Multiparty Session Types Within a Canonical Binary Theory, and Beyond

Luís Caires, Jorge A. Pérez

Multiparty Session Types Within A Canonical Binary Theory, and Beyond

Luís Caires[1] and Jorge A. Pérez[2]

[1] NOVA LINCS - Universidade NOVA de Lisboa, Portugal
[2] University of Groningen, The Netherlands

Abstract. A widespread approach to software service analysis uses *session types*. Very different type theories for *binary* and *multiparty* protocols have been developed; establishing precise connections between them remains an open problem. We present the first formal relation between two existing theories of binary and multiparty session types: a binary system rooted in linear logic, and a multiparty system based on automata theory. Our results enable the analysis of multiparty protocols using a (much simpler) type theory for binary protocols, ensuring protocol fidelity and deadlock-freedom. As an application, we offer the first theory of multiparty session types with *behavioral genericity*. This theory is natural and powerful; its analysis techniques reuse results for binary session types.

1 Introduction

The purpose of this paper is to demonstrate, in a precise technical sense, how an expressive and extensible theory of multiparty systems can be extracted from a basic theory for binary sessions, thus developing the first formal connection between multiparty and binary session types. Our approach relies on a theory of binary session types rooted in linear logic and on *medium processes* that capture the behavior of global types.

Relating the global behavior of a distributed system and the components that implement it is a challenging problem in many scenarios. This problem is also important in the analysis of software services, where the focus is on message-passing programs with advanced forms of concurrency and distribution. Within language-based techniques, notable approaches include *interface contracts* (cf. [8]) and *behavioral types* [15]. Our interest is in the latter: by classifying behaviors (rather than values), behavioral types abstract structured protocols and enforce disciplined communication exchanges.

Session types [13,14] are a much-studied class of behavioral types. They organize multiparty protocols as *sessions*, basic units of structured conversations. Several session typed frameworks have been developed (see [15] for an overview). This diversity makes it hard to compare their associated techniques, and hinders the much desirable transfer of techniques between different typed models.

In this paper, we formally relate two distinct typed models for structured communications. By relying on a type theory of binary sessions rooted in linear logic [5], we establish natural bridges between typed models for *binary* and *multiparty* sessions [13,14]. Our results reveal logically motivated justifications for key concepts in typed models of global/local behaviors, and enable the transfer of reasoning techniques from binary to

multiparty sessions. In fact, our approach naturally enables us to define the first model of multiparty session types with *parametric polymorphism*, which in our setting means *behavioral genericity* (i.e., passing first-class behavioral interfaces in messages), not just datatype genericity. This new model is very powerful; we equip it with analysis techniques for behavioral genericity by reusing results for binary session types [4].

Binary protocols [13] involve two partners, each abstracted by a behavioral type; correct interactions rely on *compatibility*, i.e., when one partner performs an action, the other performs a complementary one. Multiparty protocols may involve more than two partners: there is a global specification to which all of them, from their local perspectives, should adhere. In multiparty session types [14,12], these visions are described by a *global type* and *local types*, respectively; a *projection function* relates the two. Previous research shows that type systems for multiparty protocols have a more involved theory than binary ones. For instance, the analysis of deadlock-freedom in multiparty protocols is challenging [10], and certainly harder than for binary protocols.

The question is then: *could multiparty session types be reduced into binary ones?* Defining such a reduction is far from trivial, as it should satisfy at least two requirements. First, the resulting collection of binary interactions must preserve crucial *sequencing information* among multiparty exchanges. Second, it should avoid *undesirable behaviors*: synchronization errors, deadlocks, non-terminating reductions.

This paper answers the above question in the affirmative. We tightly relate: (i) a standard theory of multiparty session types [14,12], and (ii) the theory of deadlock-free binary session types proposed in [5]. The key device in our approach is the *medium process* of a multiparty protocol.

Given a global type G, its medium process $\mathsf{M}[\![G]\!]$ is an entity that mediates in all communications. Therefore, $\mathsf{M}[\![G]\!]$ extracts the semantics of G, uniformly capturing its sequencing information. Process $\mathsf{M}[\![G]\!]$ is meant to interact with well-typed implementations for all participants declared in G. This way, for instance, given the global type $G = \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}$ (i.e., a labeled, directed exchange from $\mathtt{p}$ to $\mathtt{q}$, indexed by I, which precedes execution of a protocol G_i), its medium $\mathsf{M}[\![G]\!]$ first receives a label l_j and message of type U_j sent by $\mathtt{p}$'s implementation (with $j \in I$); then, it forwards these two objects to $\mathtt{q}$'s implementation; lastly, it executes process $\mathsf{M}[\![G_j]\!]$.

Interestingly, our medium-based approach applies to global types with name passing, delegation, and parallel composition. To fully characterize a global type G, we determine the conditions under which $\mathsf{M}[\![G]\!]$ may be well-typed using binary session types, with respect to its local types. A key ingredient here is the theory for binary session types introduced in [5]. Due to their logical foundations, typability in [5] entails: *fidelity* (protocols are respected), *safety* (absence of communication errors), *deadlock-freedom* (processes do not get stuck), and *termination* (infinite internal behavior is ruled out). Most relevant for our approach is deadlock-freedom, not directly ensured by alternative type systems.

Here we present an analysis of multiparty session types using a theory of binary session types, ensuring fidelity and deadlock-freedom. Our *technical contributions* are:

- *Characterization results* relating (a) a global type that is well-formed (correct projectability) and (b) typability of its medium using binary session types (Thms. 4-5).

- *Operational correspondence results* relating (a) the behavior of a global type and (b) the behavior of its medium (instrumented in a natural way) composed with well-typed implementations for each local type (Thm. 7). These results confirm that our analysis does not introduce extra sequentiality in protocols.
- A proof that *behavioral transformations* of global types [6] can be justified by typed equalities for binary sessions [19] expressed at the level of mediums (Thm. 6). This result offers a deep semantic justification of structural identities on global types, such as those capturing parallelism via interleaving of causally independent exchanges.
- *Transfer of techniques* from binary to multiparty protocols. We define the first theory of multiparty session types with *behavioral genericity*; its analysis techniques reuse the binary session type theory with parametric polymorphism given in [4].

Our results define the first formal relation between multiparty and binary session types. They highlight the fundamental character of the notions involved, since they can be independently explained by communicating automata (cf. [12]) and linear logic (cf. [5]).

Next, we collect definitions on multiparty sessions [14,12] and binary sessions [5]. Our technical contributions are reported in § 3. In §4 we illustrate these contributions by means of an example that features non-trivial forms of replication and sharing. In § 5 we introduce multiparty session types with behavioral genericity, and in § 6 we illustrate our approach in the analysis of a multiparty protocol. § 7 concludes and discusses related works.

2 Preliminaries: Binary and Multiparty Session Types

Binary Session Types. We build upon the theory of binary session types of [5,21], based on an interpretation of session types as linear logic propositions. We assume no background on linear logic from the reader; we refer to [5] for further details.

The Process Model. We define a synchronous π-calculus [20] with forwarding and n-ary labeled choice. We use $l_1, l_2, \ldots$ to range over *labels*. Given an infinite set Λ of *names* (x, y, z, u, v), the set of *processes* (P, Q, R) is defined by

$$\begin{array}{lllllll} P ::= \mathbf{0} & \mid P \mid Q & \mid (\boldsymbol{\nu} y)P & \mid \overline{x}\, y.P & \mid x(y).P & \mid\ !x(y).P & \mid \\ \quad x \triangleleft l_i; P & \mid x \triangleright \{l_i : P_i\}_{i \in I} & \mid [x \leftrightarrow y] & & & & \end{array}$$

Operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition), and $(\boldsymbol{\nu} y)P$ (restriction) are standard. We have $\overline{x}\, y.P$ (send y on x, proceed as P), $x(y).P$ (receive a z on x, proceed as P with y replaced by z), and the replicated input $!x(y).P$. Operators $x \triangleleft l; P$ and $x \triangleright \{l_i{:}P_i\}_{i \in I}$ define labeled choice [13]. Forwarding $[x \leftrightarrow y]$ equates x and y; it is a copycat process, similar to *wires* in [20]. Also, $\overline{x}(y)$ denotes the bound output $(\boldsymbol{\nu} y)\overline{x}\, y$.

In restriction $(\boldsymbol{\nu} y)P$ and input $x(y).P$ the occurrence of name y is binding, with scope P. The set of *free names* of a process P is denoted $\mathit{fn}(P)$. In a statement, a name is *fresh* if it is not among the names of the objects (processes, actions, etc.) of the statement. A process is *closed* if it does not contain free occurrences of names. We identify processes up to consistent renaming of bound names. The capture-avoiding substitution of x for y in P is denoted $P\{x/y\}$. Notation $\widetilde{k}$ denotes a finite sequence of pairwise distinct names $k_1, k_2, \cdots$. We sometimes treat sequences of names as sets.

$$\text{(id)}\ (\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \xrightarrow{\tau} P\{y/x\} \quad \text{(n.out)}\ \overline{x}\, y.P \xrightarrow{\overline{x}\, y} P \quad \text{(n.in)}\ x(y).P \xrightarrow{x(z)} P\{z/y\}$$
$$\text{(s.out)}\ x \lhd l; P \xrightarrow{\overline{x \lhd l}} P \quad \text{(s.in)}\ x \rhd \{l_i : P_i\}_{i \in I} \xrightarrow{x \lhd l_j} P_j \quad (j \in I)$$

Fig. 1. LTS for Processes (Excerpt).

Reduction expresses the internal behavior of processes. Closed under structural congruence (noted $\equiv$, see [5]), it is the binary relation on processes defined by the rules:

$$\begin{array}{cc} \overline{x}\, y.Q \mid x(z).P \to Q \mid P\{y/z\} & \overline{x}\, y.Q \mid !x(z).P \to Q \mid P\{y/z\} \mid !x(z).P \\ (\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \to P\{y/x\} & Q \to Q' \Rightarrow P \mid Q \to P \mid Q' \\ P \to Q \Rightarrow (\boldsymbol{\nu} y)P \to (\boldsymbol{\nu} y)Q & x \lhd l_j; P \mid x \rhd \{l_i : Q_i\}_{i \in I} \to P \mid Q_j \ \ (j \in I) \end{array}$$

The interaction of a process with its environment is defined by an early labeled transition system (LTS) for the π-calculus [20], extended with labels and transition rules for choice and forwarding. Transition $P \xrightarrow{\lambda} Q$ says that P may evolve to Q by performing the action represented by label λ, defined as: $\lambda ::= \tau \mid x(y) \mid x \lhd l \mid \overline{x}\, y \mid \overline{x}(y) \mid \overline{x \lhd l}$. Actions are the input $x(y)$, the offer $x \lhd l$, and their co-actions: the output $\overline{x}\, y$ and bound output $\overline{x}(y)$ actions, and the selection $\overline{x \lhd l}$, resp. The bound output $\overline{x}(y)$ denotes extrusion of y along x. Internal action is denoted τ. Fig. 1 gives a selection of the rules that define $P \xrightarrow{\lambda} Q$. Weak transitions are as usual: $\Longrightarrow$ is the reflexive, transitive closure of $\xrightarrow{\tau}$. Notation $\stackrel{\lambda}{\Longrightarrow}$ stands for $\Longrightarrow \xrightarrow{\lambda} \Longrightarrow$ (given $\lambda \neq \tau$) and $\stackrel{\tau}{\Longrightarrow}$ stands for $\Longrightarrow$.

