}=\emptyset$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \text{etc.}}{\mathsf{G}\xrightarrow{\mathsf{c}\mathit{cucu}\mathsf{d}}\mathsf{G}}}{\mathsf{G}\xrightarrow{\mathsf{c}\mathit{cucu}\mathsf{d}}\mathsf{G}}}{\mathsf{G}\xrightarrow{\mathsf{c}\mathit{cucu}\mathsf{d}}\mathsf{G}}}{\mathsf{G}\xrightarrow{\mathsf{c}\mathit{cucu}\mathsf{d}}\mathsf{G}}$$

Such an infinite derivation can be described more formally and compactly by

$$\mathscr{D} = \dfrac{\mathscr{D}}{\mathsf{G}\xrightarrow{\mathsf{c}\mathit{cucu}\mathsf{d}}\mathsf{G}}$$

Exercise 6 *Would it also be possible to get* $\mathsf{G} \xrightarrow{\mathsf{c}\mathit{love}\mathsf{d}} \mathsf{G}$*?*

[5]Fairness is also a property of *schedulers* of interleaved implementations of concurrent languages enabling concurrent programs to be run on sequential machines.

The above unwanted behaviour occurs because a derivation with coinductive rules may have infinite branches, that in the present case will infinitely defer the proof that the action $\mathsf{c}\mathit{cucu}\mathsf{d}$ can happen even if this will never be the case. We can fix that problem by simply requiring that, when applying a rule, we can deal only with communications explicitly represented in the global type. We formalise this using the notion of *capabilities*, i.e. we coinductively define a set containing all the communications explicitly represented in a global type.

Definition 2.5 (Capabilities) *Let* G *be a type. The set* $\mathsf{cap}(\mathsf{G})$ *of the* capabilities *of* G *is defined via the following rules:*

$$\frac{}{\mathsf{p}\lambda_k\mathsf{q} \in \mathsf{cap}(\mathsf{p}{\to}\mathsf{q}{:}\{\lambda_i.\mathsf{G}_i\}_{i\in I})}\ (k\in I) \qquad \frac{\mathsf{p}\lambda_k\mathsf{q} \in \mathsf{cap}(\mathsf{G}_k)}{\mathsf{p}\lambda_k\mathsf{q} \in \mathsf{cap}(\mathsf{r}{\to}\mathsf{s}{:}\{\lambda_i.\mathsf{G}_i\}_{i\in I})}\ (k\in I)$$

Exercise 7 *Even if Definition 2.5 is coinductive, as* G *may be infinite, the set* $\mathsf{cap}(\mathsf{G})$ *is finite for any* G*. Show this.*

Definition 2.6 (Coinductive LTS for global types)

$$[\text{G-EComm}]\ \frac{}{\mathsf{p}\to\mathsf{q}:\{\lambda_i.\mathsf{G}_i\}_{i\in I} \xrightarrow{\mathsf{p}\lambda_j\mathsf{q}} \mathsf{G}_j}\ (j\in I)$$

$$[\text{G-IComm}]\ \frac{\mathsf{G}_i \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}_i' \quad \forall i\in I \qquad \{\mathsf{p},\mathsf{q}\}\cap\{\mathsf{r},\mathsf{s}\}=\emptyset \qquad \mathsf{p}\lambda\mathsf{q}\in\bigcap_{i\in I}\mathsf{cap}(\mathsf{G}_i)}{\mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i\}_{i\in I} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{r}\to\mathsf{s}:\{\lambda_i.\mathsf{G}_i'\}_{i\in I}}$$

Exercise 8 *Show that* $\mathsf{G}_{\mathtt{LG}} \xrightarrow{\mathsf{c}\mathit{hello}\mathsf{d}} \mathsf{G}'$ *for some* G' *using the coinductive LTS for global types.*

Exercise 9 *Show that*

$$\mathsf{G}_{\mathtt{LG}} = \mathsf{a}{\to}\mathsf{b}{:}\begin{cases} \mathit{love}.\ \mathsf{G}_{\mathtt{LG}} \\ \mathit{bye}.\ \mathsf{c}{\to}\mathsf{d}{:}\mathit{hello} \end{cases} \quad \textit{and}$$

$$\widehat{\mathsf{G}}_{\mathtt{LG}} = \mathsf{c}{\to}\mathsf{d}{:}\mathit{hello}.\widehat{\mathsf{G}}'_{\mathtt{LG}} \quad \textit{where}\ \ \widehat{\mathsf{G}}'_{\mathtt{LG}} = \mathsf{a}{\to}\mathsf{b}{:}\begin{cases} \mathit{love}.\ \widehat{\mathsf{G}}'_{\mathtt{LG}} \\ \mathit{bye} \end{cases}$$

are equivalent for the coinductive LTS, i.e. $\mathbf{Tr}(\mathsf{G}_{\mathtt{LG}}) = \mathbf{Tr}(\widehat{\mathsf{G}}_{\mathtt{LG}})$.

Exercise 10 *Which is the intended protocol represented by the following global type?*

$$\widetilde{\mathsf{G}}_{\mathtt{LG}} = \mathsf{c}{\to}\mathsf{d}{:}\mathit{hello}.\mathsf{a}{\to}\mathsf{b}{:}\begin{cases} \mathit{love}.\ \widetilde{\mathsf{G}}_{\mathtt{LG}} \\ \mathit{bye} \end{cases}$$

Show that, for no G_1 *and* G_2*, it is possible to get* $\widetilde{\mathsf{G}}_{\mathtt{LG}} \xrightarrow{\mathsf{c}\mathit{hello}\mathsf{d}} \mathsf{G}_1 \xrightarrow{\mathsf{c}\mathit{hello}\mathsf{d}} \mathsf{G}_2$.

Let us consider another classic protocol.

`Buyer, Seller and Carrier`

A buyer (b) communicates to a seller (s) a list of arbitrarily many goods she is willing to purchase. This is achieved by repeatedly sending the message *item* until she, possibly, decides to issue the message *buy*. In case this happens, the seller instructs the carrier (c) for the shipment of the goods by sending her the message *ship*.

This protocol corresponds to the following global type:

$$\mathsf{G}_{\mathsf{bsc}} = \mathsf{b}\to\mathsf{s}:\begin{cases} \mathit{item}.\mathsf{G}_{\mathsf{bsc}} \\ \mathit{buy}.\mathsf{s}\to\mathsf{c}:\mathit{ship} \end{cases}$$

Exercise 11 *Notice how the structure of* $\mathsf{G}_{\mathsf{bsc}}$ *is different than that of* $\mathsf{G}_{\mathtt{LG}}$*, as in* $\mathsf{G}_{\mathsf{bsc}}$ *the actions* $\mathsf{b}\mathit{buy}\mathsf{s}$ *and* $\mathsf{s}\mathit{ship}\mathsf{c}$ *share the participant* s *and indeed the latter causally depends on the former. Show that*

$$\mathsf{G}_{\mathsf{bsc}} = \mathsf{b}\to\mathsf{s}:\begin{cases} \mathit{item}.\mathsf{G}_{\mathsf{bsc}} \\ \mathit{buy}.\mathsf{s}\to\mathsf{c}:\mathit{ship} \end{cases} \qquad \textit{and} \qquad \mathsf{G}'_{\mathsf{bsc}} = \mathsf{s}\to\mathsf{c}:\mathit{ship}.\,\mathsf{b}\to\mathsf{s}:\begin{cases} \mathit{item}.\mathsf{G}''_{\mathsf{bsc}} \\ \mathit{buy} \end{cases}$$

where $\mathsf{G}''_{\mathsf{bsc}} = \mathsf{b}\to\mathsf{s}:\{\mathit{item}.\mathsf{G}''_{\mathsf{bsc}}, \mathit{buy}\}$*, are not equivalent.*

MEANINGLESS GLOBAL TYPES

Some global types could be meaningless. For instance, let us consider the following global type, which is a finite version of the previous $\mathsf{G}_{\mathtt{LG}}$.

$$\mathsf{a}\to\mathsf{b}:\begin{cases} \mathit{love} \\ \mathit{bye}.\ \mathsf{c}\to\mathsf{d}:\mathit{hello} \end{cases}$$

Since the communication $\mathsf{c}\mathit{hello}\mathsf{d}$ is independent from the other ones, it is reasonable to expect also the trace $\mathsf{c}\mathit{hello}\mathsf{d}\cdot\mathsf{a}\mathit{love}\mathsf{b}$ to be among those represented by the above global type. This however is not the case when we use the LTS of Def. 2.6. More simply, we have to accept the fact that such a global type is actually meaningless, i.e. non *well-formed* (the usual terminology in the multiparty sessions literature). Roughly, well-formedness for a global type does coincide with the existence of at least a system behaving as described by the global type. Such notion is formalised in terms of *typability* in SMPS and in terms of *projectability* in MPST (see Sections 4.1 and 4.2, respectively).

3 Local Views as Multiparty Sessions

Before providing the formalism we intend to use for representing local views, we have to make some assumptions about which sort of concurrent systems we intend to deal with, in particular

a) a system is made of *named* independent participants[6];

b) to each participant, we associate an abstract behaviour;

c) we are interested only in the communication behaviour of participants;

d) participants interact via message passing using *send* and *receive* communication actions [7].

Under the above assumptions, we can now define a set of terms intended to represent the behaviour of participants. Since we aim to focus only on the interactions among participants, we call such terms *abstract processes* and define them as follows.

Definition 3.1 (Abstract processes) Abstract processes *(A-processes for short) are defined by:*

$$\mathsf{P} \quad ::=^{coind} \quad \mathbf{0} \;\mid\; \mathsf{p}!\{\lambda_i.\mathsf{P}_i\}_{i\in I} \;\mid\; \mathsf{p}?\{\lambda_i.\mathsf{P}_i\}_{i\in I}$$

where $I \neq \emptyset$ *is finite and* $\lambda_h \neq \lambda_k$ *for* $h,k \in I$ *and* $h \neq k$*. We restrict the set of processes to the regular ones, i.e. terms having finitely many distinct subterms.*

[6] There is no univocal sense for a participant name, which can be looked at as an *identifier* or a *location* (in case we used our formalism in the setting of distributed programming).

[7] In the style of concurrent programming languages like Erlang.

We call *input action* an expression of the form $\mathtt{p}?\lambda$ and *output action* one of the form $\mathtt{p}!\lambda$; in both cases we call $\mathtt{p}$ the *subject* of the action. We dub $\mathscr{A}$ the set of possible input and output actions. A-processes, simply "processes" when no ambiguity arises, describe the communication behaviours of participants. The output process $\mathtt{p}!\{\lambda_i.\mathtt{P}_i\}_{i\in I}$ non-deterministically chooses one message λ_k for some $k \in I$, and sends it to the participant $\mathtt{p}$, thereafter continuing as $\mathtt{P}_k$. Symmetrically, the input process $\mathtt{p}?\{\lambda_i.\mathtt{P}_i\}_{i\in I}$ waits for one of the messages λ_i from the participant $\mathtt{p}$, then continues as $\mathtt{P}_k$ after receiving, say, λ_k. The symbol $\mathbf{0}$ is used to denote the terminated process. We shall omit writing trailing $\mathbf{0}$'s in processes and denote $\mathtt{p}!\{\lambda.\mathtt{P}\}$ and $\mathtt{p}?\{\lambda.\mathtt{P}\}$ by $\mathtt{p}!\lambda.\mathtt{P}$ and $\mathtt{p}?\lambda.\mathtt{P}$, respectively.

As they are presented, the input and output actions are just simple synchronisations on messages. As a matter of fact, in actual communicating systems, messages would also carry values that are abstracted away in A-processes for the sake of simplicity. Hence, no selection operation over values is included in the syntax. Actual value transmissions will be considered when we consider more concrete processes. We use $\mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \ldots$ to range over A-processes. *Participants of a session* can now be represented by $(\mathtt{p}, \mathtt{P})$ pairs, where $\mathtt{p}$ is the *name* of the participant and $\mathtt{P}$ is the A-process representing its communication behaviour. We use the intuitive notation $\mathtt{p}[\mathtt{P}]$[8] for $(\mathtt{p}, \mathtt{P})$. Since it does not make sense for a participant $\mathtt{p}$ to send or receive messages from itself, we say that $\mathtt{p}[\mathtt{P}]$ is *well formed* if $\mathtt{P}$ does not include any action of which $\mathtt{p}$ is the subject.

Parallel compositions of a finite number of participants can hence represent concurrent communicating systems.