Session Types as Linear Logic Propositions. The type theory of [5] connects session types as linear logic propositions. Main properties derived from typing, absent from other binary session type theories, are *global progress* (deadlock-freedom) and *termination* [19]. The syntax of binary types is as follows:

Definition 1 (Binary Types). *Types* (A, B, C) *are given by*
$$A, B ::= \mathbf{1} \mid\ !A \mid A \otimes B \mid A \multimap B \mid\ \&\{l_i : A_i\}_{i \in I} \mid\ \oplus\{l_i : A_i\}_{i \in I}$$

We use $A \otimes B$ (resp. $A \multimap B$) to type a name that performs an output (resp. an input) to its partner, sending (resp. receiving) a name of type A, and then behaves as type B. Thus, $A \otimes B$ and $A \multimap B$ represent the session types $!A; B$ and $?A; B$ introduced in [13]. We generalize [5] by considering n-ary offer $\&$ and choice $\oplus$. Given a finite index set I, $\&\{l_i{:}A_i\}_{i \in I}$ types a name that offers a choice between an l_i. Dually, $\oplus\{l_i{:}A_i\}_{i \in I}$ types the selection of one of the l_i. Type $!A$ types a shared channel, used by a server to spawn an arbitrary number of new sessions (possibly none), each one conforming to A. Type $\mathbf{1}$ is the terminated session; names of type $\mathbf{1}$ may be passed around as opaque values.

A *type environment* collects type assignments of the form $x{:}A$, where x is a name and A is a type, the names being pairwise disjoint. We consider two typing environments, subject to different structural properties: a *linear* part Δ and an *unrestricted* part Γ, where weakening and contraction principles hold for Γ but not for Δ.

A *type judgment* $\Gamma; \Delta \vdash P :: z{:}C$ asserts that P provides behavior C at channel z, building on "services" declared in $\Gamma; \Delta$. This way, e.g., a client Q that relies on external services and does not provide any is typed as $\Gamma; \Delta \vdash Q :: z{:}\mathbf{1}$. The domains of Γ, Δ and $z{:}C$ are required to be pairwise disjoint. We write $dom(\Gamma)$ (resp. $dom(\Delta)$) to denote

$$(\mathsf{Tid})\ \frac{}{\Gamma; x{:}A \vdash [x \leftrightarrow z] :: z{:}A} \qquad (\mathsf{T1R})\ \frac{}{\Gamma; \cdot \vdash \mathbf{0} :: x{:}\mathbf{1}} \qquad (\mathsf{Tcut})\ \frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash (\boldsymbol{\nu} x)(P \mid Q) :: z{:}C}$$

$$(\mathsf{T}{\multimap}\mathsf{L})\ \frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta', x{:}A{\multimap}B \vdash \overline{x}(y).(P \mid Q) :: z{:}C} \qquad (\mathsf{T}{\otimes}\mathsf{L})\ \frac{\Gamma; \Delta, y{:}A, x{:}B \vdash P :: z{:}C}{\Gamma; \Delta, x{:}A \otimes B \vdash x(y).P :: z{:}C}$$

$$(\mathsf{T}{\oplus}\mathsf{L})\ \frac{\Gamma; \Delta, x{:}A_1 \vdash P_1 :: z{:}C \quad \cdots \quad \Gamma; \Delta, x{:}A_k \vdash P_k :: z{:}C \quad I = \{1, \ldots, k\}}{\Gamma; \Delta, x{:} \oplus\{l_i : A_i\}_{i \in I} \vdash x \triangleright \{l_i : P_i\}_{i \in I} :: z{:}C}$$

$$(\mathsf{T}{\&}\mathsf{R})\ \frac{\Gamma; \Delta \vdash P_1 :: x{:}A_1 \quad \cdots \quad \Gamma; \Delta \vdash P_k :: x{:}A_k \quad I = \{1, \ldots, k\}}{\Gamma; \Delta \vdash x \triangleright \{l_i : P_i\}_{i \in I} :: x{:}\,\&\{l_i : A_i\}_{i \in I}}$$

$$(\mathsf{T}{\&}\mathsf{L}_1)\ \frac{\Gamma; \Delta, x{:}A \vdash P :: z{:}C}{\Gamma; \Delta, x{:}\,\&\{l_i : A\}_{\{i\}} \vdash x \triangleleft l_i; P :: z{:}C} \qquad (\mathsf{T}{\&}\mathsf{L}_2)\ \frac{\Gamma; \Delta, x{:}\,\&\{l_i{:}A_i\}_{i \in I} \vdash P :: z{:}C \quad k \notin I}{\Gamma; \Delta, x{:}\,\&\{l_j{:}A_j\}_{j \in I \cup \{k\}} \vdash P :: z{:}C}$$

Fig. 2. The Type System for Binary Sessions (Excerpt).

the domain of Γ (resp. Δ), a sequence of names. Empty environments are denoted '$\cdot$'. As π-calculus terms are considered up to structural congruence, typability is closed under $\equiv$ by definition. We sometimes abbreviate $\Gamma; \Delta \vdash P :: z{:}\mathbf{1}$ as $\Gamma; \Delta \vdash P$.

Fig. 2 presents selected typing rules; see [5] for a full account. We have right and left rules: they say how to *implement* and *use* a session of a given type, respectively. We briefly comment on some of the rules. Rule (Tid) defines identity in terms of forwarding. Rule (Tcut) define typed composition via parallel composition and restriction. Implementing a session type $\&\{l_i{:}A_i\}_{i \in I}$ amounts to offering a choice between n sessions with type A_i (Rule (T&R)); its use on name x entails selecting an alternative, using prefix $x \triangleleft l_j$ (Rules (T&L$_1$) and (T&L$_2$)). Type $\oplus\{l_i : A_i\}_{i \in I}$ has a dual interpretation.

We now recall some main results for well-typed processes. For any P, define $live(P)$ iff $P \equiv (\boldsymbol{\nu}\widetilde{n})(\pi.Q \mid R)$, for some names $\widetilde{n}$, a process R, and a *non-replicated* guarded process $\pi.Q$. Also, we write $P \Downarrow$, if there is no infinite reduction path from process P.

Theorem 1 (Properties of Well-Typed Processes).

1. *Subject Reduction [5]: If $\Gamma; \Delta \vdash P :: x{:}A$ and $P \to Q$ then $\Gamma; \Delta \vdash Q{::}x{:}A$.*
2. *Global Progress [5]: If $\cdot; \cdot \vdash P{::}z{:}\mathbf{1}$ and $live(P)$ then exists a Q such that $P \to Q$.*
3. *Termination / Strong Normalization [19]: If $\Gamma; \Delta \vdash P :: x{:}A$ then $P \Downarrow$.*

Theorem 1(2), key in our work, implies that our type discipline ensures deadlock freedom. Further properties of well-typed processes concern *proof conversions* and *typed behavioral equivalences*. The correspondence in [5] is realized by relating *proof conversions* in linear logic with appropriate process equivalences. There is a group of *commuting conversions* that induces a behavioral congruence on typed processes, noted $\simeq_c$. Process equalities justified by $\simeq_c$ include, e.g., (see [19] for details):

$$\begin{aligned}
\Gamma; \Delta, y{:}A \otimes B &\vdash (\boldsymbol{\nu} x)(P \mid y(z).Q) \simeq_c y(z).(\boldsymbol{\nu} x)(P \mid Q) :: z{:}C \\
\Gamma; \Delta, y{:}A{\multimap}B &\vdash (\boldsymbol{\nu} x)(P \mid \overline{y}(z).(Q \mid R)) \simeq_c \overline{y}(z).(Q \mid (\boldsymbol{\nu} x)(P \mid R)) :: T \\
\Gamma; \Delta, y{:}\,\&\{l_i{:}A_i\}_{i \in I} &\vdash (\boldsymbol{\nu} x)(P \mid y \triangleleft l_i; Q) \simeq_c y \triangleleft l_i; (\boldsymbol{\nu} x)(P \mid Q) :: T
\end{aligned}$$

These equalities reflect a behavioral equivalence over session-typed processes, called *context bisimilarity* (noted $\approx$) [19]. Roughly, typed processes $\Gamma; \Delta \vdash P :: x{:}A$ and

$\Gamma;\Delta \vdash Q :: x{:}A$ are context bisimilar if, once composed with their requirements (described by Γ, Δ), they perform the same actions on x (as described by A). Context bisimilarity is a congruence relation on well-typed processes. We have:

Theorem 2 ([19]). *If $\Gamma;\Delta \vdash P \simeq_c Q :: z{:}C$ then $\Gamma;\Delta \vdash P \approx Q :: z{:}C$.*

Multiparty Session Types. Our syntax of global types includes constructs from [14,12]. With respect to [12], we consider value passing in branching (cf. U below), fully supporting delegation and add parallel composition. Below, *participants* are ranged over by $\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots$; *labels* are ranged over by $l_1, l_2, \ldots$. To streamline the presentation, we consider standard global types without recursion. Our approach extends to global types with recursion, exploiting the extension of [5] with co-recursion [21]. Results for global types with recursion can be found in an online technical report [3].

Definition 2 (Global/Local Types). *Define global types (G) and local types (T) as*

$$G ::= \mathsf{end} \mid \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I} \mid G_1 \mid G_2 \qquad U ::= \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{str} \mid \ldots \mid T$$
$$T ::= \mathsf{end} \mid \mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$$

$\mathcal{G}$ denotes the above set of global types. Given a finite I and pairwise different labels, $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}$ specifies that by choosing label l_i, participant $\mathtt{p}$ may send a message of type U_i to participant $\mathtt{q}$, and then continue as G_i. We decree $\mathtt{p} \neq \mathtt{q}$, so reflexive interactions are disallowed. The global type $G_1 \mid G_2$ allows the concurrent execution of G_1 and G_2. We write end to denote the completed global type. The local type $\mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I}$ denotes an offer of a set of labeled alternatives; the local type $\mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$ denotes a behavior that chooses one of such alternatives. The terminated local type is end. Following [14], there is no local type for parallel.