Definition 3.2 (Multiparty Sessions) Multiparty sessions *(*sessions *for short) are defined by:*

$$\mathbb{M} = \mathtt{p}_1[\mathtt{P}_1] \parallel \cdots \parallel \mathtt{p}_n[\mathtt{P}_n]$$

with $\mathtt{p}_h \neq \mathtt{p}_k$ *for any* $h \neq k$ *and all the* $\mathtt{p}_i[\mathtt{P}_i]$ *are well formed. The set of* participant names *of a session* $\mathbb{M}$, $\mathsf{prt}(\mathbb{M})$, *is defined as*

$$\mathsf{prt}(\mathtt{p}_1[\mathtt{P}_1] \parallel \cdots \parallel \mathtt{p}_n[\mathtt{P}_n]) = \{\mathtt{p}_i \mid \mathtt{P}_i \neq \mathbf{0} \ \ \& \ \ 1 \leq i \leq n\}$$

We call $\mathscr{S}$ *the set of sessions.*

The notation $\mathsf{prt}(\mathbb{M})$ is deliberately overloaded with repsect to $\mathsf{prt}(\mathtt{G})$ in Definition 2.2, as they are subsets of the same set of names. Because of the condition $\mathtt{p}_h \neq \mathtt{p}_k$ for any $h \neq k$, a session is essentially a finite set of (not necessarily distinct) processes $\mathtt{P}_i$ having distinct participant names $\mathtt{p}_i$.

To make $\mathbb{M}$ an actual representation of a set of independent participants running in parallel, we introduce a binary relation $\equiv$ over sessions, usually called *structural congruence* in the literature, such that

$$\mathtt{p}_1[\mathtt{P}_1] \parallel \cdots \parallel \mathtt{p}_n[\mathtt{P}_n] \equiv \mathtt{p}_{\pi_1}[\mathtt{P}_{\pi_1}] \parallel \cdots \parallel \mathtt{p}_{\pi_n}[\mathtt{P}_{\pi_n}]$$

for any permutation π on $\{1,\ldots,n\}$. More precisely, $\equiv$ is the least equivalence making $\parallel$ into a commutative and associative binary operator on participants. Further, since $\mathbf{0}$ represents a terminated process, we postulate that, whenever $\mathtt{p}$ is not a participant in a session, $\mathtt{p}[\mathbf{0}]$ is the unit with respect to $\parallel$, namely $\mathtt{p}[\mathbf{0}] \parallel \mathbb{M} \equiv \mathbb{M}$ for any $\mathbb{M}$. As a consequence, the unit $\mathtt{p}[\mathbf{0}]$ is unique up to $\equiv$ since $\mathtt{p}[\mathbf{0}] \equiv \mathtt{p}[\mathbf{0}] \parallel \mathtt{q}[\mathbf{0}] \equiv \mathtt{q}[\mathbf{0}]$ for all distinct $\mathtt{p}, \mathtt{q}$.

Exercise 12 *Provide an inductive definition of the* $\equiv$ *relation. Is it possible/reasonable to give a coinductive definition?*

[8] When it is clear from the context, we shall ambiguously call *participant* both the whole $\mathtt{p}[\mathtt{P}]$ (with $\mathtt{P} \neq \mathbf{0}$) and its name $\mathtt{p}$.

So far, we have not made any assumption about the synchronisation model for our processes, i.e. we have not specified whether the interactions are synchronous or asynchronous. *Synchronous interactions* require *blocking* output actions, that is, the sender proceeds only when the receiver is ready. In practice, the actual interaction is carried on using a so called *handshaking protocol*, whose details we disregard here. In contrast, *asynchronous interactions* allow instead for *non-blocking* output actions, that is the sender puts the message for the receiver in an appropriate data structure – typically a queue – and the receiver eventually takes it through an input action.

For the sake of simplicity, we assume that participants interact **synchronously**. Although the semantics of global types have been defined in the previous sections keeping in mind synchronous communications, there is no substantial change we ought to do for adapting to the asynchronous model; indeed, communications in $\mathscr{C}$ are abstract representations of the fact that messages can be exchanged respecting certain dependencies, not how this is implemented.

Sessions do evolve using events that are synchronous communications, and their meaning can be defined in terms of traces on $\mathscr{C}$. This will make it easier to prove precise relationships (for instance, correctness and completeness) between sessions and global types. We proceed now toward the formalisation of synchronous interactions, through which sessions evolve.

Formalising synchronous communications

We begin by observing that two participants, say $\mathsf{p}[\![\mathsf{P}]\!]$ and $\mathsf{q}[\![\mathsf{Q}]\!]$, are potentially able to interact whenever P and Q are of the form, respectively, $\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}$ and $\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}$. We also recall that the interaction we intend to describe abstractly is not a simple agreement between p and q on which branch they should proceed on. As previously mentioned, we intend in fact $\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}$ to represent p's nondeterministic choice among the λ_i's together with a communication to the participant q of such a choice, also keeping in mind that in actual interactions a label would carry a value along with it. Notice that, at our level of abstraction, a nondeterministic choice could represent the result of a computation, as discussed in Section 5. Being such a choice uniquely determined by the sender, it is usually referred to also as *internal choice*. The choice among the possible inputs in the receiver depends instead on the message sent. This sort of *driven* choice is usually called *external choice*. In case p chose a label that q were not willing to receive – i.e. not contained in $\{\lambda_j\}_{j\in J}$ – a runtime error would occur, in particular a *communication mismatch* error. This sort of error, arising from the asymmetry between sender and receiver, would not occur if we adopted the more permissive condition that communication is plain synchronization, happening whenever a message λ_i is both among the sender's and the receiver's choices. In our setting, the communication mismatch is modeled by blocking the progress of a session like the above whenever $I \not\subseteq J$.

By the above discussion, we can hence employ the following LTS as the basis of the operational semantics of multiparty sessions.

Definition 3.3 (LTS for synchronous sessions) *The* labelled transition system (LTS) for multiparty sessions*, with communications in $\mathscr{C}$ as actions, is the closure under structural congruence of the LTS specified by the following axiom:*

$$[\text{S-Comm}]\quad \frac{}{\mathsf{p}[\![\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}]\!] \parallel \mathsf{q}[\![\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}]\!] \parallel \mathbb{M} \xrightarrow{\mathsf{p}\lambda_k\mathsf{q}} \mathsf{p}[\![\mathsf{P}_k]\!] \parallel \mathsf{q}[\![\mathsf{Q}_k]\!] \parallel \mathbb{M}}\ (k\in I\subseteq J)$$

We remark that $I \subseteq J$ is not to be intended as a condition to be dynamically checked at run time before any communication (and hence quite unfeasible in that case), but rather as a condition possibly leading to a communication mismatch when violated, and also as a condition we can rely on for typable sessions. Such an approach enables formalizing synchronous communications among participants in a very

abstract way using a single axiom. Other approaches are of course, possible. For instance, by explicitly and separately representing the nondeterministic (internal) choice of p and the subsequent communication to q (see Definition 5.5 and related discussion). As well as by representing explicitly an error state to be reached in case $k \notin J$ (see Exercise 42).

Exercise 13 *Formally define* $\mathbf{Tr}(\mathbb{M})$ *as the set of traces of a session* $\mathbb{M}$ *according to the LTS of Definition 3.3.*

Let us now provide an example of a session intended to behave as prescribed by the `Love and Greeting` protocol.

Example 3.4 (A session for the `Love and Greeting` protocol)

$$\mathbb{M}_{\mathtt{LG}} = \mathsf{a}[\mathsf{P}_{\mathtt{LG}}] \parallel \mathsf{b}[\mathsf{Q}_{\mathtt{LG}}] \parallel \mathsf{c}[\mathsf{d}\,!\,\mathit{hello}] \parallel \mathsf{d}[\mathsf{c}\,?\,\mathit{hello}]$$

where $\mathsf{P}_{\mathtt{LG}} = \mathsf{b}\,!\,\{\mathit{love}.\mathsf{P}_{\mathtt{LG}},\ \mathit{bye}\}$, *and* $\mathsf{Q}_{\mathtt{LG}} = \mathsf{a}\,?\,\{\mathit{love}.\mathsf{Q}_{\mathtt{LG}},\ \mathit{bye}\}$. ◇

Exercise 14 *Show that* $\mathbb{M}_{\mathtt{LG}} \xrightarrow{\mathsf{a}\,\mathit{love}\,\mathsf{b}} \mathbb{M}_{\mathtt{LG}}$, *namely that there exists an infinite sequence of labelled transitions in which neither Carl nor Daisy are involved.*
Moreover, show that Carl and Daisy are never prevented from greeting each other. That is, formally

$$\mathbb{M}_{\mathtt{LG}} \xrightarrow{(\mathsf{a}\,\mathit{love}\,\mathsf{b})^*} \mathbb{M}_{\mathtt{LG}} \xrightarrow{\mathsf{c}\,\mathit{hello}\,\mathsf{d}} \mathsf{a}[\mathsf{P}_{\mathtt{LG}}] \parallel \mathsf{b}[\mathsf{Q}_{\mathtt{LG}}]$$

Lastly, show that a *and* b *can always terminate by a transition labelled by* bye.

Example 3.5 (A session for the `Buyer, Seller and Carrier` protocol)

$$\mathbb{M}_{\mathsf{bsc}} = \mathsf{b}[\mathsf{P}_{\mathsf{bsc}}] \parallel \mathsf{s}[\mathsf{Q}_{\mathsf{bsc}}] \parallel \mathsf{c}[\mathsf{s}?\mathit{ship}]$$

where $\mathsf{P}_{\mathsf{bsc}} = \mathsf{s}!\{\mathit{item}.\mathsf{P}_{\mathsf{bsc}}, \mathit{buy}\}$, $\mathsf{Q}_{\mathsf{bsc}} = \mathsf{b}?\{\mathit{item}.\mathsf{Q}_{\mathsf{bsc}}, \mathit{buy}.\mathsf{c}!\mathit{ship}\}$.
Similarly to the previous example, we have $\mathbb{M}_{\mathsf{bsc}} \xrightarrow{(\mathsf{b}\,\mathit{item}\,\mathsf{s})}{}^{*} \mathbb{M}_{\mathsf{bsc}}$, *but the carrier will receive the message* ship *only after the seller has received* buy *from the buyer.* ◇

Session properties We now define some relevant properties of sessions; some are related to communications, and others are specific for choreographic settings.

The property of Lock-freedom ensures there is always a continuation enabling a participant to communicate whenever it is willing to do so. Deadlock freedom ensures that a session terminates if and only if all its participants do. Communication-mismatch freedom ensures that two corresponding senders and receivers can always safely interact.

Definition 3.6 (Communication properties) *Let* $\mathbb{M} \to^* \mathbb{M}'$ *abbreviate* $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$ *for some* $\sigma \in \mathscr{C}^*$. *A session* $\mathbb{M}$ *is*

i) *a* deadlock *if* $\mathbb{M} \not\equiv \mathsf{p}[\mathbf{0}]$ *and* $\mathbb{M} \not\to$

ii) *a* lock for p *if no* $\sigma \in \mathbf{Tr}(\mathbb{M})$ *contains a communication either of the form* $\mathsf{p}\lambda\mathsf{q}$ *or* $\mathsf{q}\lambda\mathsf{p}$

iii) *a (potential)* communication mismatch *if*

$$\mathbb{M} \equiv \mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M}' \text{ and } I \not\subseteq J$$

iv) deadlock free *if, for any* $\mathbb{M}'$, $\mathbb{M} \to^* \mathbb{M}'$ *implies* $\mathbb{M}'$ *is not a deadlock*

v) lock free *if, for any* $\mathbb{M}'$ *and* $\mathsf{p} \in \mathsf{ptp}(\mathbb{M})$, $\mathbb{M} \to^* \mathbb{M}'$ *implies* $\mathbb{M}'$ *is not a lock for* p

vi) communication-mismatch free *if, for any* $\mathbb{M}'$,
$$\mathbb{M} \to^* \mathbb{M}' \text{ implies } \mathbb{M}' \text{ is not a communication-mismatch.}$$

Exercise 15 *Show that the following one is an equivalent definition of lock freedom.*

> *A session* $\mathbb{M}$ *is* lock free *if* $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$ *with* σ *finite and* $\mathsf{p} \in \mathsf{prt}(\mathbb{M}')$ *imply* $\mathbb{M}' \xrightarrow{\sigma' \cdot \Lambda}$ *for some* σ' *and* $\Lambda \in \mathscr{C}$ *such that* $\mathsf{p} \in \mathsf{prt}(\Lambda)$, *where* $\mathsf{prt}(\mathsf{r}\lambda\mathsf{s}) = \{\mathsf{r}, \mathsf{s}\}$.

Exercise 16 *Show that*

i) lock freedom implies deadlock freedom and that the vice versa does not hold;

ii) lock freedom implies communication-mismatch freedom and that the vice versa does not hold;

iii) neither one between deadlock freedom and communication-mismatch freedom implies the other.