Example 1. Consider a global type G_{BS}, a variant of the the two-buyer protocol in [14], in which two buyers ($\mathtt{B}_1$ and $\mathtt{B}_2$) coordinate to buy an item from a seller ($\mathtt{S}$):

$$G_{\mathrm{BS}} = \mathtt{B}_1 \twoheadrightarrow \mathtt{S}{:}\big\{ send\langle \mathsf{str}\rangle.\mathtt{S} \twoheadrightarrow \mathtt{B}_1{:}\{ rep\langle \mathsf{int}\rangle.\mathtt{S} \twoheadrightarrow \mathtt{B}_2{:}\{ rep\langle \mathsf{int}\rangle.$$
$$\mathtt{B}_1 \twoheadrightarrow \mathtt{B}_2{:}\{ shr\langle \mathsf{int}\rangle.\mathtt{B}_2 \twoheadrightarrow \mathtt{S}{:}\{ ok\langle \mathbf{1}\rangle.\mathsf{end}\,,\ quit\langle \mathbf{1}\rangle.\mathsf{end}\}\}\}\}\big\}$$

Intuitively, $\mathtt{B}_1$ requests the price of an item to $\mathtt{S}$, who replies to $\mathtt{B}_1$ and $\mathtt{B}_2$. Then $\mathtt{B}_1$ communicates to $\mathtt{B}_2$ her contribution in the transaction; finally, $\mathtt{B}_2$ either confirms the purchase to $\mathtt{S}$ or closes the transaction.

We now define *projection* for global types. Following [12], projection relies on a *merge* operator on local types, which in our case considers messages U.

Definition 3 (Merge). *We define $\sqcup$ as the commutative partial operator on base and local types such that: 1.* $\mathsf{bool} \sqcup \mathsf{bool} = \mathsf{bool}$ *(and similarly for other base types); 2.* $\mathsf{end} \sqcup \mathsf{end} = \mathsf{end}$*; 3.* $\mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \sqcup \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} = \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$*; and*

4. $\mathtt{p}?\{l_k\langle U_k\rangle.T_k\}_{k\in K} \sqcup \mathtt{p}?\{l'_j\langle U'_j\rangle.T'_j\}_{j\in J} =$
$\mathtt{p}?\big(\{l_k\langle U_k\rangle.T_k\}_{k\in K\setminus J} \cup \{l'_j\langle U'_j\rangle.T'_j\}_{j\in J\setminus K} \cup \{l_l\langle U_l \sqcup U'_l\rangle.(T_l \sqcup T'_l)\}_{l\in K\cap J}\big)$

and is undefined otherwise.

$$\text{(SW1)}\quad \frac{\{\mathtt{p}_1,\mathtt{q}_1\}\#\{\mathtt{p}_2,\mathtt{q}_2\}}{\begin{array}{c}\mathtt{p}_1 \twoheadrightarrow \mathtt{q}_1{:}\{l_i\langle U_i\rangle.\mathtt{p}_2 \twoheadrightarrow \mathtt{q}_2{:}\{l'_j\langle U'_j\rangle.G_{ij}\}_{j\in J}\}_{i\in I}\\ \simeq_{\mathtt{sw}}\\ \mathtt{p}_2 \twoheadrightarrow \mathtt{q}_2{:}\{l'_j\langle U'_j\rangle.\mathtt{p}_1 \twoheadrightarrow \mathtt{q}_1{:}\{l_i\langle U_i\rangle.G_{ij}\}_{i\in I}\}_{j\in J}\end{array}}$$

$$\text{(SW2)}\quad \frac{\{\mathtt{p},\mathtt{q}\}\#\mathsf{part}(G_1)\qquad \forall i,j\in I.G_i^1=G_j^1}{\begin{array}{c}\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.(G_i^1 \mid G_i^2)\}_{i\in I}\\ \simeq_{\mathtt{sw}}\\ G_1^1 \mid \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i^2\}_{i\in I}\end{array}}$$

Fig. 3. Swapping relation on global types (Def. 6). $A\#B$ denotes that sets A, B are disjoint.

Therefore, for $U_1 \sqcup U_2$ to be defined there are two options: (a) U_1 and U_2 are identical base, terminated or selection types; (b) U_1 and U_2 are branching types, but not necessarily identical: they may offer different options but with the condition that the behavior in labels occurring in both U_1 and U_2 must be mergeable. The set of participants of G ($\mathsf{part}(G)$) is defined as: $\mathsf{part}(\mathtt{end}) = \emptyset$, $\mathsf{part}(G_1 \mid G_2) = \mathsf{part}(G_1) \cup \mathsf{part}(G_2)$, $\mathsf{part}(\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}) = \{\mathtt{p},\mathtt{q}\} \cup \bigcup_{i\in I} \mathsf{part}(G_i)$.

Definition 4 (Projection [12]). *Let G be a global type. The* projection *of G under participant $\mathtt{r}$, denoted $G\upharpoonright \mathtt{r}$, is defined as: $\mathtt{end}\upharpoonright \mathtt{r} = \mathsf{end}$ and*

- $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}\upharpoonright \mathtt{r} = \begin{cases}\mathtt{p}!\{l_i\langle U_i\rangle.G_i\upharpoonright \mathtt{r}\}_{i\in I} & \text{if } \mathtt{r}=\mathtt{p}\\ \mathtt{p}?\{l_i\langle U_i\rangle.G_i\upharpoonright \mathtt{r}\}_{i\in I} & \text{if } \mathtt{r}=\mathtt{q}\\ \sqcup_{i\in I} G_i\upharpoonright \mathtt{r} & \text{otherwise } (\sqcup \text{ as in Def. 3})\end{cases}$
- $(G_1 \mid G_2)\upharpoonright \mathtt{r} = \begin{cases}G_i\upharpoonright \mathtt{r} & \text{if } \mathtt{r}\in\mathsf{part}(G_i) \text{ and } \mathtt{r}\notin\mathsf{part}(G_j),\ i\neq j\in\{1,2\}\\ \mathsf{end} & \text{if } \mathtt{r}\notin\mathsf{part}(G_1) \text{ and } \mathtt{r}\notin\mathsf{part}(G_2)\end{cases}$

When a side condition does not hold, the map is undefined.

Definition 5 (Well-Formed Global Types [12]). *Global type $G \in \mathcal{G}$ is* well-formed (WF) *if for all $\mathtt{r} \in \mathsf{part}(G)$, the projection $G\upharpoonright \mathtt{r}$ is defined.*

The last notion required in our characterization of multiparty session types as binary sessions is a *swapping relation* over global types [6], which enables transformations among causally independent communications. Such transformations may represent optimizations that increase parallelism while preserving the intended global behavior.

Definition 6 (Global Swapping). *The* swapping *relation $\simeq_{\mathtt{sw}}$ is the smallest congruence on $\mathcal{G}$ which satisfies the rules in Fig. 3. (The symmetric of* (SW2) *is omitted.)*

3 Relating Multiparty and Binary Session Type Theories

Our analysis of multiparty protocols as binary sessions relies on the *medium process* of a global type. Mediums offer a simple device for analyzing global types using the binary session types of [5]. Mediums uniformly capture the sequencing behavior in a global type, for they take part in all message exchanges between local participants.

After defining mediums (Def. 7), we establish their characterization results (Thms. 4 and 5). We then present a process characterization of global swapping (Def. 6) in terms

of context bisimilarity (Thm. 6). Subsequently, we state operational correspondence results (Thm. 7), exploiting the auxiliary notions of *instrumented* mediums (Def. 10) and *multiparty systems* (Def. 11). We use the following conventions.

Convention 3 (Indexed/Annotated Names) *We consider* names indexed by participants $\mathtt{p}, \mathtt{q}, \ldots$, *noted* $c_{\mathtt{p}}, c_{\mathtt{q}}, \ldots$ *and* names annotated by participants, *noted* $k^{\mathtt{p}}, k^{\mathtt{q}}, \ldots$. *Given* $\mathtt{p} \neq \mathtt{q}$, *indexed names* $c_{\mathtt{p}}$ *and* $c_{\mathtt{q}}$ *denote two different objects; in contrast, annotated names* $k^{\mathtt{p}}$ *and* $k^{\mathtt{q}}$ *denote the same name* k *with different annotations. Given a* G *with* $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$, *we will write* $\mathsf{npart}(G)$ *to denote the set that contains a unique name* $c_{\mathtt{p}_j}$ *for every participant* $\mathtt{p}_j$ *of* G. *We will occasionally use* $\mathsf{npart}(G)$ *as an unordered sequence of names.*

Definition 7 (Mediums). *The* medium *of* $G \in \mathcal{G}$, *denoted* $\mathsf{M}[\![G]\!]$, *is defined as follows:*

- $\mathsf{M}[\![\mathtt{end}]\!] = \mathbf{0}$
- $\mathsf{M}[\![G_1 \mid G_2]\!] = \mathsf{M}[\![G_1]\!] \mid \mathsf{M}[\![G_2]\!]$
- $\mathsf{M}[\![\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}]\!] = c_{\mathtt{p}} \triangleright \big\{ l_i : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_i; \overline{c_{\mathtt{q}}}(v).([u \leftrightarrow v] \mid \mathsf{M}[\![G_i]\!]) \big\}_{i\in I}$

The key case is $\mathsf{M}[\![\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}]\!]$: note how the medium uses two prefixes (on name $c_{\mathtt{p}}$) to mediate with $\mathtt{p}$, followed by two prefixes (on name $c_{\mathtt{q}}$) to mediate with $\mathtt{q}$. We illustrate mediums by means of an example:

Example 2. The medium process for global type $G_{\mathtt{BS}}$ in Example 1 is:

$$\begin{array}{l}
\mathsf{M}[\![G_{\mathtt{BS}}]\!] = c_{\mathtt{B1}} \triangleright \big\{ \mathit{send} : c_{\mathtt{B1}}(v).c_{\mathtt{S}} \triangleleft \mathit{send}; \overline{c_{\mathtt{S}}}(w).([w \leftrightarrow v] \mid \\
\qquad c_{\mathtt{S}} \triangleright \{ \mathit{rep} : c_{\mathtt{S}}(v).c_{\mathtt{B1}} \triangleleft \mathit{rep}; \overline{c_{\mathtt{B1}}}(w).([w \leftrightarrow v] \mid \\
\qquad\quad c_{\mathtt{S}} \triangleright \{ \mathit{rep} : c_{\mathtt{S}}(v).c_{\mathtt{B2}} \triangleleft \mathit{rep}; \overline{c_{\mathtt{B2}}}(w).([w \leftrightarrow v] \mid \\
\qquad\qquad c_{\mathtt{B1}} \triangleright \{ \mathit{shr} : c_{\mathtt{B1}}(v).c_{\mathtt{B2}} \triangleleft \mathit{shr}; \overline{c_{\mathtt{B2}}}(w).([w \leftrightarrow v] \mid \\
\qquad\qquad\quad c_{\mathtt{B2}} \triangleright \{ \ \mathit{ok} : c_{\mathtt{B2}}(v).c_{\mathtt{S}} \triangleleft \mathit{ok}; \overline{c_{\mathtt{S}}}(w).([w \leftrightarrow v] \mid \mathbf{0}) \, , \\
\qquad\qquad\qquad \mathit{quit} : c_{\mathtt{B2}}(v).c_{\mathtt{S}} \triangleleft \mathit{quit}; \overline{c_{\mathtt{S}}}(w).([w \leftrightarrow v] \mid \mathbf{0}) \}) \}) \}) \}) \big\}
\end{array}$$

Intuitively, we expect that (well-typed) process implementations for $\mathtt{B}_1$, $\mathtt{B}_2$, and $\mathtt{S}$ should interact with $\mathsf{M}[\![G_{\mathtt{BS}}]\!]$ through channels $c_{\mathtt{B1}}$, $c_{\mathtt{B2}}$, and $c_{\mathtt{S}}$, respectively. □

We now move on to state our characterization results. We require two auxiliary notions, given next. Below, we sometimes write $\Gamma; \Delta \vdash \mathsf{M}[\![G]\!]$ instead of $\Gamma; \Delta \vdash \mathsf{M}[\![G]\!] :: z{:}\mathbf{1}$, when $z \notin \mathit{fn}(\mathsf{M}[\![G]\!])$.

Definition 8 (Compositional Typing). *We say* $\Gamma; \Delta \vdash \mathsf{M}[\![G]\!] :: z{:}C$ *is a* compositional typing *if: (i) it is a valid typing derivation; (ii)* $\mathsf{npart}(G) \subseteq \mathit{dom}(\Delta)$; *and (iii)* $C = \mathbf{1}$.

A compositional typing says that $\mathsf{M}[\![G]\!]$ depends on behaviors associated to each participant of G; it also specifies that $\mathsf{M}[\![G]\!]$ does not offer any behaviors of its own. We relate binary session types and local types: the main difference is that the former do not mention participants. Below, B ranges over base types ($\mathsf{bool}, \mathsf{nat}, \ldots$) in Def. 2.

Definition 9 (Local Types → Binary Types). *Mapping* $\langle\!\langle \cdot \rangle\!\rangle$ *from local types* T *(Def. 2) into binary types* A *(Def. 1) is inductively defined as* $\langle\!\langle \mathsf{end} \rangle\!\rangle = \langle\!\langle B \rangle\!\rangle = \mathbf{1}$ *and*

$$\begin{array}{l}
\langle\!\langle \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \rangle\!\rangle = \oplus\{l_i : \langle\!\langle U_i \rangle\!\rangle \otimes \langle\!\langle T_i \rangle\!\rangle\}_{i\in I} \\
\langle\!\langle \mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \rangle\!\rangle = \&\{l_i : \langle\!\langle U_i \rangle\!\rangle \multimap \langle\!\langle T_i \rangle\!\rangle\}_{i\in I}
\end{array}$$

3.1 Characterization Results

Our characterization results relate process $\mathsf{M}[\![G]\!]$ (well-typed with a compositional typing) and $\langle\!\langle G \upharpoonright \mathtt{p}_1 \rangle\!\rangle, \ldots, \langle\!\langle G \upharpoonright \mathtt{p}_n \rangle\!\rangle$ (i.e., the local types of G transformed into binary session types via Def. 9). Our characterization results are in two directions, given by Thms. 4 and 5. The first direction says that well-formedness of global types (Def. 5) ensures compositional typings for mediums with (logic based) binary session types:

Theorem 4 (Global Types $\rightarrow$ Typed Mediums). *Let $G \in \mathcal{G}$. If G is WF with $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ then $\Gamma; c_{\mathtt{p}_1}{:}\langle\!\langle G \upharpoonright \mathtt{p}_1 \rangle\!\rangle, \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G \upharpoonright \mathtt{p}_n \rangle\!\rangle \vdash \mathsf{M}[\![G]\!]$ is a compositional typing, for some Γ.*

The second direction of the characterization is the converse of Thm. 4: compositional typings for mediums induce global types which are WF. Given local types T_1, T_2, below we write $T_1 \preceq^{\sqcup} T_2$ if there exists a local type T' such that $T_1 \sqcup T' = T_2$ (cf. Def. 3). This notation allows us to handle the labeled alternatives silently introduced by rule (T&L$_2$).

Theorem 5 (Well-Typed Mediums $\rightarrow$ Global Types). *Let $G \in \mathcal{G}$.*
If $\Gamma; c_{\mathtt{p}_1}{:}A_1, \ldots, c_{\mathtt{p}_n}{:}A_n \vdash \mathsf{M}[\![G]\!]$ is a compositional typing then $\exists T_1, \ldots, T_n$ such that $G \upharpoonright \mathtt{p}_j \preceq^{\sqcup} T_j$ and $\langle\!\langle T_j \rangle\!\rangle = A_j$, for all $\mathtt{p}_j \in \mathsf{part}(G)$.

Thms. 4 and 5 tightly connect (i) global types, local types and projection, and (ii) medium processes, and logic-based binary session types. They also provide an independent deep justification, through purely logical arguments, to the notion of projection.

3.2 A Behavioral Characterization of Global Swapping

Global swapping (Def. 6, Fig. 3) can be directly justified from more primitive notions, based on the characterizations given by Thms. 4 and 5. By abstracting a global type's behavior in terms of its medium we may reduce transformations on global types to type-preserving transformations on processes. This is the content of Thm. 6 below, which connects global swapping ($\simeq_{\mathtt{sw}}$) and context bisimilarity ($\approx$). Hence, sequentiality of mediums can be relaxed in the case of causally independent exchanges captured by $\simeq_{\mathtt{sw}}$.

Theorem 6. *Let $G_1 \in \mathcal{G}$ such that $\mathsf{M}[\![G_1]\!]$ has a compositional typing $\Gamma; \Delta \vdash \mathsf{M}[\![G_1]\!]$, for some Γ, Δ. If $G_1 \simeq_{\mathtt{sw}} G_2$ then $\Gamma; \Delta \vdash \mathsf{M}[\![G_1]\!] \approx \mathsf{M}[\![G_2]\!]$.*

Since $\mathsf{M}[\![G]\!]$ is a low-level representation of G, the converse of Thm. 6 is less interesting, for type-preserving transformations at the (low) level of mediums do not always correspond to behavior-preserving transformations at the (high) level of global types. That is, since $\mathsf{M}[\![G]\!]$ implements each communication in G using several prefixes, swapping in G occurs only when *all* relevant prefixes in $\mathsf{M}[\![G]\!]$ can be commuted via $\simeq_c$.

3.3 Operational Correspondence Results

The results given so far focus on the static semantics of multiparty and binary systems, and are already key to justify essential properties such as absence of global deadlock.

$$\mathtt{p} \twoheadrightarrow \mathtt{q}\colon\{l_i\langle U_i\rangle.\mathbf{G}_i\}_{i\in I} \xrightarrow{\overline{\mathtt{p}\lhd l_j}} \mathtt{p}\rightsquigarrow\mathtt{q}\colon l_j\langle U_j\rangle.\mathbf{G}_j \ \ (j\in I) \qquad \mathtt{p}\rightsquigarrow\mathtt{q}\colon l\langle U\rangle.\mathbf{G} \xrightarrow{\overline{\mathtt{p}}} \mathtt{p}\rightsquigarrow\mathtt{q}\colon l((U)).\mathbf{G}$$

$$\mathtt{p}\rightsquigarrow\mathtt{q}\colon l((U)).\mathbf{G} \xrightarrow{\mathtt{q}\lhd l} \mathtt{p}\rightsquigarrow\mathtt{q}\colon ((U)).\mathbf{G} \qquad \mathtt{p}\rightsquigarrow\mathtt{q}\colon ((U)).\mathbf{G} \xrightarrow{\mathtt{q}} \mathbf{G}$$

Fig. 4. LTS over Finite, Extended Global Types (Excerpt).

We now move on to dynamic semantics, and establish operational correspondence results between a global type and its medium process (Thm. 7).

We define the *instrumented medium* of a global type G, denoted $\mathbf{M}_{\widetilde{k}}[\![G]\!]$, as a natural extension of Def. 7. Process $\mathbf{M}_{\widetilde{k}}[\![G]\!]$ exploits fresh sessions (denoted $\widetilde{k}$), to emit a visible signal for each action of G. We use $\widetilde{k}$ as annotated names (cf. Conv. 3): each action on a k_i contains the identity of the participant of G which performs it. Then, using $\mathbf{M}_{\widetilde{k}}[\![G]\!]$ we define the set of *systems* associated to G (Def. 11), which collects process implementations for G mediated by $\mathbf{M}_{\widetilde{k}}[\![G]\!]$. Since interactions between local implementations and $\mathbf{M}_{\widetilde{k}}[\![G]\!]$ are unobservable actions, Thm. 7 connects (i) the visible behavior of a system along annotated names $\widetilde{k}$, and (ii) the visible behavior of G, defined by an LTS on global types (a variant of that in [12]). Below, $k^{\mathtt{p}}.P$ stands for $k^{\mathtt{p}}(x).P$ when x is not relevant in P. Also, $\widehat{k^{\mathtt{p}}}.P$ stands for $\overline{k^{\mathtt{p}}}(v).(\mathbf{0} \mid P)$ for some v.