Hint: *for (i) and (ii), consider a session* $\mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M}'$ *such that* $I \cap J = \emptyset$, p, q *do not occurr in any* $\mathsf{r}[\mathsf{R}]$ *in* $\mathbb{M}'$ *and* $\mathbb{M}' \xrightarrow{\Lambda} \mathbb{M}''$ *for some* $\Lambda \in \mathscr{C}$ *and session* $\mathbb{M}''$.

Exercise 17 *Discuss about the possibility of looking also at a session like the following one as a* communication-mismatch.
$$\mathbb{M} \equiv \mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}] \parallel \mathsf{q}[\mathsf{r}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M}' \quad \text{with } \mathsf{r} \neq \mathsf{p}$$

We now formalise a couple of properties previously defined in an informal way.

Definition 3.7 (Choreographic properties) *Let* G *be a global type and* $\mathbb{M}$ *a session.*

i) G *is* correct for $\mathbb{M}$ *if* $\mathbf{Tr}(\mathsf{G}) \subseteq \mathbf{Tr}(\mathbb{M})$.

ii) G *is* complete for $\mathbb{M}$ *if* $\mathbf{Tr}(\mathbb{M}) \subseteq \mathbf{Tr}(\mathsf{G})$.

Exercise 18 *Show that the first global type below cannot be complete and that the second one cannot be neither correct nor complete for any session.*

$$\mathsf{p}\to\mathsf{q}: \begin{cases} m1.\mathsf{r}\to\mathsf{s}:hello \\ m2.\mathsf{r}\to\mathsf{s}:hi \end{cases} \qquad \mathsf{p}\to\mathsf{q}: \begin{cases} m1.\mathsf{p}\to\mathsf{r}:hello \\ m2.\mathsf{r}\to\mathsf{s}:hi \end{cases}$$

4 Relating Global and Local views

Types in programming are used in two possible and orthogonal ways (or in a mix of both):

- as constraints helping the programmer to prevent run-time errors;
- as predicates whose proofs do guarantee the (partial) correctness of programs.

The very same approaches (now for concurrent programming) can be followed in our abstract choreographic setting where *sessions* and *global types* formalise, respectively, sets of communicating processes running in parallel and their overall behaviours.

4.1 The SMPS approach

We discuss now a *type assignment* approach to relate global types and sessions via a formal system (a type system). The idea is to derive statements of the shape $\mathsf{G} \vdash \mathbb{M}$ whose intended meaning is that $\mathbb{M}$ is a faithful implementation of the protocol G (i.e. G is correct an complete for $\mathbb{M}$). This is different than what happens in the standard MPST formalisms, where global types are projected to local types, called *session types*, which in turn can be deduced for processes. (We refer to [18] for a simple introduction to the standard MPST formalisms.) The present approach is dubbed *Simple MultiParty Sessions* or SMPS as it relates global types and sessions directly; for a discussion relating SMPS to MPST, see 4.2 below.

Definition 4.1 (Type System) *Judgements of the form $\mathsf{G} \vdash \mathbb{M}$ are coinductively derived by the type system below, by considering sessions up to structural congruence:*

$$[\text{T-End}]\ \frac{}{\mathsf{End} \vdash \mathsf{p}[\mathbf{0}]}$$

$$[\text{T-Comm}]\ \frac{I \subseteq J \qquad \mathsf{G}_i \vdash \mathsf{p}[\mathsf{P}_i] \parallel \mathsf{q}[\mathsf{Q}_i] \parallel \mathbb{M} \qquad \mathsf{prt}(\mathsf{G}_i) \setminus \{\mathsf{p},\mathsf{q}\} = \mathsf{prt}(\mathbb{M}) \quad \forall i \in I}{\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I} \vdash \mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M}}$$

Rule [T-Comm] just adds communications to global types and to their respective processes inside sessions. More inputs than corresponding outputs are allowed by this rule, in accordance with the side condition of rule [S-Comm] (Definition 3.3). It also allows more branches in the input process than the global type. Such a feature is related to subtyping, which we shall briefly deal with when discussing the MPST approach. Instead, the number of branches in the output process and the global type must be the same. This will allow us to get the property of Session Fidelity (i.e. Correctness) without impairing typability.

Unwanted effects of infinite derivations are prevented by requiring that the global type and the session have the same set of participants, as expressed by the condition $\mathsf{prt}(\mathsf{G}_i) \setminus \{\mathsf{p},\mathsf{q}\} = \mathsf{prt}(\mathbb{M})$ for all $i \in I$. Otherwise, the following judgement would be derivable for any $\mathsf{R} \neq \mathbf{0}$:

$$\mathsf{p} \to \mathsf{q} : \lambda.\mathsf{G} \vdash \mathsf{p}[\mathsf{q}!\lambda.\mathsf{P}] \parallel \mathsf{q}[\mathsf{p}?\lambda.\mathsf{Q}] \parallel \mathsf{r}[\mathsf{R}]$$

where $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \lambda.\mathsf{G}$, $\mathsf{P} = \mathsf{q}!\lambda.\mathsf{P}$ and $\mathsf{Q} = \mathsf{p}?\lambda.\mathsf{Q}$. In particular,

$$\mathscr{D} = \frac{\mathscr{D}}{\mathsf{p} \to \mathsf{q} : \lambda.\mathsf{G} \vdash \mathsf{p}[\mathsf{q}!\lambda.\mathsf{P}] \parallel \mathsf{q}[\mathsf{p}?\lambda.\mathsf{Q}] \parallel \mathsf{r}[\mathsf{R}]}$$

We now illustrate the type system by deriving global types for the sessions of Examples 3.4 and 3.5. In the derivations, we omit the axiom/rule names as they are clear from the context. Moreover, we do not write the conditions on the participants, which can be easily checked.

Example 4.2 (A typing for `Love and Greeting`) *Consider the session $\mathbb{M}_{\mathtt{LG}}$ from Example 3.4. Then, by recalling*

$$\mathsf{G}_{\mathtt{LG}} = \mathsf{a} \to \mathsf{b} : \{\mathit{love}.\mathsf{G}_{\mathtt{LG}},\ \mathit{bye}.\mathsf{c} \to \mathsf{d} : \mathit{hello}\},$$

the following is a derivation proving that $\mathsf{G}_{\mathtt{LG}} \vdash \mathbb{M}_{\mathtt{LG}}$.

$$\dfrac{\dfrac{\dfrac{\mathtt{etc.}}{\vdots}\quad \dfrac{\mathsf{End} \vdash \mathsf{c}[\mathbf{0}] \parallel \mathsf{d}[\mathbf{0}]}{\mathsf{c} \to \mathsf{d} : \mathit{hello} \vdash \mathsf{c}[\mathsf{d}\,!\,\mathit{hello}] \parallel \mathsf{d}[\mathsf{c}\,?\,\mathit{hello}]}}{\mathsf{G}_{\mathtt{LG}} \vdash \mathbb{M}_{\mathtt{LG}}} \qquad \dfrac{\mathsf{End} \vdash \mathsf{c}[\mathbf{0}] \parallel \mathsf{d}[\mathbf{0}]}{\mathsf{c} \to \mathsf{d} : \mathit{hello} \vdash \mathsf{c}[\mathsf{d}\,!\,\mathit{hello}] \parallel \mathsf{d}[\mathsf{c}\,?\,\mathit{hello}]}}{\mathsf{G}_{\mathtt{LG}} \vdash \mathbb{M}_{\mathtt{LG}}}$$

which is finitely representable by

$$\mathscr{D}_{\mathtt{LG}} \;=\; \dfrac{\dfrac{\mathscr{D}_{\mathtt{LG}}}{\mathsf{G}_{\mathtt{LG}} \vdash \mathbb{M}_{\mathtt{LG}}} \qquad \dfrac{}{\mathsf{End} \vdash \mathsf{c}[\mathbf{0}] \parallel \mathsf{d}[\mathbf{0}]}}{\mathsf{c} \to \mathsf{d} : \mathit{hello} \vdash \mathsf{c}[\mathsf{d}\,!\,\mathit{hello}] \parallel \mathsf{d}[\mathsf{c}\,?\,\mathit{hello}]}}{\mathsf{G}_{\mathtt{LG}} \vdash \mathbb{M}_{\mathtt{LG}}}$$

◇

Notice that in the above example, the interaction between a and b, and the interaction between c and d are independent.

Exercise 19 *Let* $\widehat{\mathsf{G}}_{\mathtt{LG}}$ *be the global type of Exercise 9. Show that* $\widehat{\mathsf{G}}_{\mathtt{LG}} \vdash \mathbb{M}_{\mathtt{LG}}$.

Exercise 20 *Let* $\widetilde{\mathsf{G}}_{\mathtt{LG}}$ *be the global type of Exercise 10. Show informally that there is no session* $\mathbb{M}$ *such that* $\widetilde{\mathsf{G}}_{\mathtt{LG}} \vdash \mathbb{M}$.

Example 4.3 (A typing for `Buyer, Seller and Carrier`) $\mathbb{M}_{\mathsf{bsc}}$ *can be typed by* $\mathsf{G}_{\mathsf{bsc}}$ *using the following derivation:*

$$\dfrac{\dfrac{\dfrac{\mathtt{etc.}}{\vdots} \qquad \dfrac{}{\mathsf{End} \vdash \mathsf{b}[\mathbf{0}] \parallel \mathsf{s}[\mathbf{0}] \parallel \mathsf{c}[\mathbf{0}]}}{\mathsf{s} \to \mathsf{c}{:}\mathit{ship} \vdash \mathsf{b}[\mathbf{0}] \parallel \mathsf{s}[\mathsf{c}!\mathit{ship}] \parallel \mathsf{c}[\mathsf{s}?\mathit{ship}]} \qquad \dfrac{}{\mathsf{End} \vdash \mathsf{b}[\mathbf{0}] \parallel \mathsf{s}[\mathbf{0}] \parallel \mathsf{c}[\mathbf{0}]}}{\mathsf{G}_{\mathsf{bsc}} \vdash \mathbb{M}_{\mathsf{bsc}}} \qquad \dfrac{}{\mathsf{s} \to \mathsf{c}{:}\mathit{ship} \vdash \mathsf{b}[\mathbf{0}] \parallel \mathsf{s}[\mathsf{c}!\mathit{ship}] \parallel \mathsf{c}[\mathsf{s}?\mathit{ship}]}}{\mathsf{G}_{\mathsf{bsc}} \vdash \mathbb{M}_{\mathsf{bsc}}}$$

Exercise 21 *Provide a finite representation of the infinite derivation informally depicted in Example 4.3.*

Exercise 22 *Is it possible to find a global type for* $\mathbb{M}_{\mathsf{bsc}}$ *different from* $\mathsf{G}_{\mathsf{bsc}}$*? Justify the answer.*

Exercise 23 *Describe a session for the global type representing the* `Travel Agency` *protocol. Then, provide a derivation showing that the session has that global type.*

Typability entails several properties.

Theorem 4.4 (Subject Reduction) *If* $\mathsf{G} \vdash \mathbb{M}$ *and* $\mathbb{M} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathbb{M}'$*, then* $\mathsf{G} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}'$ *and* $\mathsf{G}' \vdash \mathbb{M}'$ *for some* G'.

Exercise 24 *Show that Subject Reduction implies that if* $\mathsf{G} \vdash \mathbb{M}$*, then* G *is* complete *for* $\mathbb{M}$.

Theorem 4.5 (Session Fidelity) *If* $\mathsf{G} \vdash \mathbb{M}$ *and* $\mathsf{G} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}'$*, then* $\mathbb{M} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathbb{M}'$ *and* $\mathsf{G}' \vdash \mathbb{M}'$ *for some* $\mathbb{M}'$.

Exercise 25 *Show that Session Fidelity implies that if* $\mathsf{G} \vdash \mathbb{M}$*, then* G *is* correct *for* $\mathbb{M}$.

Remark 4.6 *What is established by theorems 4.4 and 4.5 and the exercises above is apparently more than the equality* $\mathbf{Tr}(\mathsf{G}) = \mathbf{Tr}(\mathbb{M})$ *when* $\mathsf{G} \vdash \mathbb{M}$*. Indeed, if* $\mathsf{G} \vdash \mathbb{M}$ *then* G *and* $\mathbb{M}$ *are* bisimilar *w.r.t. the LTS* $(\mathscr{G} \cup \mathscr{S}, \mathscr{C}, \to)$ *where* $\to$ *is the union of the LTS in definitions 2.6 and 3.3. We defer the discussion on this point to the end of Section 5.* ◇

It is possible to obtain the following as a consequence of Subject Reduction and Session Fidelity [3].