Definition 10 (Instrumented Mediums). *Let $\widetilde{k}$ be fresh, annotated names. The* instrumented medium *of $G \in \mathcal{G}$ with respect to $\widetilde{k}$, denoted $\mathbf{M}_{\widetilde{k}}[\![G]\!]$, is defined as follows:*

- $\mathbf{M}_k[\![\mathtt{end}]\!] = \mathbf{0}$
- $\mathbf{M}_{k_1,k_2}[\![G_1 \mid G_2]\!] = \mathbf{M}_{k_1}[\![G_1]\!] \mid \mathbf{M}_{k_2}[\![G_2]\!]$
- $\mathbf{M}_k[\![\mathtt{p}\twoheadrightarrow\mathtt{q}\colon\{l_i\langle U_i\rangle.G_i\}_{i\in I}]\!] =$
  $c_{\mathtt{p}} \triangleright\big\{ l_i : \ k^{\mathtt{p}} \lhd l_i; c_{\mathtt{p}}(u).\widehat{k^{\mathtt{p}}}.\big(c_{\mathtt{q}} \lhd l_i; k^{\mathtt{q}} \triangleright\{ l_i : \overline{c_{\mathtt{q}}}(v).([u\leftrightarrow v] \mid k^{\mathtt{q}}.\mathbf{M}_k[\![G_i]\!])\}_{\{i\}} \ \big)\big\}_{i\in I}$

The key case is $\mathbf{M}_k[\![\mathtt{p}\twoheadrightarrow\mathtt{q}\colon\{l_i\langle U_i\rangle.G_i\}_{i\in I}]\!]$. Each action of the multiparty exchange is "echoed" by an action on annotated name k: the selection of label l_i by $\mathtt{p}$ is followed by prefix $k^{\mathtt{p}} \lhd l_i$; the output from $\mathtt{p}$ (captured by the medium by the input $c_{\mathtt{p}}(u)$) is echoed by prefix $k^{\mathtt{p}}$. This way, the instrumented process $\mathbf{M}_{\widetilde{k}}[\![G]\!]$ induces a fine-grained correspondence with G, exploiting process actions with explicit participant identities.

To state our operational correspondence result, we introduce *extended* global types and a labeled transition system (LTS) for (extended) global types. The syntax of extended global types is defined as $G ::= \mathbf{G} \mid \mathbf{G}_1 \mid \mathbf{G}_2$ with

$$\mathbf{G} ::= \mathtt{end} \mid \mathtt{p}\twoheadrightarrow\mathtt{q}\colon\{l_i\langle U_i\rangle.\mathbf{G}_i\}_{i\in I} \mid \mathtt{p}\rightsquigarrow\mathtt{q}\colon l\langle U\rangle.\mathbf{G} \mid \mathtt{p}\rightsquigarrow\mathtt{q}\colon l((U)).\mathbf{G} \mid \mathtt{p}\rightsquigarrow\mathtt{q}\colon ((U)).\mathbf{G}$$

We consider parallel composition of sequential global types. We also have three auxiliary forms for global types, denoted with $\rightsquigarrow$: they represent intermediate steps. Types $\mathtt{p}\rightsquigarrow\mathtt{q}\colon l\langle U\rangle.G$ and $\mathtt{p}\rightsquigarrow\mathtt{q}\colon l((U)).G$ denote the commitment of $\mathtt{p}$ to *output* and input along label l, resp. Type $\mathtt{p}\rightsquigarrow\mathtt{q}\colon ((U)).G$ represents the state just before the actual input action by $\mathtt{q}$. We need the expected extension of Def. 10 for these types.

We adapt the LTS in [12] to the case of (extended) global types. The set of observables is $\sigma ::= \mathtt{p} \mid \mathtt{p}\lhd l \mid \overline{\mathtt{p}} \mid \overline{\mathtt{p}\lhd l}$. Below, $psubj(\sigma)$ denotes the participant in σ. This

way, e.g., $psubj(\mathtt{p} \triangleleft l) = \mathtt{p}$. The LTS over global types, noted $G \xrightarrow{\sigma} G'$, is defined by rules including those in Fig. 4. Since Def. 10 annotates prefixes on k with participant identities, their associated actions will be annotated too; given a participant $\mathtt{p}$, we may define the set of annotated visible actions as: $\lambda^{\mathtt{p}} ::= k^{\mathtt{p}}(y) \mid k^{\mathtt{p}} \triangleleft l \mid \overline{k^{\mathtt{p}}}\, y \mid \overline{k^{\mathtt{p}}}(y) \mid \overline{k^{\mathtt{p}} \triangleleft l}$. We write $k^{\mathtt{p}}$ and $\widehat{k^{\mathtt{p}}}$ to denote actions $k^{\mathtt{p}}(y)$ and $\overline{k^{\mathtt{p}}}(y)$, resp., whenever object y is unimportant. Also, $psubj(\lambda^{\mathtt{p}})$ denotes the participant $\mathtt{p}$ which annotates λ. This way, e.g., $psubj(k^{\mathtt{p}}) = \mathtt{p}$ and $psubj(\overline{k^{\mathtt{q}} \triangleleft l}) = \mathtt{q}$. To relate labels for global types and process labels, given an annotated name k, we define mappings $\{\!\{\cdot\}\!\}^k$ and $\|\cdot\|$ as follows:

$$\{\!\{\mathtt{p}\}\!\}^k = k^{\mathtt{p}} \qquad \{\!\{\mathtt{p} \triangleleft l\}\!\}^k = k^{\mathtt{p}} \triangleleft l \qquad \{\!\{\overline{\mathtt{p}}\}\!\}^k = \widehat{k^{\mathtt{p}}} \qquad \{\!\{\overline{\mathtt{p} \triangleleft l}\}\!\}^k = \overline{k^{\mathtt{p}} \triangleleft l}$$
$$\|k^{\mathtt{p}}\| = \mathtt{p} \qquad \|k^{\mathtt{p}} \triangleleft l\| = \mathtt{p} \triangleleft l \qquad \|\widehat{k^{\mathtt{p}}}\| = \overline{\mathtt{p}} \qquad \|\overline{k^{\mathtt{p}} \triangleleft l}\| = \overline{\mathtt{p} \triangleleft l}$$

Operational correspondence is stated in terms of the *multiparty systems* of a global type. Following Def. 8, we say that $\Gamma; \Delta, \Delta' \vdash \mathbf{M}_{\widetilde{k}}[\![G]\!] :: z{:}C$ is an *instrumented* compositional typing if (i) it is a valid typing derivation; (ii) $\mathsf{npart}(G) \subseteq dom(\Delta)$; (iii) $C = \mathbf{1}$; (iv) $dom(\Delta') = \widetilde{k}$:

Definition 11 (Systems). *Let $G \in \mathcal{G}$ be a WF global type, with $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$. Also, let $\Gamma; \Delta, \Delta' \vdash \mathbf{M}_{\widetilde{k}}[\![G]\!]$ be an instrumented compositional typing, with $\Delta = c_{\mathtt{p}_1}{:}\langle\!\langle G \upharpoonright \mathtt{p}_1 \rangle\!\rangle, \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G \upharpoonright \mathtt{p}_n \rangle\!\rangle$, for some Γ. Let $\widetilde{z} = \mathsf{npart}(G)$. The* set of systems *of G is defined as:*

$$\mathcal{S}_{\widetilde{k}}(G) = \left\{ (\boldsymbol{\nu}\widetilde{z})(Q_1 \mid \cdots \mid Q_n \mid \mathbf{M}_{\widetilde{k}}[\![G]\!]) \;\middle|\; \cdot; \cdot \vdash Q_j :: c_{\mathtt{p}_j}{:}\langle\!\langle G \upharpoonright \mathtt{p}_j \rangle\!\rangle,\ j \in 1 \ldots n \right\}$$

Thus, given G, a multiparty system is obtained by composing $\mathbf{M}_{\widetilde{k}}[\![G]\!]$ with well-typed implementations for each of the local projections of G. An $R \in \mathcal{S}_{\widetilde{k}}(G)$ is an implementation of the multiparty protocol G. By construction, its only visible actions are on annotated names: interactions between all the Q_j and $\mathbf{M}_{\widetilde{k}}[\![G]\!]$ will be unobservable.

Thm. 7 below connects global types and systems: it confirms that (instrumented) medium processes faithfully mirror the communicating behavior of extended global types. Below, we write $G \xrightarrow{\sigma[\mathtt{p}]} G'$ if $G \xrightarrow{\sigma} G'$ and $psubj(\sigma) = \mathtt{p}$. Also, we write $P \xrightarrow{\lambda[\mathtt{p}]} P'$ (resp. $P \xRightarrow{\lambda[\mathtt{p}]} P'$) if $P \xrightarrow{\lambda} P'$ (resp. $P \xRightarrow{\lambda} P'$) holds and $psubj(\lambda) = \mathtt{p}$.

Theorem 7 (Operational Correspondence). *Let G be an extended WF global type and $R \in \mathcal{S}_{\widetilde{k}}(G)$. We have: If $G \xrightarrow{\sigma[\mathtt{p}]} G'$ then $R \xRightarrow{\lambda[\mathtt{p}]} R'$, for some $\lambda, R', k \in \widetilde{k}$ such that $\lambda = \{\!\{\sigma\}\!\}^k$ and $R' \in \mathcal{S}_{\widetilde{k}}(G')$. Moreover, if there is some R_0 s.t. $R \Longrightarrow R_0 \xrightarrow{\lambda[\mathtt{p}]} R'$ then $G \xrightarrow{\sigma[\mathtt{p}]} G'$, for some σ, G' such that $\sigma = \|\lambda\|$ and $R' \in \mathcal{S}_{\widetilde{k}}(G')$.*

4 Example: Sharing in Multiparty Conversations

Here we further illustrate reasoning about global types in $\mathcal{G}$ by exploiting the properties given in § 3. In particular, we illustrate non-trivial forms of replication and sharing.

Let us consider the global type G_{BS}, given in Example 1. The medium processes of G_{BS}, denoted $\mathsf{M}[\![G_{\mathrm{BS}}]\!]$, has been detailed in Example 2; we proceed to examine its properties. Relying on Thms. 4 and 5, we have the compositional typing:

$$\Gamma;\ c_1{:}\mathsf{B1}, c_2{:}\mathsf{S}, c_3{:}\mathsf{B2} \vdash \mathsf{M}[\![G_{\mathrm{BS}}]\!] :: -{:}\mathbf{1} \tag{1}$$

for some Γ and with $\mathsf{B1} = \langle\!\langle G_{\mathrm{BS}} \upharpoonright \mathtt{B}_1 \rangle\!\rangle$, $\mathsf{S} = \langle\!\langle G_{\mathrm{BS}} \upharpoonright \mathtt{S} \rangle\!\rangle$, and $\mathsf{B2} = \langle\!\langle G_{\mathrm{BS}} \upharpoonright \mathtt{B}_2 \rangle\!\rangle$. To implement the protocol, one may simply compose $\mathsf{M}[\![G_{\mathrm{BS}}]\!]$ with type compatible processes $\cdot;\cdot \vdash \mathit{Buy1} :: c_1{:}\mathsf{B1}$, $\cdot;\cdot \vdash \mathit{Sel} :: c_2{:}\mathsf{S}$, and $\cdot;\cdot \vdash \mathit{Buy2} :: c_3{:}\mathsf{B2}$:

$$\Gamma;\cdot \vdash (\boldsymbol{\nu} c_1)(\mathit{Buy1} \,|\, (\boldsymbol{\nu} c_2)(\mathit{Sel} \mid (\boldsymbol{\nu} c_3)(\mathit{Buy2} \,|\, \mathsf{M}[\![G_{\mathrm{BS}}]\!]))) \tag{2}$$

The binary session types in § 2 allows us to infer that the multiparty system defined by (2) adheres to the declared projected types, is deadlock-free, and terminating.