Theorem 4.7 (Lock freedom) *If* $\mathbb{M}$ *is typable, then* $\mathbb{M}$ *is lock free.*

It is worth pointing out that, by allowing more branches in the global type than in the output process (and fewer branches in the global type than in the input process), the property that we dubbed *Session Fidelity* above would be lost. In fact, one could derive

$$\mathsf{p}{\to}\mathsf{q}{:}\{\lambda_1,\lambda_2\} \vdash \mathsf{p}[\mathsf{q}!\lambda_1] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_1,\lambda_2\}]$$

and check that

$$\mathsf{p}{\to}\mathsf{q}{:}\{\lambda_1,\lambda_2\} \xrightarrow{\mathsf{p}\lambda_2\mathsf{q}} \mathsf{End} \quad \text{but} \quad \mathsf{p}[\mathsf{q}!\lambda_1] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_1,\lambda_2\}] \not\xrightarrow{\mathsf{p}\lambda_2\mathsf{q}}.$$

One could prove, however, the following version of Session Fidelity[9]:

$$\text{If } \mathsf{G} \vdash \mathbb{M} \text{ and } \mathsf{G} \xrightarrow{\mathsf{p}\lambda\mathsf{q}}, \text{ then there are } \lambda' \text{ and } \mathsf{G}' \text{ s.t. } \mathbb{M} \xrightarrow{\mathsf{p}\lambda'\mathsf{q}} \mathbb{M}' \text{ and } \mathsf{G}' \vdash \mathbb{M}'. \tag{3}$$

Exercise 26 *Justify the following statement:*
Subject Reduction and Property (3) imply deadlock freedom for typable sessions.

4.2 The MPST approach

The MPST approach consists of getting sessions directly from global types such that the obtained sessions enjoy relevant properties. The A-processes for the various participants are "extracted" (when possible) in a natural way from the global type.

We start by adapting the MPST projection to the present case where we directly consider processes instead of local types.

Definition 4.8 (Projections) [10] *Given a global type* G *and a participant* p*, the A-process* $\mathsf{G}\upharpoonright\mathsf{p}$*, called* the projection of G at p*, is* **coinductively** *defined by:*

$$\frac{}{\mathsf{G}\upharpoonright\mathsf{p} = \mathbf{0}}\ \mathsf{p} \notin \mathsf{prt}(\mathsf{G})$$

$$\frac{\mathsf{G}_i\upharpoonright\mathsf{p} = \mathsf{P}_i \quad \forall i \in I}{(\mathsf{p}{\to}\mathsf{s}{:}\{\lambda_i.\mathsf{G}_i\}_{i\in I})\upharpoonright\mathsf{p} = \mathsf{s}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}} \qquad \frac{\mathsf{G}_i\upharpoonright\mathsf{p} = \mathsf{P}_i \quad \forall i \in I}{(\mathsf{r}{\to}\mathsf{p}{:}\{\lambda_i.\mathsf{G}_i\}_{i\in I})\upharpoonright\mathsf{p} = \mathsf{r}?\{\lambda_i.\mathsf{P}_i\}_{i\in I}}$$

$$\frac{\mathsf{G}_i\upharpoonright\mathsf{p} = \mathsf{P}_i \quad \forall i \in I \quad \bigsqcap_{i\in I}\mathsf{P}_i = \mathsf{P}}{(\mathsf{r}{\to}\mathsf{s}{:}\{\lambda_i.\mathsf{G}_i\}_{i\in I})\upharpoonright\mathsf{p} = \mathsf{P}}\ (\mathsf{p} \in \mathsf{prt}(\mathsf{G}) \textit{ and } \mathsf{p} \notin \{\mathsf{r},\mathsf{s}\})$$

where the (full) merge $P \sqcap Q$ *is the partial operation* **coinductively** *defined by*

$$\frac{}{\mathbf{0} \sqcap \mathbf{0} = \mathbf{0}} \qquad \frac{\mathsf{P}_i \sqcap \mathsf{Q}_i = \mathsf{R}_i \quad \forall i \in I}{\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I} \sqcap \mathsf{q}!\{\lambda_i.\mathsf{Q}_i\}_{i\in I} = \mathsf{q}!\{\lambda_i.\mathsf{R}_i\}_{i\in I}}$$

$$\frac{\mathsf{P}_h \sqcap \mathsf{Q}_h = \mathsf{R}_h \quad \forall h \in I \cap J}{\mathsf{q}?\{\lambda_i.\mathsf{P}_i\}_{i\in I} \sqcap \mathsf{q}?\{\lambda_i.\mathsf{Q}_j\}_{i\in J} = \mathsf{q}?\{\lambda_i.\mathsf{P}_i, \lambda_j.\mathsf{Q}_j, \lambda_h.\mathsf{R}_h \mid i \in I\setminus J, j \in J\setminus I, h \in I\cap J\}}$$

[9] Such a version is actually what was primarily proved by K.Honda in his seminal papers and referred to as *Session Fidelity* in a relevant part of the literature on MPST.

[10] There is a subtle difference between the present definition of projection and Definition 6.1 in [3]. The latter adapts to SMPS the definitions in [11, 16], where both projections and merge are defined *corecursively* (erroneously named coinductive in [3]) over the structure of types and processes. Here we adopt, instead, a coinductive definition of the relation of being the projection of a global type, which turns out to be functional and properly extends the corecursive definition.

Example 4.9 *Let* $\mathsf{G}_{\mathtt{TA}}$ *be the global type (1) in Sect. 2 for the finite version of the* `TravelAgency` *protocol, namely*

$$\mathsf{G}_{\mathtt{TA}} = \mathsf{c}\to\mathsf{a}{:}\mathit{request}.\,\mathsf{a}\to\mathsf{c}{:}\mathit{price}.\,\mathsf{c}\to\mathsf{a}\left\{\begin{array}{l}\mathit{accept}.\,\mathsf{c}\to\mathsf{s}{:}\mathit{date}\\ \mathit{reject}.\,\mathsf{c}\to\mathsf{s}{:}\mathit{nodeal}\end{array}\right.$$

Let us compute $\mathsf{G}_{\mathtt{TA}}\upharpoonright\mathsf{s}$*:*

$$\begin{array}{rcll}\mathsf{G}_{\mathtt{TA}}\upharpoonright\mathsf{s} & = & (\mathsf{c}\to\mathsf{s}{:}\mathit{date})\upharpoonright\mathsf{s}\sqcap(\mathsf{c}\to\mathsf{s}{:}\mathit{nodeal})\upharpoonright\mathsf{s} & \text{since } \mathsf{s}\notin\{\mathsf{c},\mathsf{a}\}\\ & = & (\mathsf{c}?\mathit{date})\sqcap(\mathsf{c}?\mathit{nodeal}) & \\ & = & \mathsf{c}?\{\mathit{date},\mathit{nodeal}\} & \text{since } \{\mathit{date}\}\cap\{\mathit{nodeal}\}=\emptyset\end{array}$$

◇

Exercise 27 *Compute* $\mathsf{G}_{\mathtt{TA}}\upharpoonright\mathsf{c}$ *and* $\mathsf{G}_{\mathtt{TA}}\upharpoonright\mathsf{a}$*.*

When $\mathsf{G}\upharpoonright\mathsf{p}=\mathsf{P}$ is derivable we say that $\mathsf{G}\upharpoonright\mathsf{p}$ is *defined*. On the other hand, it is not difficult to find a type G and a participant $\mathsf{p}\in\mathsf{prt}(\mathsf{G})$ such that $\mathsf{G}\upharpoonright\mathsf{p}$ is undefined, e.g.

$$(\mathsf{r}\to\mathsf{s}{:}\{\lambda_1.\mathsf{p}\to\mathsf{q}{:}\lambda_2.\mathsf{End},\lambda_3.\mathsf{End}\})\upharpoonright\mathsf{p}$$

where $(\mathsf{p}\to\mathsf{q}{:}\lambda_2.\mathsf{End})\upharpoonright\mathsf{p}=\mathsf{q}!\lambda_2$, $(\lambda_3.\mathsf{End})\upharpoonright\mathsf{p}=\mathbf{0}$ but there is no P such that $\mathsf{q}!\lambda_2\sqcap\mathbf{0}=\mathsf{P}$ is derivable.

Example 4.10 *Let* $\mathsf{G}'_{\mathtt{TA}}$ *be the following variant of* $\mathsf{G}_{\mathtt{TA}}$*, where the roles of* c *and* s *have been exchanged in the rightmost part of the type:*

$$\mathsf{G}'_{\mathtt{TA}} = \mathsf{c}\to\mathsf{a}{:}\mathit{request}.\,\mathsf{a}\to\mathsf{c}{:}\mathit{price}.\,\mathsf{c}\to\mathsf{a}\left\{\begin{array}{l}\mathit{accept}.\,\mathsf{s}\to\mathsf{c}{:}\mathit{date}\\ \mathit{reject}.\,\mathsf{s}\to\mathsf{c}{:}\mathit{nodeal}\end{array}\right.$$

Now, let us compute:

$$\begin{array}{rcll}\mathsf{G}'_{\mathtt{TA}}\upharpoonright\mathsf{s} & = & (\mathsf{s}\to\mathsf{c}{:}\mathit{date})\upharpoonright\mathsf{s}\sqcap(\mathsf{s}\to\mathsf{c}{:}\mathit{nodeal})\upharpoonright\mathsf{s} & \text{since } \mathsf{s}\notin\{\mathsf{c},\mathsf{a}\}\\ & = & (\mathsf{c}!\mathit{date})\sqcap(\mathsf{c}!\mathit{nodeal}) & \\ & = & \mathit{undefined} & \text{since } \{\mathit{date}\}\neq\{\mathit{nodeal}\}\end{array}$$

This should be confronted with the fact that $\mathsf{G}_{\mathtt{TA}}\upharpoonright\mathsf{s}$ *is defined, as we have seen in Example 4.9. The intuitive explanation is that in both types* $\mathsf{c}\to\mathsf{s}{:}\mathit{date}$ *and* $\mathsf{c}\to\mathsf{s}{:}\mathit{nodeal}$ *the participant* s *is passive, waiting for a message from* c *to decide how to continue. Also, there is no ambiguity in the sets of messages on the two branches, as they are disjoint.*

On the contrary, when computing $\mathsf{G}'_{\mathtt{TA}}\upharpoonright\mathsf{s}$ *we have to merge the projection to* s *of the types* $\mathsf{s}\to\mathsf{c}{:}\mathit{date}$ *and* $\mathsf{s}\to\mathsf{c}{:}\mathit{nodeal}$*, where* s *is the active participant that is expected to send either* date *or* nodeal*, a choice that is unspecified in* $\mathsf{G}'_{\mathtt{TA}}$*.* ◇

Example 4.11 *Consider the type* $\mathsf{G}_{\mathsf{bsc}}$ *of the Buyer, Seller and Carrier protocol:*

$$\mathsf{G}_{\mathsf{bsc}} = \mathsf{b}\to\mathsf{s}{:}\left\{\begin{array}{l}\mathit{item}.\mathsf{G}_{\mathsf{bsc}}\\ \mathit{buy}.\mathsf{s}\to\mathsf{c}{:}\mathit{ship}\end{array}\right.$$

Let us compute $\mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c}$*.*

$$\begin{array}{rcll}\mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c} & = & \mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c}\sqcap(\mathsf{s}\to\mathsf{c}{:}\mathit{ship})\upharpoonright\mathsf{c} & \text{because } \mathsf{c}\notin\{\mathsf{b},\mathsf{s}\}\\ & & \mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c}\sqcap\mathsf{s}?\mathit{ship} & \end{array}$$

We are stuck with a recursive equation that cannot be solved by only looking at the definition of merge. However, $\mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c}$ *is not undefined. Instead, let us try to derive* $\mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c} = \mathsf{s}?\mathit{ship}$ *using the last rule in the definition of projection; then we get*

$$\mathscr{D} \;=\; \dfrac{\dfrac{\mathscr{D}}{\mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c} = \mathsf{s}?\mathit{ship}} \qquad \dfrac{\mathsf{End}\upharpoonright\mathsf{c} = \mathbf{0}}{(\mathsf{s}\to\mathsf{c}{:}\mathit{ship})\upharpoonright\mathsf{c} = \mathsf{s}?\mathit{ship}} \qquad \dfrac{}{\mathbf{0}\sqcap\mathbf{0} = \mathbf{0}}}{\mathsf{G}_{\mathsf{bsc}}\upharpoonright\mathsf{c} = \mathsf{s}?\mathit{ship}}$$

which is perfectly legit. ◇

The relation $\mathsf{G}\upharpoonright\mathsf{p} = \mathsf{P}$ defined by the rules in Definition 4.8 is functional in the sense that if both $\mathsf{G}\upharpoonright\mathsf{p} = \mathsf{P}$ and $\mathsf{G}\upharpoonright\mathsf{p} = \mathsf{Q}$ then $\mathsf{P} = \mathsf{Q}$. This is proved by coinduction on the definition of both $\mathsf{G}\upharpoonright\mathsf{p}$ and $\mathsf{P}\sqcap\mathsf{Q}$. First, we establish the following lemma, whose proof is provided also as an example of coinductive reasoning:

Lemma 4.12 *If both* $\mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}$ *and* $\mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}'$ *are derivable then* $\mathsf{R} = \mathsf{R}'$.