Just as we inherit strong properties for $\mathit{Buy1}$, Sel, and $\mathit{Buy2}$ above, we may inherit the same properties for more interesting system configurations. In particular, local implementations which appeal to replication and sharing, admit also precise analyses thanks to the characterizations in § 3. Let us consider a setting in which the processes to be composed with the medium must be invoked from a replicated service (a source of generic process definitions). We may have $\cdot;\cdot \vdash !u_1(w).\mathit{Buy1}_{\,w} :: u_1{:}\,!\mathsf{B1}$ and

$$\cdot;\cdot \vdash !u_2(w).\mathit{Sel}_w :: u_2{:}\,!\mathsf{S} \qquad \cdot;\cdot \vdash !u_3(w).\mathit{Buy2}_{\,w} :: u_3{:}\,!\mathsf{B2}$$

and the following "initiator processes" would spawn a copy of the medium's requirements, instantiated at appropriate names:

$$\cdot; u_1{:}\,!\mathsf{B1} \vdash \overline{u_1}(x).[x \leftrightarrow c_1] :: c_1{:}\mathsf{B1} \qquad \cdot; u_2{:}\,!\mathsf{S} \vdash \overline{u_2}(x).[x \leftrightarrow c_2] :: c_2{:}\mathsf{S}$$
$$\cdot; u_3{:}\,!\mathsf{B2} \vdash \overline{u_3}(x).[x \leftrightarrow c_3] :: c_3{:}\mathsf{B2}$$

Let us write $\mathit{RBuy1}$, $\mathit{RBuy2}$, and RSel to denote the composition of replicated definitions and initiators above. Intuitively, they represent the "remote" variants of $\mathit{Buy1}$, $\mathit{Buy2}$, and RSel, respectively. We may then define the multiparty system:

$$\Gamma;\cdot \vdash (\boldsymbol{\nu} c_1)(\mathit{RBuy1} \,|\, (\boldsymbol{\nu} c_2)(\mathit{RSel} \,|\, (\boldsymbol{\nu} c_3)(\mathit{RBuy2} \,|\, \mathsf{M}[\![G_{\mathrm{BS}}]\!])))$$

which, with a concise specification, improves (2) with concurrent invocation/instantiation of replicated service definitions. As (2), the revised composition above is correct, deadlock-free, and terminating.

Rather than appealing to initiators, a scheme in which the medium invokes and instantiates services directly is also expressible in our framework, in a type consistent way. Using (1), and assuming $\Gamma = u_1{:}\mathsf{B1}, u_2{:}\mathsf{S}, u_3{:}\mathsf{B2}$, we may derive:

$$\Gamma;\cdot \vdash \overline{u_1}(c_1).\overline{u_2}(c_2).\overline{u_3}(c_3).\mathsf{M}[\![G_{\mathrm{BS}}]\!] \tag{3}$$

Hence, prior to engaging in the mediation behavior for G_{BS}, the medium first spawns a copy of the required services. We may relate the guarded process in (3) with the multicast session request construct in multiparty session processes [14]. Observe that (3) cleanly distinguishes between session initiation and actual communication behavior:

the distinction is given at the level of processes (cf. output prefixes on u_1, u_2, and u_3) but also at the level of typed interfaces.

The service invocation (3) may be regarded as "eager": all required services must be sequentially invoked prior to executing the protocol. We may also obtain, in a type-consistent manner, a medium process implementing a "lazy" invocation strategy that spawns services only when necessary. For the sake of example, consider process

$$\mathit{Eager}_{\mathrm{BS}} \triangleq \overline{u_3}(c_3).\mathsf{M}[\![G_{\mathrm{BS}}]\!]$$

in which only the invocation on u_3 is blocking the protocol, with "open" dependencies on c_1, c_2. That is, we have $\Gamma; c_1{:}\mathsf{B1}, c_2{:}\mathsf{S} \vdash \mathit{Eager}_{\mathrm{BS}} :: z{:}\mathbf{1}$. It could be desirable to postpone the invocation on u_3 as much as possible. By combining the commutations on process prefixes realized by $\simeq_c$ [19] and Thm. 2, we may obtain:

$$\Gamma; c_1{:}\mathsf{B1}, c_2{:}\mathsf{S} \vdash \mathit{Eager}_{\mathrm{BS}} \approx \mathit{Lazy}_{\mathrm{BS}} :: -{:}\mathbf{1}$$

where $\mathit{Lazy}_{\mathrm{BS}}$ is obtained from $\mathit{Eager}_{\mathrm{BS}}$ by "pushing inside" prefix $\overline{u_3}(c_3)$.

5 Multiparty Session Types with Behavioral Genericity

To illustrate the modularity of our approach, we conservatively extend, for the first time, multiparty session types with *parametric polymorphism*, developed for binary sessions in [4,22]. Although expressed by second-order quantifiers on (session) types—in the style of the polymorphic λ-calculus—parametric polymorphism in our setting means *behavioural genericity* in multiparty conversations (i.e., passing first-class behavioral interfaces in messages), not just datatype genericity. In this section we describe how to extend the approach and results in § 3 to polymorphic, multiparty session types.

In [4] we have extended the framework of [5] with impredicative universal and existential quantification over session types, denoted with $\forall X.A$ and $\exists X.A$, respectively. These two types are interpreted as the input and output of a session type, respectively. More precisely, $\forall X.A$ is the type of a process that inputs some session type S (which we may as a kind of interface passing) and then behaves as prescribed by $A\{S/X\}$. $\exists X.A$ is the type of a process that sends some session type S and then behaves as prescribed by $A\{S/X\}$. From the point of view of the receiver of such S, the protocol S is in a sense opaque; therefore, after inputting S the receiver behaves parametrically (in the sense of behavioral polymorphism) for any such S. In any case, any usage of S by the sender will necessarily be matched by some appropriate parties in the system. A relevant example of the phenomenon can be recovered from [4]. Consider the type

$$\mathsf{CloudServer} : \forall X.(\mathsf{api} \multimap X) \multimap X$$

A session with this type will first input some session type X, say GMail, and then will input a session with type api$\multimap$GMail (that may be realized by a piece of code that will first receive a channel implementing the api behavior and will after—building on it—behave as specified by GMail) and then offers the behavior GMail. A system implementing the CloudServer type must somehow provide the api service internally

$$\text{(T}\forall\text{L)}\quad \frac{\Omega \vdash B\ \mathsf{type} \quad \Omega;\Gamma;\Delta, x : A\{^{B}/_{X}\} \vdash P :: T}{\Omega;\Gamma;\Delta, x : \forall X.A \vdash \overline{x}\, B.P :: T} \qquad \text{(T}\forall\text{R)}\quad \frac{\Omega, X;\Gamma;\Delta \vdash P :: z{:}A}{\Omega;\Gamma;\Delta \vdash z(X).P :: z{:}\forall X.A}$$

$$\text{(T}\exists\text{L)}\quad \frac{\Omega, X;\Gamma;\Delta, x{:}A \vdash P :: T}{\Omega;\Gamma;\Delta, x : \exists X.A \vdash x(X).P :: T} \qquad \text{(T}\exists\text{R)}\quad \frac{\Omega \vdash B\ \mathsf{type} \quad \Omega;\Gamma;\Delta \vdash P :: x{:}A\{^{B}/_{X}\}}{\Omega;\Gamma;\Delta \vdash \overline{x}\, B.P :: x{:}\exists X.A}$$

Fig. 5. Typing Rules for Polymorphic, Binary Session Types.

and pass it to the channel of type $\mathsf{api}\multimap\mathsf{GMail}$ (e.g., representing a piece of mobile code). Notice that after that the GMail service may be produced by copy-cating the resulting behavior to the cloud server client. The crucial point here is that the cloud server behaves uniformly for whatever session type X is requested for it to execute; its role in this case is to provide the suitable api. Of course, at runtime, all interactions at X will take place as prescribed by the concrete session type involved (in this example, GMail), which may be an arbitrarily complex (behaviorally generic) session type.

5.1 Binary Session Types with Parametric Polymorphism

We now recall key definitions from [4]. The process model in § 2 is extended with processes $\overline{x}\, A.P$ (output type A, proceed as P) and $x(X).P$ (receive a type A, proceed as $P\{^{A}/_{X}\}$) and the reduction rule: $\overline{x}\, A.Q \mid x(X).P \rightarrow Q \mid P\{^{A}/_{X}\}$, where $\{^{A}/_{X}\}$ is the substitution of type variable X with session type A. Thus, our process syntax allows terms such as, e.g., $\overline{x}\, A.Q \mid x(X).\overline{y}\, X.P$, where A is a session typed protocol.

We extend binary types (cf. Def. 1) with existential and universal quantification:

$$A, B ::= \mathbf{1} \mid\, !A \mid\, A \otimes B \mid A\multimap B \mid\, \&\{l_i{:}A_i\}_{i\in I} \mid\, \oplus\{l_i{:}A_i\}_{i\in I} \mid\, X \mid\, \exists X.A \mid\, \forall X.A$$

Besides Δ and Γ, the polymorphic type system uses environment Ω to record type variables. We have two judgments. Judgment $\Omega \vdash A$ type denotes that A is a well-formed type with free variables registered in Ω (see [4] for well-formedness rules). Also, judgement $\Omega;\Gamma;\Delta \vdash P :: x{:}A$ states that P implements a session of type A along channel x, provided it is composed with processes providing sessions linearly in Δ and persistently in Γ, such that the types occurring in Ω are well-formed.

The required typing rules result by adding Ω in Fig. 2 and by considering rules in Fig. 5, which explain how to *provide* and *use* sessions of a polymorphic type. Rule (T$\forall$R) types the offering of a session of universal type $\forall X.A$ by inputing an arbitrary type, bound to X, and proceeding as A, which may bind the type variable X, regardless of what the received type is. Rule (T$\forall$L) says that the use of type $\forall X.A$ consists of the output of a type B (well-formed under Ω) which then warrants the use of the session as $A\{^{B}/_{X}\}$. The existential type is dual: providing an existentially typed session $\exists X.A$ is accomplished by outputting a type B and then providing a session of type $A\{^{B}/_{X}\}$ (Rule (T$\exists$R)) . Using an existential session $\exists X.A$ implies inputing a type and then using the session as A, regardless of the received session type (Rule (T$\exists$L)).