Proof. Observe that, while the equations $\mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}$ and $\mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}'$ are formally derivable statements, by $\mathsf{R} = \mathsf{R}'$ we mean syntactical coincidence, as we treat the expressions like $\{\lambda_i.\mathsf{P}_i\}_{i\in I}$ as (finite) sets.

Now, the last rule in the derivation of $\mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}$ is determined by the shapes of P and Q in the sense that either they must be both $\mathbf{0}$ or they must begin both by either $\mathsf{q}!$ or $\mathsf{q}?$ for the same q. In the first case, we have that $\mathsf{R} = \mathbf{0} = \mathsf{R}'$. Suppose instead that $\mathsf{P} = \mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}$ and $\mathsf{Q} = \mathsf{q}!\{\lambda_i.\mathsf{Q}_i\}_{i\in I}$. Then $\mathsf{R} = \mathsf{q}!\{\lambda_i.\mathsf{S}_i\}_{i\in I}$ and $\mathsf{R}' = \mathsf{q}!\{\lambda_i.\mathsf{S}'_i\}_{i\in I}$ where for all $i \in I$

$$\mathsf{P}_i\sqcap\mathsf{Q}_i = \mathsf{S}_i \qquad \text{and} \qquad \mathsf{P}_i\sqcap\mathsf{Q}_i = \mathsf{S}'_i$$

are derivable. By coinductive hypothesis $\mathsf{S}_i = \mathsf{S}'_i$ for all $i \in I$, hence $\mathsf{R} = \mathsf{R}'$.

The case in which both P and Q begin by $\mathsf{q}?$ is similar. □

The above proof might appear incorrect at a first sight since, while it looks very similar to induction, it cannot be such in general as the P,Q and R can be infinite and so the axiom $\mathbf{0}\sqcap\mathbf{0} = \mathbf{0}$ is never reached in at least some branch of the derivation. Indeed, what we have proved is that the relation

$$\mathscr{R} = \{(\mathsf{R},\mathsf{R}') \mid \exists \mathsf{P},\mathsf{Q}.\ \mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}\ \ \&\ \ \mathsf{P}\sqcap\mathsf{Q} = \mathsf{R}'\}$$

passes backword any inference in the definition of merge, that is, from the conclusion to the premises, and that it is included into syntactical equality of possibly infinite expressions.

With Lemma 4.12 at hand, we can establish the proposition:

Proposition 4.13 *The relation* $\mathsf{G}\upharpoonright\mathsf{p} = \mathsf{P}$ *is a partial function.*

Exercise 28 *Work out the details of the proof of Proposition 4.13 using Lemma 4.12.*

We say that G is *projectable* if $\mathsf{G}\upharpoonright\mathsf{p}$ is defined for all $\mathsf{p} \in \mathsf{prt}(\mathsf{G})$. In that case, we also define $\mathsf{G}\upharpoonright$ as the session with $\mathsf{prt}(\mathsf{G})$ as set of participants and with the projections of G on the participants in $\mathsf{prt}(\mathsf{G})$ as processes, that is

$$\mathsf{G}\upharpoonright = \Big\|_{\mathsf{p}\in\mathsf{prt}(\mathsf{G})} \mathsf{p}[\mathsf{G}\upharpoonright\mathsf{p}]$$

In the literature, there are various definitions of merge. We use a version of the more permissive one called *full merge*.

Exercise 29 *Complete Example 4.10 and build the session*

$$\mathsf{G}_{\mathtt{TA}}\upharpoonright = \mathsf{G}_{\mathtt{TA}}\upharpoonright \mathsf{c} \parallel \mathsf{G}_{\mathtt{TA}}\upharpoonright \mathsf{a} \parallel \mathsf{G}_{\mathtt{TA}}\upharpoonright \mathsf{s}.$$

Then, compare it with the one of Exercise 23.

Exercise 30 *Show that the two global types of Exercise 18 are not projectable.*

Exercise 31 *Complete the example 4.11 building the session* $\mathsf{G}_{\mathsf{bsc}}\upharpoonright$.

Exercise 32 *Show that* $\widetilde{\mathsf{G}}_{\mathtt{LG}}$ *of Exercise 10 is not projectable.*

The possibility of typing a session where a receiver has more inputs than those present in the global type, makes the SMPS approach more expressive than the MPST approach as described up to now. In fact

$$\mathsf{p}{\to}\mathsf{q}{:}\lambda \vdash \mathsf{p}[\mathsf{q}!\lambda] \parallel \mathsf{q}[\mathsf{p}?\{\lambda, \lambda'\}] \quad \text{and} \quad (\mathsf{p}{\to}\mathsf{q}{:}\lambda)\upharpoonright \mathsf{q} = \mathsf{p}?\lambda \neq \mathsf{p}?\{\lambda, \lambda'\}.$$

To get the equivalence of expressivity, we won't restrict the SMPS type system: it would be inconsistent in the typing system to rule out the session $\mathsf{p}[\mathsf{q}!\lambda] \parallel \mathsf{q}[\mathsf{p}?\{\lambda, \lambda'\}]$, since $\mathsf{p}[\mathsf{q}!\lambda] \parallel \mathsf{q}[\mathsf{p}?\{\lambda, \lambda'\}]$, and $\mathsf{p}[\mathsf{q}!\lambda] \parallel \mathsf{q}[\mathsf{p}?\lambda]$ behave the same (i.e. have the same traces).

SUB-BEHAVIOUR RELATION

Hence, we extend the expressivity of the MPST approach by resorting to the *substitution principle* that, in general, allows to replace elements in a structure or computation with other ones when the overall behaviour is unaffected. This simple principle has been variously implemented in computer science (it led, for instance, to *subtyping*). In the present setting, we dub it *sub-behaviour*. Formally, we define a pre-order relation $\leqslant$ on A-processes making the MPST and SMPS approaches equivalent and such that the following holds.

$$\forall i{\in}\{1,..,n\}\ \mathsf{P}_i \leqslant \mathsf{P}'_i \ \Rightarrow\ \mathbf{Tr}(\mathsf{p}_1[\mathsf{P}_1] \parallel \cdots \parallel \mathsf{p}_n[\mathsf{P}_n]) = \mathbf{Tr}(\mathsf{p}_1[\mathsf{P}'_1] \parallel \cdots \parallel \mathsf{p}_n[\mathsf{P}'_n]) \tag{4}$$

Definition 4.14 ($\leqslant$ on A-processes) *The preorder $\leqslant$ between A-processes is coinductively defined by*

$$[\text{sub-}\mathbf{0}]\ \frac{}{\mathbf{0} \leqslant \mathbf{0}} \qquad [\text{sub-Out}]\ \frac{\mathsf{P}_i \leqslant \mathsf{Q}_i \qquad \forall i \in I}{\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I} \leqslant \mathsf{q}!\{\lambda_i.\mathsf{Q}_i\}_{i\in I}}$$

$$[\text{sub-In}]\ \frac{\mathsf{P}_i \leqslant \mathsf{Q}_i \qquad \forall i \in I}{\mathsf{q}?\{\lambda_i.\mathsf{P}_i\}_{i\in I\cup J} \leqslant \mathsf{q}?\{\lambda_i.\mathsf{Q}_i\}_{i\in I}}$$

As previously hinted, the intuition underlying the above relation is that if $\mathsf{P} \leqslant \mathsf{Q}$ then the behaviour of a session is unaffected when a process Q is replaced by P, particularly an input process with another one that can receive more inputs than the former.

Exercise 33 *Show that the inverse implication of (4) does not hold.*

The relation $\leqslant$ can be straightforwardly extended to sessions as follows

Definition 4.15 ($\leqslant$ on sessions) *We define*

$$\mathbb{M} \leqslant \mathbb{M}' \ \textit{ if } \ \mathsf{ptp}(\mathbb{M}) = \mathsf{ptp}(\mathbb{M}') \textit{ and } \forall \mathsf{p}.\ [\mathsf{p}[\mathsf{P}] \in \mathbb{M} \textit{ and } \mathsf{p}[\mathsf{P}'] \in \mathbb{M}' \textit{ implies } \mathsf{P} \leqslant \mathsf{P}']$$

We also define

$$\mathsf{G} \triangleright \mathbb{M} \quad \textit{if} \quad \mathsf{G}\upharpoonright \textit{ is defined and } \mathbb{M} \leqslant \mathsf{G}\upharpoonright$$

Theorem 4.16 $\mathsf{G} \triangleright \mathbb{M}$ *implies that* G *is correct and complete for* $\mathbb{M}$. *Moreover,* G *is lock-free.*

The above theorem is an immediate consequence of the following equivalence between the MPST and SMPS approaches to multiparty sessions.

Proposition 4.17 (Equivalence between typability and projectability)

$$\mathsf{G} \vdash \mathbb{M} \quad \Leftrightarrow \quad \mathsf{G} \triangleright \mathbb{M}$$

Proof. See Proposition 6.7. □

Now (4) is an immediate consequence of the equivalence of the MPST and SMPS approaches.

Exercise 34 *Let* $\vdash^-$ *be the type system where [*T-Comm*] is replaced by the following rule:*

$$[\text{T-Comm}^-]\ \frac{\mathsf{G}_i \vdash^- \mathsf{p}[\mathsf{P}_i] \parallel \mathsf{q}[\mathsf{Q}_i] \parallel \mathbb{M} \qquad \mathsf{prt}(\mathsf{G}_i) \setminus \{\mathsf{p},\mathsf{q}\} = \mathsf{prt}(\mathbb{M}) \quad \forall i \in I}{\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I} \vdash^- \mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_i.\mathsf{Q}_i\}_{i\in I}] \parallel \mathbb{M}}$$

Besides, let us define

$$\mathsf{G} \triangleright^- \mathbb{M} \quad \textit{if} \quad \mathbb{M} \equiv \mathsf{G}\!\upharpoonright$$

Show that $\mathsf{G} \triangleright^- \mathbb{M} \not\Rightarrow \mathsf{G} \vdash^- \mathbb{M}$. Hint: $\mathsf{p}{\to}\mathsf{q}{:}\{\lambda_1.\mathsf{p}{\to}\mathsf{r}{:}\lambda_1, \lambda_2.\mathsf{p}{\to}\mathsf{r}{:}\lambda_2\}$. *Franco: Mariangiola diceva di non riuscire a fare questo esercizio, ma credo basti prendere come* G *quello appena indicato e come* $\mathbb{M}$ *il seguente*

$$\mathsf{p}[\mathsf{q}!\{\lambda_1.\mathsf{r}!\lambda_1, \lambda_2.\mathsf{r}!\lambda_2\}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_1, \lambda_2\}] \parallel \mathsf{r}[\mathsf{p}?\{\lambda_1, \lambda_2\}]$$

Franco: Qui ho eliminato (lasciandolo commentato nel sorgente) un esercizio che in effetti, come diceva Mariangiola, aveva un enunciato errato. Ho anche eliminato, di conseguenza, un paragrafo che ne discuteva.

Exercise 35 *Modify the definition of* $\sqcap$ *to get* $\mathsf{G} \triangleright^- \mathbb{M} \Leftrightarrow \mathsf{G} \vdash^- \mathbb{M}$.

The full sub-behaviour relation MPST formalisms usually consider the following sub-behaviour relation.

Definition 4.18 ($\leqslant^+$ on A-processes) *The preorder* $\leqslant^+$ *between processes is coinductively defined by the rules of 4.14 where Rule* [sub-Out] *is replaced by the following rule.*

$$\frac{\mathsf{P}_i \leqslant^+ \mathsf{Q}_i \qquad \forall i \in I}{\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I} \leqslant^+ \mathsf{q}!\{\lambda_i.\mathsf{Q}_i\}_{i\in I\cup J}}$$

We also define $\mathsf{G} \triangleright^+ \mathbb{M}$ *as* $\mathsf{G} \triangleright \mathbb{M}$ *in Def. 4.15, but for the use of* $\leqslant^+$ *instead of* $\leqslant$.

The intuition underlying the above rule is that an output process can be safely substituted with another process which can send less outputs than the former.