Well-typed polymorphic processes satisfy Thm. 1 and *relational parametricity* [4], a reasoning principle stated next. We require some notation, fully detailed in [4]: $\omega{:}\Omega$

denotes a type substitution that assigns a closed type to variables in Ω. Notation $\widehat{\omega}(P)$ denotes the application of ω to type variables in P. Also, $\eta{:}\omega_1 \Leftrightarrow \omega_2$ is an equivalence candidate assignment (a typed relation on processes) between ω_1 and ω_2. Moreover, $\approx_{\mathrm{L}}$ denotes a *logical equivalence* relation that coincides with barbed congruence.

Theorem 8 (Relational Parametricity [4]). *If $\Omega; \Gamma; \Delta \vdash P :: z{:}A$ then, for all $\omega_1{:}\Omega$, $\omega_2{:}\Omega$, and $\eta{:}\omega_1 \Leftrightarrow \omega_2$: $\Gamma; \Delta \vdash \widehat{\omega_1}(P) \approx_{\mathrm{L}} \widehat{\omega_2}(P) :: z{:}A[\eta{:}\omega_1 \Leftrightarrow \omega_2]$.*

Thm. 8 entails *behavioral genericity*, a form of representation independence: it says that process P behaves the same independently from instantiations of its free type variables.

5.2 Multiparty Session Types with Polymorphism

We extend global types in $\mathcal{G}$ (Def. 2) with variables $X, X', \ldots$ and with a construct $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l[X].G'\}$, which introduces parametric polymorphism (X is meant to occur in G'). To our knowledge, this is the first theory of its kind:

Definition 12 (Polymorphic Session Types). *Define global types and local types as*

$$
\begin{array}{l}
G ::= \mathsf{end} \mid G_1 \mid G_2 \mid \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I} \mid \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l[X].G\} \mid X \\
U ::= \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{str} \mid \ldots \mid T \mid (T)^{\dagger} \\
T ::= \mathsf{end} \mid \mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \mathtt{p}!\{l[X].T\} \mid \mathtt{p}?\{l[X].T\} \mid X
\end{array}
$$

Above $(\cdot)^{\dagger}$ denotes a function on local types that discards participant identities. E.g., $(\mathtt{p}?\{l[X].T\})^{\dagger} = ?\{l[X].(T)^{\dagger}\}$ and $(\mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I})^{\dagger} = ?\{l_i\langle U_i\rangle.(T_i)^{\dagger}\}_{i\in I}$.

We write $\mathcal{G}^{\forall\exists}$ to denote the above global types. The global type $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l[X].G\}$ signals that $\mathtt{p}$ sends to $\mathtt{q}$ an arbitrary local type (protocol), thus specifying $\mathtt{q}$ as a generic partner. Also, G is a *generic* global specification: its behavior will be depend on the type sent by $\mathtt{p}$ to $\mathtt{q}$, which should be explicit in $\mathtt{p}$'s implementation. This new global type is related to local types $\mathtt{p}!\{l[X].T\}$ and $\mathtt{p}?\{l[X].T\}$, which are to be understood as existential and universal quantification on local types, respectively—see below. The global type X should be intuitively understood as a behavior that remains "globally abstract", in the sense that it is determined by a concrete local type exchanged between two participants, namely, as a result of a (previous) communication of the form $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l[X].G\}$. As a result, the (global) communication behavior associated to local type exchanged between $\mathtt{p}$ and $\mathtt{q}$ should remain abstract (opaque) to other participants of the protocol.

The *projection* of $G \in \mathcal{G}^{\forall\exists}$ onto participant $\mathtt{r}$, denoted $G \upharpoonright \mathtt{r}$, extends Def. 4 by adding $X \upharpoonright \mathtt{r} = X$ and by letting:

$$
(\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l[X].G\}) \upharpoonright \mathtt{r} = \begin{cases} \mathtt{p}!\{l[X].(G\upharpoonright \mathtt{r})\} & \text{if } \mathtt{r} = \mathtt{p} \\ \mathtt{p}?\{l[X].(G\upharpoonright \mathtt{r})\} & \text{if } \mathtt{r} = \mathtt{q} \\ G\upharpoonright \mathtt{r} & \text{otherwise} \end{cases}
$$

Well-formedness of global types in $\mathcal{G}^{\forall\exists}$ is based on projectability but also on consistent uses of type variables: a participant can only communicate the types it knows. (This condition is similar to *history-sensitivity*, as in [1].) This way, e.g., an ill-formed type is $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_1[X].\mathtt{r} \twoheadrightarrow \mathtt{s}{:}\{l_2\langle ?\{l\langle \mathsf{int}\rangle.X\}\rangle.\mathtt{end}\}\}$, since $\mathtt{r}$, $\mathtt{s}$ do not know the type sent by $\mathtt{p}$.

5.3 Mediums for Multiparty Session Types With Polymorphism

Mediums for global types in $\mathcal{G}^{\forall\exists}$ are defined by extending Def. 7 as follows:

$$\mathsf{M}[\![\mathtt{p}\twoheadrightarrow\mathtt{q}{:}\{l[X].G\}]\!] = c_{\mathtt{p}} \triangleright \big\{ l : c_{\mathtt{p}}(X).c_{\mathtt{q}} \triangleleft l; \overline{c_{\mathtt{q}}}\, X.\mathsf{M}[\![G]\!]\big\} \qquad\qquad \mathsf{M}[\![X]\!] = \mathbf{0}$$

Observe that type variable X should not generate a mediator behavior, as we want to remain generic. The relation between local types and binary types extends Def. 9 with:

$$\langle\!\langle \mathtt{p}!\{l[X].T\}\rangle\!\rangle = \oplus\{l : \exists X.\langle\!\langle T\rangle\!\rangle\} \qquad \langle\!\langle \mathtt{p}?\{l[X].T\}\rangle\!\rangle = \&\{l : \forall X.\langle\!\langle T\rangle\!\rangle\}$$

and by letting $\langle\!\langle X\rangle\!\rangle = X$ and $\langle\!\langle (T)^{\dagger}\rangle\!\rangle = \langle\!\langle T\rangle\!\rangle$. The characterization results in § 3.1 hold also for global types in $\mathcal{G}^{\forall\exists}$.

6 Mediums At Work: A Behaviorally Generic Multiparty Protocol

We illustrate our approach and results via a simple example. Consider the global type G_{p}, inspired by the CloudServer from [4] already hinted to above. It features behavioral genericity (as enabled by parametric polymorphism); below, str, bool, denote basic data types, and api is a session type describing the cloud infrastructure API.

$$G_{\mathrm{p}} = \mathtt{p}\twoheadrightarrow\mathtt{q}{:}\big\{ l_1\langle\mathsf{bool}\rangle.\mathtt{q}\twoheadrightarrow\mathtt{r}{:}\{l[X].\mathtt{q}\twoheadrightarrow\mathtt{r}{:}\{l_2\langle ?\{l_3\langle\mathsf{api}\rangle.X\}\rangle.X\}\}\big\}$$

We have participants p, q, and r. The intent is that r is a behaviorally generic participant, that provides a behavior of type api required by q. Crucially, r may interact with q independently of the local type sent by q. Such a local type is explicit in q's implementation (see below), rather than in the global type G_{p}.

In G_{p}, participant p first sends a boolean value to q; then, q sends an unspecified protocol to r, say M, which is to be used subsequently in an exchange from q to r. Notice that M occurs in the value that r receives from q and influences the behavior after that exchange. Indeed, the value $?\{l_3\langle\mathsf{api}\rangle.X\}$ denotes an unspecified session type that relies on the reception of a session of type api. The local projections for G_{p} are $G_{\mathrm{p}}\upharpoonright\mathtt{p} = \mathtt{p}!\{l_1\langle\mathsf{bool}\rangle.\mathsf{end}\}$ and

$$G_{\mathrm{p}}\upharpoonright\mathtt{q} = \mathtt{p}?\big\{ l_1\langle\mathsf{bool}\rangle.\mathtt{q}!\{l[X].\mathtt{q}!\{l_2\langle ?\{l_3\langle\mathsf{api}\rangle.X\}\rangle.X\}\}\big\}$$
$$G_{\mathrm{p}}\upharpoonright\mathtt{r} = \mathtt{q}?\big\{ l[X].\mathtt{q}?\{l_2\langle ?\{l_3\langle\mathsf{api}\rangle.X\}\rangle.X\}\big\}$$

Above, the occurrences of X at the end of both $G_{\mathrm{p}}\upharpoonright\mathtt{q}$ and $G_{\mathrm{p}}\upharpoonright\mathtt{r}$ may appear surprising, as they should represent dual behaviors. Notice that in each case, X should be interpreted according to the local type that "bounds" X (i.e., the output $\mathtt{q}!\{l[X]\ldots\}$ in $G_{\mathrm{p}}\upharpoonright\mathtt{q}$ and the input $\mathtt{q}?\{l[X]\ldots\}$ in $G_{\mathrm{p}}\upharpoonright\mathtt{r}$). This dual perspective should become evident when looking at the binary session types associated to these projections. First, notice that we have that $\langle\!\langle ?\{l_3\langle\mathsf{api}\rangle.X\}\rangle\!\rangle = \&\{l_3{:}(\mathsf{api}\multimap X)\}$. Writing $(\mathsf{api}\multimap X)$ to stand for $\&\{l_3{:}(\mathsf{api}\multimap X)\}$), we have the binary session types $\langle\!\langle G_{\mathrm{p}}\upharpoonright\mathtt{p}\rangle\!\rangle = \oplus\{l_1 : \mathbf{1}\otimes\mathbf{1}\}$ and

$$\langle\!\langle G_{\mathrm{p}}\upharpoonright\mathtt{q}\rangle\!\rangle = \&\{l_1 : \mathbf{1}\multimap \oplus\{l : \exists X.\, \oplus\{l_2 : (\mathsf{api}\multimap X)\otimes X\}$$
$$\langle\!\langle G_{\mathrm{p}}\upharpoonright\mathtt{r}\rangle\!\rangle = \&\{l : \forall X.\, \&\{l_2 : (\mathsf{api}\multimap X)\multimap X\}\}\}\}$$