By using $\leqslant^+$ instead of $\leqslant$, Session Fidelity would be lost, as it can be shown by considering the same example discussed at the end of Subsection 4.1:

$$\mathsf{p}{\to}\mathsf{q}{:}\{\lambda_1, \lambda_2\} \triangleright^+ \mathsf{p}[\mathsf{q}!\lambda_1] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_1, \lambda_2\}]$$

Exercise 36 *Formally prove the above statement.*

Expressiveness, however, does not change, since the following holds.

Proposition 4.19 $\mathsf{G} \triangleright^{+} \mathbb{M}$ *implies* $\exists \mathsf{G}' s.t. \mathsf{G}' \triangleright \mathbb{M}$

By Proposition 4.17 we immediately get that $\mathsf{G} \triangleright^{+} \mathbb{M}$ implies $\mathbb{M}$ to be lock-free.

Also considering the above proposition, in the rest of this chapter we shall stick to $\leqslant$ as sub-behaviour relation.

5 Multi-layer Local Views

In choreographic formalisms one might be interested in different levels of abstraction of local views. It is natural to expect that the different "layers" of abstractions be related through some *refinement* relation representing some process to be a finer description of another one. In our simple choreographic formalism, such a relation can be formalised in terms of traces.

Let us introduce in our local view a further set of processes more concrete than the A-processes considered so far. They make visible the values transmitted along with the messages, as well as the computations corresponding to the internal choices of outputs. We assume that the transmitted values belong to certain given sorts and their transmission makes them substituted to variables. The variables possibly occur in the expression driving the selection in a `case` expressions.

Let `Int`, `Bool` and `String` be the (ground) sorts of integers, booleans, and strings. Elements of these sorts are referred to as *values* (`v`). We assume to have some basic elements and functions ($0, \texttt{succ}, \texttt{true}, \texttt{false}, \texttt{concat}, +, -, \texttt{and}, \texttt{or}, \ldots$) enabling to form expressions (`e`) which we can evaluate, possibly getting values (`v`). We write $\texttt{e}{\downarrow}\texttt{v}$ whenever expression `e` evaluates to value `v`. We also assume to have a denumerable set of variables ($x, y, \ldots$).

Definition 5.1 (Concrete Processes and sessions) Concrete Processes *(C-processes for short) are coinductively defined by*

$$
\begin{array}{rcl}
P & ::=^{coind} & \mathbf{0} \\
& \mid & \mathsf{p}!\lambda\langle \texttt{e} \rangle.P \\
& \mid & \mathsf{p}?\{\lambda_i(x_i).P_i\}_{i\in I} \\
& \mid & \texttt{case e of } \texttt{v}_1\texttt{->}\mathsf{p}!\lambda_1\langle \texttt{e}_1\rangle.P_1; \ldots, \texttt{v}_\texttt{n}\texttt{->}\mathsf{p}!\lambda_n\langle \texttt{e}_\texttt{n}\rangle.P_n \texttt{ other } \mathsf{p}!\lambda_{n+1}\langle \texttt{e}_{\texttt{n}+1}\rangle.P_{n+1}
\end{array}
$$

where $I \neq \emptyset$ is finite, $1 \leq n$ and $\lambda_h \neq \lambda_k$ for $h,k \leq n$ and $h \neq k$. We restrict the set of processes to the regular ones, i.e. terms having finitely many distinct subterms.
A C-session *is a session where C-processes are used instead of A-processes. We use $\mathbb{M}$ to range over C-sessions.*

We represent the operational meaning of C-sessions in terms of an LTS, in turn defined using an LTS on C-processes. It would be possible to directly define an LTS for C-sessions. We preferred however to propose a slightly different style of presentation (also following some authors in the literature).

Definition 5.2 (LTS for C-processes and C-sessions) *The LTS on C-session is defined below where the elements of Act (events) are either of the form $\mathsf{p}!\lambda\langle \texttt{v} \rangle$ or $\mathsf{p}?\lambda(\texttt{v})$ or τ.*

$$
\frac{}{\mathsf{p}!\lambda\langle \texttt{e}\rangle.P \xrightarrow{\mathsf{p}!\lambda\langle \texttt{v}\rangle} P} \quad (\texttt{e}{\downarrow}\texttt{v}) \qquad \frac{}{\mathsf{p}?\{\lambda_i(x).P_i\}_{i\in I} \xrightarrow{\mathsf{p}?\lambda_j(\texttt{v})} P_j[\texttt{v}/x]} \quad (j \in I)
$$

$$\frac{}{\texttt{case e of } \texttt{v}_1\texttt{->p!}\lambda_1\langle \texttt{e}_1\rangle.P_1;\ \vdots\ \texttt{v}_n\texttt{->p!}\lambda_n\langle \texttt{e}_n\rangle.P_n;\ \texttt{other p!}\lambda_{n+1}\langle \texttt{e}_{n+1}\rangle.P_{n+1} \xrightarrow{\tau} \texttt{p!}\lambda_k\langle \texttt{e}_k\rangle.P_k} \quad (\texttt{e}{\downarrow}\texttt{v}_k)$$

$$\frac{}{\texttt{case e of } \texttt{v}_1\texttt{->p!}\lambda_1\langle \texttt{e}_1\rangle.P_1;\ \vdots\ \texttt{v}_n\texttt{->p!}\lambda_n\langle \texttt{e}_n\rangle.P_n;\ \texttt{other p!}\lambda_{n+1}\langle \texttt{e}_{n+1}\rangle.P_{n+1} \xrightarrow{\tau} \texttt{p!}\lambda_{n+1}\langle \texttt{e}_{n+1}\rangle.P_{n+1}} \quad (\texttt{e}{\downarrow}\texttt{v} \neq \texttt{v}_i\ \forall i \in \{1,..,n\})$$

The LTS on C-sessions is defined below, where $Act = \mathscr{C} \cup \{\tau\}$.

$$\frac{P_1 \xrightarrow{\tau} P_1'}{\texttt{p}_1[P_1] \parallel \ldots \texttt{p}_n[P_n] \xrightarrow{\tau} \texttt{p}_1[P_1'] \parallel \ldots \texttt{p}_n[P_n]}$$

$$\frac{P_1 \xrightarrow{\texttt{p}_2!\lambda\langle \texttt{v}\rangle} P_1' \qquad P_2 \xrightarrow{\texttt{p}_1?\lambda(\texttt{v})} P_2'}{\texttt{p}_1[P_1] \parallel \texttt{p}_2[P_2] \parallel \ldots \texttt{p}_n[P_n] \xrightarrow{\texttt{p}_1\lambda\texttt{p}_2} \texttt{p}_1[P_1'] \parallel \texttt{p}_2[P_2'] \parallel \ldots \texttt{p}_n[P_n]}$$

Notice that the usual conditional expression `if b then` $\texttt{p!}\lambda_1\langle \texttt{e}_1\rangle.P_1$ `else` $\texttt{p!}\lambda_2\langle \texttt{e}_2\rangle.P_2$, where `b` is a boolean expression, can be considered as short for

$$\texttt{case b of true->p!}\lambda_1\langle \texttt{e}_1\rangle.P_1 \texttt{ other p!}\lambda_2\langle \texttt{e}_2\rangle.P_2$$

Example 5.3 (`Buyer, Seller and Carrier` **C-processes)**

$$P_\texttt{b} = \texttt{s!}\mathit{item}\langle\texttt{choose-item(self)}\rangle.\texttt{if satisfied(self) then s!}\mathit{buy}\langle\texttt{address(self)}\rangle\texttt{ else } P_\texttt{b}$$

$$P_\texttt{s} = \texttt{c?}\{\mathit{item}(x).P_\texttt{s},\ \mathit{buy}(y).\texttt{c!}\mathit{ship}\langle\texttt{msg}(y,\texttt{goods(self)})\rangle\} \qquad P_\texttt{c} = \texttt{s?}\{\mathit{ship}(x)\}$$

In the above C-processes for the `Buyer, Seller and Carrier`*, the procedure* `choose-item` *chooses an item to buy depending on the internal state of the process. Also the predicate* `satisfied` *depends on the internal state of the process. By* `msg` *we denote instead a procedure returning a message for the carrier depending on the received address and the ordered goods.* ◇

As recalled above, the LTS on C-processes is used as a first step for defining the LTS on sessions. Both the LTSs have τ as a possible event, which is an unobservable internal action. It is used to make explicit the choice event between different branches in `case` statements. The observable events for C-processes show the exchanged values, while the observable events for C-sessions are those in $\mathscr{C}$. We aim in fact, at abstracting from the actual values exchanged among the participants in our observations on a session, since we are interested in communication properties like lock freedom. Such properties depend just on the effects of the exchanged values (i.e. the branch selected by a `case` expression). This choice also enables us to define a refinement correspondence between A-sessions and C-sessions.

SESSION REFINEMENT

We intend to look at C-processes as concrete realizations of what A-processes represent. In particular, we intend to see a nondeterministic choice among outputs as an abstract representation of a `case` expression. The refinement correspondence between C-session and A-session can be formally defined in terms of a type system, where A-sessions are looked at as types (in "classical" MPST formalisms A-sessions are dubbed *local types*). The aim of the refinement relation is guaranteeing the same properties of an A-sessions $\mathbb{M}$ to any C-session $\mathbb{M}$ refining $\mathbb{M}$, and possibly more. We notice in fact that in C-sessions a further cause of error is present: an expression `e` in a `case` expression might not have a value, or might have a value not of the `Bool` sort.

Definition 5.4 (A-processes as types) *We assume given a function* TL *associating sorts to messages. We also assume* S *to range over the available sorts and* Γ *to range over finite sets of associations between variables and sorts. We recall that the relation* $\leqslant$ *below is as in Def.4.14.*

$$[\text{INACT}]\ \frac{}{\Gamma \vdash \mathbf{0} : \mathbf{0}} \qquad [\leqslant]\ \frac{\Gamma \vdash P : \mathsf{P} \quad \mathsf{P} \leqslant \mathsf{P}'}{\Gamma \vdash P : \mathsf{P}'}$$

$$[\text{IN-CHOICE}]\ \frac{\Gamma, x_i : \text{TL}(\lambda_i) \vdash P_i : \mathsf{P}_i \quad \forall i \in I}{\Gamma \vdash \mathsf{p}?\{\lambda_i(x_i).P_i\} : \mathsf{p}?\{\lambda_i.\mathsf{P}_i\}} \qquad [\text{OUT}]\ \frac{\Gamma \vdash \mathtt{e} : \text{TL}(\lambda) \quad \Gamma \vdash P : \mathsf{P}}{\Gamma \vdash \mathsf{p}!\lambda\langle \mathtt{e}\rangle.P : \mathsf{p}!\lambda.\mathsf{P}}$$

$$[\mathtt{case}]\ \frac{\begin{array}{c}\Gamma \vdash \mathtt{e} : \mathtt{S} \quad \Gamma \vdash \mathtt{v_i} : \mathtt{S}\ \ (\forall i \in \{1,..,n\}) \\ \Gamma \vdash \mathtt{e_i} : \text{TL}(\lambda_i) \quad \Gamma \vdash P_i : \mathsf{P}_i\ \ (\forall i \in \{1,..,n+1\})\end{array}}{\begin{array}{l}\mathtt{case\ e\ \ of\ \ v_1\text{->}}\mathsf{p}!\lambda_1\langle \mathtt{e_1}\rangle.P_1; \\ \qquad\qquad \vdots \\ \qquad\qquad \mathtt{v_n\text{->}}\mathsf{p}!\lambda_n\langle \mathtt{e_n}\rangle.P_n; \\ \qquad\qquad \mathtt{other}\ \ \mathsf{p}!\lambda_{n+1}\langle \mathtt{e_{n+1}}\rangle.P_{n+1}\end{array} : \mathsf{p}!\{\lambda_i.\mathsf{P}_i\}_{i\in\{1,..,n+1\}}}$$

The above is inspired by the usual type systems for processes and local types in MPST formalisms. Notice the above type system rules out any infinite immediate nestings of `case` expressions. Moreover, the single line in $[\leqslant]$ indicates that only a finite number of consecutive applications of such a rule are allowed.

We can now define a type system for sessions. (The same symbol $\vdash$ is used for the sake of readability.) Notice that $\vdash P : \mathsf{P}$ is short for $\emptyset \vdash P : \mathsf{P}$.

$$\frac{\vdash P_i : \mathsf{P}_i \qquad (\forall i \in \{1,..,n\})}{\vdash \mathsf{p}_1[P_1] \parallel \ldots \mathsf{p}_n[P_n] : \mathsf{p}_1[\mathsf{P}_1] \parallel \ldots \mathsf{p}_n[\mathsf{P}_n]}$$

Exercise 37 *Provide a derivation for*

$$\vdash \mathsf{b}[P_\mathsf{b}] \parallel \mathsf{s}[P_\mathsf{s}] \parallel \mathsf{c}[P_\mathsf{c}] \ :\ \mathsf{b}[\mathsf{P}_{\mathsf{bsc}}] \parallel \mathsf{s}[\mathsf{Q}_{\mathsf{bsc}}] \parallel \mathsf{c}[\mathsf{s}?\mathit{ship}]$$

where P_b, P_s *and* P_c *are as in Example 5.3 and* $\mathsf{P}_{\mathsf{bsc}}$, $\mathsf{Q}_{\mathsf{bsc}}$ *as in Example 3.5.*

We also define

$$\vdash M : \mathbb{M} \triangleleft \mathsf{G} \qquad \text{if} \qquad \vdash M : \mathbb{M} \quad \text{and} \quad \mathsf{G} \triangleright \mathbb{M}$$

The reading of the term "session refinement" is not univocal, and can vary according to the needs it is used for. In general, a session M refines $\mathbb{M}$ if what M does is a "more detailed" specification of

what $\mathbb{M}$ can more abstractedly do. In our particular setting, any trace of M is also a trace of $\mathbb{M}$ if we abstract away the τ actions. (We shall briefly discuss later on how the notion itself of refinement can be "refined".)

Let us now introduce a slightly different LTS for A-processes enabling to compare their traces to those of the C-processes. The new LTS essentially consists of decoupling the internal choice of the label to be sent from the actual sending action; as the internal choice is unobservable, it is labelled by τ.

Definition 5.5 (The $\hookrightarrow$ LTS for A-processes)

$$\frac{}{\mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{P}_i\}_{i\in I}] \parallel \mathbb{M} \overset{\tau}{\hookrightarrow} \mathsf{p}[\mathsf{q}!\lambda_k.\mathsf{P}_k] \parallel \mathbb{M}} \ (k \in I)$$

$$\frac{}{\mathsf{p}[\mathsf{q}!\lambda.\mathsf{P}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M} \overset{\mathsf{p}\lambda\mathsf{q}}{\hookrightarrow} \mathsf{p}[\mathsf{P}] \parallel \mathsf{q}[Q_k] \parallel \mathbb{M}} \ (\lambda = \lambda_k, k \in J)$$

We denote the set of traces for the above LTS by $\mathbf{Tr}^{\hookrightarrow}$.

With respect to $\hookrightarrow$, an *actual communication mismatch* is a session like

$$\mathsf{p}[\mathsf{q}!\lambda.\mathsf{P}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M}$$

where $\lambda \neq \lambda_j$ for all $j \in J$.

Exercise 38 *Define the set of traces* $\mathbf{Tr}^{\hookrightarrow}(\mathbb{M})$ *out of the session* $\mathbb{M}$ *with the above LTS. Notice that now traces are sequences on* $\mathscr{C} \cup \{\tau\}$.

One of the communication properties we aim at ensuring is the absence of *communication mismatches* during any possible evolution of a session. It is worth remarking that the LTS of Definition 3.3 and the LTS above do produce different set of traces.

Exercise 39 *Show that the set of traces of the two LTS for sessions are distinct even by disregarding* τ.

Exercise 40 *Let* $\mathbb{M}$ *be a session that cannot evolve in a communication mismatch using either of the two LTSs. Then, if we disregard* τ, *the traces we get are the same.*

Lemma 5.6 $\vdash M : \mathbb{M}$ *and* $M \overset{\Lambda}{\to} M'$ *imply* $\vdash M' : \mathbb{M}'$ *and* $\mathbb{M} \overset{\Lambda}{\hookrightarrow} \mathbb{M}'$ *for some* $\mathbb{M}'$, *where* $\Lambda \in \mathscr{C} \cup \{\tau\}$.

Given a set of traces $T \subseteq (\mathscr{C} \cup \{\tau\})^*$, we define $T_{\not\tau}$ as the set of traces in T in which any occurrence of τ has been taken out. The following Proposition is easily proved using Lemma 5.6.

Proposition 5.7 *Let* $\vdash M : \mathbb{M}$.

i) $\mathbf{Tr}(M) \subseteq \mathbf{Tr}^{\hookrightarrow}(\mathbb{M})$

ii) $\mathbf{Tr}^{\hookrightarrow}_{\not\tau}(\mathbb{M}) = \mathbf{Tr}(\mathbb{M})$

iii) $\mathbf{Tr}_{\not\tau}(M) \subseteq \mathbf{Tr}(\mathbb{M})$

Exercise 41 *Find* M *and* $\mathbb{M}$ *such that* $\vdash M : \mathbb{M}$ *and* $\mathbf{Tr}_{\not\tau}(M) \not\supseteq \mathbf{Tr}(\mathbb{M})$.

Proposition 5.8 *Let* $\vdash M : \mathbb{M} \triangleleft \mathsf{G}$ *(or, equivalently,* $\vdash M : \mathbb{M}$ *and* $\mathsf{G} \vdash \mathbb{M}$*). Then* M *is lock-free.*

Exercise 42 *Assume we intend to investigate also the behaviour of untypable sessions. We hence explicitely represent an error state, say* **err**, *and extend Definition 5.5 with the following further rule*

$$\frac{}{\mathsf{p}[\mathsf{q}!\lambda.\mathsf{P}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_j.\mathsf{Q}_j\}_{j\in J}] \parallel \mathbb{M} \overset{\mathsf{p}\lambda\mathsf{q}}{\hookrightarrow} \mathbf{err}} \ (\lambda = \lambda_k, k \notin J)$$

Define the set of traces for the resulting LTS and compare them with the traces in the other approaches. Notice that in a synchronous scenario, the event of a sender selecting a message the receiver cannot accept (when $k \notin J$ *above) is not in general as harmful as in asynchronous settings because there the output is not a blocking action.*

Simulation and Bisimulation In our simple coreographic setting, we have modeled the notions of behaviour and of *refinement* using traces and trace inclusion. However, trace semantics is too coarse to model in general the behaviour of processes in concurrency theory. Instead, the notions of *Simulation* and *Bisimulation* have been introduced for the formal investigation of *process refinement and equivalence*. A natural question is whether we should have taken the same approach in the case of SMST.

To discuss this point, we briefly recall the definitions of these concepts. Let Proc be a set of processes whose behaviour is represented using an LTS $(\mathsf{Proc}, \mathsf{Act}, \rightarrow)$ where the observable events are elements of Act. If $\mathscr{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ we abbreviate $(P,Q) \in \mathscr{R}$ by $P \mathscr{R} Q$.

Definition 5.9 (Simulation, Bisimulation) *A relation $\mathscr{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ is a* simulation *if $P \mathscr{R} Q$ implies that for all $a \in \mathsf{Act}$*

$$P \xrightarrow{a} P' \;\Rightarrow \exists Q'.\, Q \xrightarrow{a} Q' \;\;\&\;\; P' \mathscr{R} Q'$$

Then we define $P \sqsubseteq Q$ if there exists a simulation $\mathscr{R}$ such that $P \mathscr{R} Q$.

A relation $\mathscr{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ is a bisimulation *if $P \mathscr{R} Q$ implies that for all $a \in \mathsf{Act}$*

$$\begin{array}{lll} P \xrightarrow{a} P' & \Rightarrow & \exists Q'.\, Q \xrightarrow{a} Q' \;\;\&\;\; P' \mathscr{R} Q' \quad \textit{and} \\ Q \xrightarrow{a} Q' & \Rightarrow & \exists P'.\, P \xrightarrow{a} P' \;\;\&\;\; P' \mathscr{R} Q' \end{array}$$

Then we define $P \sim Q$ if there exists a bisimulation $\mathscr{R}$ such that $P \mathscr{R} Q$.

Exercise 43 *Show that $\sqsubseteq$ and $\sim$ are themselves, respectively, a simulation and a bisimulation (in particular, the greatest simulation and bisimulation).*

Let $\mathbf{Tr}(P)$ be the set of traces in Act^∞ out of P; if $P \sqsubseteq Q$ then $\mathbf{Tr}(P) \subseteq \mathbf{Tr}(Q)$ and similarly if $P \sim Q$ then $\mathbf{Tr}(P) = \mathbf{Tr}(Q)$; however, the converse implication does not hold in general.

Exercise 44 *Justify the last statement.* Hint: consider an LTS whose transitions have the shapes

If $P \sim Q$ then both $P \sqsubseteq Q$ and $Q \sqsubseteq P$, but the opposite implication does not hold.

Exercise 45 *Justify the last statement.* Hint: consider an LTS whose transitions have the shapes

Simulation and trace inclusion coincide in case the LTS is *deterministic*; the same holds for trace equivalence and bisimulation.

Definition 5.10 (Deterministic LTS) *An LTS $(\mathsf{Proc}, \mathsf{Act}, \rightarrow)$ is* deterministic *if for all $a \in \mathsf{Act}$ and $P, Q, R \in \mathsf{Proc}$*

$$P \xrightarrow{a} Q \quad \textit{and} \quad P \xrightarrow{a} R \quad \Rightarrow \quad Q = R$$

Exercise 46 *Prove that if the LTS is deterministic then, for any $s \in \mathsf{Act}^*$, $t \in \mathsf{Act}^\infty$ and $a \in \mathsf{Act}$,*

i) $P \xrightarrow{s} Q \;\;\&\;\; P \xrightarrow{s} R \;\Rightarrow\; Q = R$

ii) $a \cdot t \in \mathbf{Tr}(P) \;\;\&\;\; P \xrightarrow{a} P' \;\Rightarrow\; t \in \mathbf{Tr}(P')$

Hint: *(i) can be proved by induction over the length of the (finite) s; then use (i) in the proof of (ii).*

Proposition 5.11 *Let $(Proc, Act, \to)$ be a deterministic LTS then for all $P, Q \in Proc$*

i) $\mathbf{Tr}(P) \subseteq \mathbf{Tr}(Q) \;\Rightarrow\; P \sqsubseteq Q$

ii) $\mathbf{Tr}(P) = \mathbf{Tr}(Q) \;\Rightarrow\; P \sim Q$

Proof. We prove (i), as the proof of (ii) is similar. Since traces are elements of Act^∞, hence possibly infinite, we reason by coinduction over the definition of Act^∞.

Let $P \sqsubseteq Q$ and $P \xrightarrow{a} P'$; let $t \in \mathbf{Tr}(P')$ be arbitrary: then $a \cdot t \in \mathbf{Tr}(P) \subseteq \mathbf{Tr}(Q)$, which implies that there is (a unique) Q' such that $Q \xrightarrow{a} Q'$. By the hypothesis that the LTS is deterministic, (ii) of Exercise 46 applies, hence $t \in \mathbf{Tr}(Q')$; then by the coinductive hypothesis we conclude $P' \sqsubseteq Q'$ as required. □

Exercise 47 *Check that the LTSs for global types and A-sessions are both deterministic.*

6 Some Proofs

To avoid interruptions in the main text, in this section we collect proofs of some results, or just references to them.

Lemma 6.1 ([3]) *If* $\mathsf{G} \vdash \mathbb{M}$, *then* $\mathsf{prt}(\mathsf{G}) = \mathsf{prt}(\mathbb{M})$.

Lemma 6.2 ([3]) *Let* P, *V and W be such that* $\mathsf{P} \leqslant V$ *and* $\mathsf{P} \leqslant W$. *Then* $V \sqcap W$ *is defined and* $\mathsf{P} \leqslant V \sqcap W$.

Lemma 6.3 ([3]) *If* $\mathsf{G} \vdash \mathbb{M}$ *and* $\mathsf{G}\upharpoonright\mathsf{p}$ *is defined, then* $\mathbb{M} \equiv \mathsf{p}[\mathsf{P}] \parallel \mathbb{M}'$ *and* $\mathsf{P} \leqslant \mathsf{G}\upharpoonright\mathsf{p}$.

We say that a global type G is *inhabited* if there exists a multiparty session $\mathbb{M}$ such that $\mathsf{G} \vdash \mathbb{M}$.

Lemma 6.4 *Given a global type* G, *inhabitation of* G *implies its projectability.*

Proof. Let G be a global type such that $\mathsf{G} \vdash \mathbb{M}$ for some $\mathbb{M}$. By Lemma 6.1 we have that $\mathsf{prt}(\mathsf{G}) = \mathsf{prt}(\mathbb{M})$.

To show that G is projectable, let consider $\mathsf{p} \in \mathsf{prt}(\mathsf{G})$ in order to get $\mathsf{G}\upharpoonright\mathsf{p}$ defined. We proceed by coinduction on the derivation of $\mathsf{G} \vdash \mathbb{M}$.

Axiom [T-End]. Immediate.

Rule [T-Comm]. In this case the last applied rule has the shape

$$\frac{\mathsf{G}_i \vdash \mathsf{r}[\mathsf{R}_i] \parallel \mathsf{s}[\mathsf{S}_i] \parallel \mathsf{p}[\mathsf{P}] \parallel \mathbb{M}' \qquad \mathsf{prt}(\mathsf{G}_i) \setminus \{\mathsf{r},\mathsf{s}\} = \mathsf{prt}(\mathsf{p}[\mathsf{P}] \parallel \mathbb{M}') \quad \forall i \in I \subseteq J}{\mathsf{G} \vdash \mathsf{r}[\mathsf{s}!\{\lambda_i.\mathsf{R}_i\}_{i\in I}] \parallel \mathsf{s}[\mathsf{r}?\{\lambda_j.\mathsf{S}_j\}_{j\in J}] \parallel \mathsf{p}[\mathsf{P}] \parallel \mathbb{M}'}$$

where $\mathsf{G} = \mathsf{r} \to \mathsf{s} : \{\lambda_i.\mathsf{G}_i\}_{i\in I}$. We observe that, for each $i \in I$, G_i is inhabited. Moreover, by coinduction, $\mathsf{G}_i\upharpoonright\mathsf{p}$ is defined for each $i \in I$. If $\mathsf{p} = \mathsf{r}$, by definition of projection we have that $\mathsf{G}\upharpoonright\mathsf{p} = \mathsf{s}!\{\lambda_i.\mathsf{G}_i\upharpoonright\mathsf{p}\}_{i\in I}$. If $\mathsf{p} = \mathsf{s}$, by definition of projection we have that $\mathsf{G}\upharpoonright\mathsf{p} = \mathsf{r}?\{\lambda_i.\mathsf{G}_i\upharpoonright\mathsf{p}\}_{i\in I}$. So in both cases $\mathsf{G}\upharpoonright\mathsf{p}$ is defined. If $\mathsf{p} \notin \{\mathsf{r},\mathsf{s}\}$, by Lemma 6.3, $\mathsf{P} \leqslant \mathsf{G}_i\upharpoonright\mathsf{p}$ for each $i \in I$. We can hence recur to Lemma 6.2 to conclude that $\bigsqcap_{i\in I}\mathsf{G}_i\upharpoonright\mathsf{p}$ is defined. The thesis now follows by definition of projection, since in the present case $\mathsf{G}\upharpoonright\mathsf{p} = \bigsqcap_{i\in I}\mathsf{G}_i\upharpoonright\mathsf{p}$. □

The meaning of the preorder on processes is exploited in the following lemma.

Lemma 6.5 ([3]) *If* $\mathsf{G} \vdash \mathsf{p}[\mathsf{Q}] \parallel \mathbb{M}$ *and* $\mathsf{P} \leqslant \mathsf{Q}$, *then* $\mathsf{G} \vdash \mathsf{p}[\mathsf{P}] \parallel \mathbb{M}$.

Lemma 6.6 ([3]) *Let* P *and* Q *be such that* $\mathsf{P} \sqcap \mathsf{Q}$ *is defined. Then* $\mathsf{P} \sqcap \mathsf{Q} \leqslant \mathsf{P}$.

By Lemmas 6.2 and 6.6, the merge $\mathsf{P} \sqcap \mathsf{Q}$, when defined, is the meet of P and Q w.r.t. $\leqslant$.

Proposition 6.7

i) $\mathsf{G} \vdash \mathbb{M}$ *and* $\mathsf{p}[\mathsf{P}] \in \mathbb{M}$ *imply* $\mathsf{G}\upharpoonright\mathsf{p}$ *is defined and* $\mathsf{P} \leqslant \mathsf{G}\upharpoonright\mathsf{p}$

ii) G *projectable implies* $\mathsf{G} \vdash \mathsf{G}\upharpoonright$

Proof. (i) By 6.4 and 6.3.
(ii) If $\mathsf{G} = \mathsf{End}$, then $\mathsf{G} \vdash \mathbb{M}$ for all $\mathbb{M} \equiv \mathsf{p}[\mathbf{0}]$. Otherwise, by recalling that we defined

$$\mathsf{G}\upharpoonright = \Big\|_{\mathsf{p}\in\mathsf{prt}(\mathsf{G})} \mathsf{p}[\mathsf{G}\upharpoonright\mathsf{p}],$$

we proceed to show that $\mathsf{G} \vdash \mathsf{G}\upharpoonright$ by coinduction on G. We observe that

$$\begin{array}{l}(\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I})\upharpoonright = \\ \mathsf{p}[\mathsf{q}!\{\lambda_i.\mathsf{G}_i\upharpoonright\mathsf{p}\}_{i\in I}] \parallel \mathsf{q}[\mathsf{p}?\{\lambda_i.\mathsf{G}_i\upharpoonright\mathsf{q}\}_{i\in I}] \Big\|_{\mathsf{r}\in\mathsf{prt}(\mathsf{G})\setminus\{\mathsf{p},\mathsf{q}\}} \mathsf{r}[\mathsf{G}\upharpoonright\mathsf{r}]\end{array}$$

where, by the assumption that G is projectable and by definition of projection, for each $\mathsf{r} \in \mathsf{prt}(\mathsf{G}) \setminus \{\mathsf{p},\mathsf{q}\}$,

$$\mathsf{G}\upharpoonright\mathsf{r} = \bigsqcap_{i\in I} \mathsf{G}_i\upharpoonright\mathsf{r}$$

By coinduction we have that, for all $i \in I$, $\mathsf{G}_i \vdash \mathsf{G}_i\upharpoonright$, where by definition,

$$\mathsf{G}_i\upharpoonright \equiv \mathsf{p}[\mathsf{G}_i\upharpoonright\mathsf{p}] \parallel \mathsf{q}[\mathsf{G}_i\upharpoonright\mathsf{q}] \Big\|_{\mathsf{r}\in\mathsf{prt}(\mathsf{G}_i)\setminus\{\mathsf{p},\mathsf{q}\}} \mathsf{r}[\mathsf{G}_i\upharpoonright\mathsf{r}]$$

Now, by Lemma 6.6 we have that, for all $i \in I$,

$$\mathsf{G}\upharpoonright\mathsf{r} = \bigsqcap_{i\in I} \mathsf{G}_i\upharpoonright\mathsf{r} \leqslant \mathsf{G}_i\upharpoonright\mathsf{r}$$

We can hence recur to Lemma 6.5 in order to obtain that, for all $i \in I$,

$$\mathsf{G}_i \vdash \mathsf{p}[\mathsf{G}_i\upharpoonright\mathsf{p}] \parallel \mathsf{q}[\mathsf{G}_i\upharpoonright\mathsf{q}] \Big\|_{\mathsf{r}\in\mathsf{prt}(\mathsf{G}_i)\setminus\{\mathsf{p},\mathsf{q}\}} \mathsf{r}[\mathsf{G}\upharpoonright\mathsf{r}]$$

By Lemma 6.1

$$\mathsf{prt}(\mathsf{G}_i) = \mathsf{prt}(\mathsf{p}[\mathsf{G}_i\upharpoonright\mathsf{p}] \parallel \mathsf{q}[\mathsf{G}_i\upharpoonright\mathsf{q}] \Big\|_{\mathsf{r}\in\mathsf{prt}(\mathsf{G}_i)\setminus\{\mathsf{p},\mathsf{q}\}} \mathsf{r}[\mathsf{G}\upharpoonright\mathsf{r}])$$

which implies $\mathsf{prt}(\mathsf{G}_i) \setminus \{\mathsf{p},\mathsf{q}\} = \mathsf{prt}(\Big\|_{\mathsf{r}\in\mathsf{prt}(\mathsf{G}_i)\setminus\{\mathsf{p},\mathsf{q}\}} \mathsf{r}[\mathsf{G}\upharpoonright\mathsf{r}])$ for all $i \in I$. It is now possible to use Rule [T-Comm] in order to get $\mathsf{G} \vdash \mathsf{G}\upharpoonright$. □

7 Further Readings

In this section, we report some references to the literature and suggest papers and textbooks to widen the knowledge of the subject. We provide just a few pointers inside a large body of research activity carried on in the last decades, which is still active and growing.

As recalled in the Introduction, both Session Types and MultiParty Session Types have been inspired by the π-calculus and its typing systems, for which the reader might consult the comprehensive textbook [15]. Session Types, originally introduced in [9, 17, 8] for a dialect of the π-calculus, are fully treated in the book [7]. An introductory treatment can instead be found in [6].

The basic reference for MultiParty Session Types is [11] (full version of [10]), which is fairly technical; among the introductory presentations of MPST is [18]. Simple MultiParty Sessions have been introduced in [5], stemming from ideas of Dagnino; further references on (variants of) the STMS system can be recovered from [3].

The paper [16] illustrates some limitations of the MPST approach, showing that some perfectly consistent typings of the participants are not obtainable by projection of any global type. In [3], projectable global types are characterised as certain bounded or balanced types, which, however, do not include the case in Example 4.11, among others.

References

[1] *Haskell home page*. http://haskell.org.

[2] Franco Barbanera, Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2022): *Open Compliance in Multiparty Sessions*. In Silvia Lizeth Tapia Tarifa & José Proença, editors: *Formal Aspects of Component Software - 18th International Conference, FACS 2022, Virtual Event, November 10-11, 2022, Proceedings*, *Lecture Notes in Computer Science* 13712, Springer, pp. 222–243, doi:10.1007/978-3-031-20872-0_13.

[3] Franco Barbanera, Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2024): *Un-projectable Global Types for Multiparty Sessions*. In Alessandro Bruni, Alberto Momigliano, Matteo Pradella & Matteo Rossi, editors: *PPDP*, ACM Press, pp. 15:1–15:13, doi:https://doi.org/10.1145/3678232.3678245.

[4] Francesco Dagnino, Paola Giannini & Mariangiola Dezani-Ciancaglini (2021): *Deconfined Global Types for Asynchronous Sessions*. In Ferruccio Damiani & Ornela Dardha, editors: *Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, *Lecture Notes in Computer Science* 12717, Springer, pp. 41–60, doi:10.1007/978-3-030-78142-2_3.

[5] Francesco Dagnino, Paola Giannini & Mariangiola Dezani-Ciancaglini (2021): *Deconfined global types for asynchronous sessions*. *CoRR* abs/2111.11984, doi:10.48550/arXiv.2111.11984.

[6] Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2009): *Sessions and Session Types: An Overview*. In Cosimo Laneve & Jianwen Su, editors: *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers*, *Lecture Notes in Computer Science* 6194, Springer, pp. 1–28, doi:10.1007/978-3-642-14458-5_1.

[7] Simon J. Gay & Vasco T. Vasconcelos (2025): *Session Types*. Cambridge University Press.

[8] K. Honda, V. T. Vasconcelos & M. Kubo (1998): *Language Primitives and Type Disciplines for Structured Communication-based Programming*. In Chris Hankin, editor: *ESOP*, *LNCS* 1381, Springer, pp. 22–138, doi:10.1007/BFb0053567.

[9] Kohei Honda (1993): *Types for Dyadic Interaction*. In Eike Best, editor: *CONCUR'93*, *LNCS* 715, Springer, pp. 509–523.

[10] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In George C. Necula & Philip Wadler, editors: *POPL*, ACM Press, pp. 273–284, doi:10.1145/1328897.1328472.

[11] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty asynchronous session types*. *Journal of the ACM* 63(1), pp. 9:1–9:67, doi:10.1145/2827695.

[12] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, Part I*. *Information and Computation* 100(1).

[13] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, Part II*. *Information and Computation* 100(1).

[14] Davide Sangiorgi (2012): *Introduction to Bisimulation and Coinduction*. Cambridge University Press, Cambridge, UK.

[15] Davide Sangiorgi & David Walker (2001): *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press.

[16] Alceste Scalas & Nobuko Yoshida (2019): *Less is more: multiparty session types revisited*. *PACMPL* 3(POPL), pp. 30:1–30:29.

[17] Kaku Takeuchi, Kohei Honda & Makoto Kubo (1994): *An Interaction-based Language and its Typing System*. In C. Halatsis, D. Maritsas, G. Philokyprou & S. Theodoridis, editors: *PARLE'94*, *LNCS* 817, Springer, pp. 398–413.

[18] Nobuko Yoshida & Lorenzo Gheri (2020): *A Very Gentle Introduction to Multiparty Session Types*. In Dang Van Hung & Meenakshi D'Souza, editors: *Distributed Computing and Internet Technology - 16th*

International Conference, ICDCIT 2020, Bhubaneswar, India, January 9-12, 2020, Proceedings, *Lecture Notes in Computer Science* 11969, Springer, pp. 73–93, doi:10.1007/978-3-030-36987-3_5.