The medium process for $G_{\mathtt{p}}$ is then:

$$\begin{aligned}\mathsf{M}[\![G_{\mathtt{p}}]\!] = c_{\mathtt{p}} \triangleright \{ l_1 : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_1; \overline{c_{\mathtt{q}}}(v).\big([u \leftrightarrow v] \mid \\ c_{\mathtt{q}} \triangleright \{ l : c_{\mathtt{q}}(X).c_{\mathtt{r}} \triangleleft l; \overline{c_{\mathtt{r}}}\, X. \\ c_{\mathtt{q}} \triangleright \{ l_2 : c_{\mathtt{q}}(u).c_{\mathtt{r}} \triangleleft l_2; \overline{c_{\mathtt{r}}}(v).([u \leftrightarrow v] \mid \mathbf{0}) \} \} \big) \}\end{aligned}$$

Using our extended characterization results, we may show that $\mathsf{M}[\![G_{\mathtt{p}}]\!]$ can safely interact with implementations for $\mathtt{p}$, $\mathtt{q}$, and $\mathtt{r}$ whose types correspond to the projections of $G_{\mathtt{p}}$ onto $\mathtt{p}$, $\mathtt{q}$, and $\mathtt{r}$. Indeed, $\mathsf{M}[\![G_{\mathtt{p}}]\!]$ can safely interact with any P, Q_i, and R such that $\Omega; \Gamma; \Delta_1 \vdash P :: c_{\mathtt{p}}{:}\langle\!\langle G_{\mathtt{p}} \upharpoonright \mathtt{p} \rangle\!\rangle$ and

$$\Omega; \Gamma; \Delta_3 \vdash R :: c_{\mathtt{r}}{:}\langle\!\langle G_{\mathtt{p}} \upharpoonright \mathtt{r} \rangle\!\rangle \qquad \Omega; \Gamma; \Delta_2 \vdash Q_i :: c_{\mathtt{q}}{:}\langle\!\langle G_{\mathtt{p}} \upharpoonright \mathtt{q} \rangle\!\rangle$$

Process $(\boldsymbol{\nu} c_{\mathtt{p}}, c_{\mathtt{q}}, c_{\mathtt{r}})(\mathsf{M}[\![G_{\mathtt{p}}]\!] \mid P \mid R \mid Q_i)$ is a system for $G_{\mathtt{p}}$ (cf. Def. 11). It is well-typed; we have $\Omega; \Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} c_{\mathtt{p}}, c_{\mathtt{q}}, c_{\mathtt{r}})(\mathsf{M}[\![G_{\mathtt{p}}]\!] \mid P \mid R \mid Q_i) :: - : \mathbf{1}$. Process $c_{\mathtt{p}} \triangleleft l_1; \overline{c_{\mathtt{p}}}(f).(B_f \mid \mathbf{0})$ is a concrete implementation for P, where name f stands for a boolean implemented by B_f. As for R and Q_i, we may have:

$$\begin{aligned} R &= c_{\mathtt{r}} \triangleright \{ l : c_{\mathtt{r}}(Y).c_{\mathtt{r}} \triangleright \{ l_2 : c_{\mathtt{r}}(y).\overline{y}(a).(A_a \mid [c_{\mathtt{r}} \leftrightarrow a]) \} \} \\ Q_1 &= c_{\mathtt{q}} \triangleright \{ l_1 : c_{\mathtt{q}}(b).c_{\mathtt{q}} \triangleleft l; \overline{c_{\mathtt{q}}}\, S.c_{\mathtt{q}} \triangleleft l_2; \overline{c_{\mathtt{q}}}(w).(w(a).\mathtt{SMTP}^b_{w,a} \mid [m \leftrightarrow c_{\mathtt{q}}]) \} \end{aligned}$$

Crucially, following the type $\langle\!\langle G_{\mathtt{p}} \upharpoonright \mathtt{r} \rangle\!\rangle$, process R is *behaviorally generic*: independently of the type received from Q_i via the medium $\mathsf{M}[\![G_{\mathtt{p}}]\!]$ (cf. the type input prefix $c_{\mathtt{r}}(Y)$), R enables process A_a to provide the API along name a. Process Q_1 is just one possible implementation for $\mathtt{q}$: it provides an implementation of a service $\mathtt{SMTP}^b_{w,a}$ that relies on behavior api along name a and a boolean along b to implement protocol S along w. A different implementation for $\mathtt{q}$ is process Q_2 below, which concerns session protocol I:

$$Q_2 = c_{\mathtt{q}} \triangleright \{ l_1 : c_{\mathtt{q}}(b).c_{\mathtt{q}} \triangleleft l; \overline{c_{\mathtt{q}}}\, I.c_{\mathtt{q}} \triangleleft l_2; \overline{c_{\mathtt{q}}}(w).(w(a).\mathtt{IMAP}^b_{w,a} \mid [m \leftrightarrow c_{\mathtt{q}}]) \}$$

where $\mathtt{IMAP}^b_{w,a}$ uses api along a and boolean b to implement protocol I along w. Note that R and any Q_i have limited interactions with $\mathsf{M}[\![G_{\mathtt{p}}]\!]$: to respect the genericity stipulated by $G_{\mathtt{p}}$, the polymorphic process $\mathsf{M}[\![G_{\mathtt{p}}]\!]$ only mediates the exchange of the local type (S or I) and plugs the necessary connections; other exchanges are direct between R and Q_1 or Q_2, and known to comply with the (dynamically passed protocol) specified by the session type S or I.

Both $(\boldsymbol{\nu} c_{\mathtt{p}}, c_{\mathtt{q}}, c_{\mathtt{r}})(\mathsf{M}[\![G_{\mathtt{p}}]\!] \mid P \mid R \mid Q_1)$ and $(\boldsymbol{\nu} c_{\mathtt{p}}, c_{\mathtt{q}}, c_{\mathtt{r}})(\mathsf{M}[\![G_{\mathtt{p}}]\!] \mid P \mid R \mid Q_2)$ are well-typed systems; hence, they satisfy fidelity and deadlock-freedom (Thm. 1). Using properties of well-typed processes together with relational parametricity (Thm. 8), we may further show that they are *observationally equivalent*, provided a typed relation between session types S and I. That is, Thm. 8 allows us to state the behavioral independence of the sub-system formed by $\mathsf{M}[\![G_{\mathtt{p}}]\!]$, P, and R with respect to any implementation Q_i for participant $\mathtt{q}$.

7 Concluding Remarks and Related Works

We developed the first analysis of multiparty protocols using binary session types. Our *medium processes* capture the semantics of multiparty session types and connect global

types to well-typed implementations; this allows us to exploit properties for typed processes to reason about multiparty systems. Since mediums have a uniform definition, we may analyze global types with features such as delegation, which go beyond the scope of recent automata-based analyses of global types [12,16]. Our work thus complements such recent works. Our approach naturally supports the analysis of multiparty session types with *behavioral genericity*. This model, the first of its kind, is very powerful; it reuses techniques from binary sessions [4], notably *relational parametricity*. These features suggest that extensions of known multiparty sessions with behavioral genericity would be hard to obtain without following linear logic foundations, as done here.

Given a global type, our *characterization results* relate its medium and its local projections; these relations allow us to transfer properties of [5] (e.g., deadlock-freedom) to multiparty protocols. Our results stress the fundamental character of key notions in multiparty sessions (e.g., projections), and build on connections between two distinct session type theories based on linear logic [5] and on automata [12]. Our developments do not depend on the interpretation of session types in [5] being intuitionistic; clearly, its reasoning techniques (e.g., behavioral equivalences [19]) are important in our results. Our approach should extend also to interpretations based on classical linear logic [22].

Related Work. One challenge in decomposing a multiparty session type is preserving its sequencing information. The work [9] shows how to decompose a global type into simpler, independent pieces: global types use an additional `calls` construct to invoke these pieces in the appropriate order, but connections with binary sessions are not established. *Correspondence assertions* [2] track data dependencies and detect unintended operations; they may allow to relate independent binary sessions. Using standard binary/multiparty session types, we capture sequencing information using a process extracted from a global type. Our approach relies on deadlock-freedom (not available in [2]) and offers a principled way of transferring it to multiparty systems.

To our knowledge, ours is the first formal characterization of multiparty session types using binary session types. Previous works have, e.g., compared different multiparty session types but without connecting to binary types [11]. The work [18] (extended version) identifies a class of multiparty systems for which deadlock-freedom analysis can be reduced to the analysis of linear π-calculus processes. This reduction, however, does not connect with binary session types, nor exploits other properties of processes to analyze global types. The work [7] relates global types and a variant of classical linear logic; as in our work, a challenge in [7] is capturing sequencing information in global types. While [7] captures sequencing information in global types via role annotations in propositions/types (using an extra proof system, called coherence), our medium-based approach enables process reasoning on global types, uses standard linear logic propositions, and allows for conservative extensions with powerful reasoning techniques, notably behavioral genericity as enabled by parametric polymorphism.

Medium processes are loosely related to the concept of *orchestrators* in service-oriented computing. The work [17] shows how to synthesize an orchestrator from a service choreography, using finite state machines. In contrast, we consider choreographies given as behavioral types; mediums are obtained directly from those types.

Acknowledgments. Thanks to Bernardo Toninho for useful discussions. We are also grateful to the anonymous reviewers for their improvement suggestions. This work was

partially supported by NOVA LINCS (Ref. UID/CEC/04516/2013) and COST Action IC1201 (Behavioural Types for Reliable Large-Scale Software Systems).

References

1. L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR'2010*, LNCS, pages 162–176. Springer, 2010.
2. E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence assertions for process synchronization in concurrent communications. *J. Funct. Program.*, 15:219–247, 2005.
3. L. Caires and J. A. Pérez. A typeful characterization of multiparty structured conversations based on binary sessions. *CoRR*, abs/1407.4242, 2014.
4. L. Caires, J. A. Pérez, F. Pfenning, and B. Toninho. Behavioral polymorphism and parametricity in session-based communication. In *ESOP*, volume 7792 of *LNCS*, pages 330–349. Springer, 2013. See also Technical Report CMU-CS-12-108, April 2012.
5. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR'2010*, LNCS, pages 222–236. Springer, 2010.
6. M. Carbone and F. Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274. ACM, 2013.
7. M. Carbone, F. Montesi, C. Schürmann, and N. Yoshida. Multiparty session types as coherence proofs. In *CONCUR 2015*, volume 42 of *LIPIcs*, pages 412–426. Dagstuhl, 2015.
8. G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. In *POPL*, ACM SIGPLAN Notices 43, pages 261–272. ACM, 2008.
9. T. Chen. Lightening global types. *J. Log. Algebr. Meth. Program.*, 84(5):708–729, 2015.
10. M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani. Global progress for dynamically interleaved multiparty sessions. *Math. Struc. in Comp. Sci.*, 26(2):238–302, 2016.
11. R. Demangeon and N. Yoshida. On the expressiveness of multiparty session types. In *FSTTCS 2015*, LIPIcs. Dagstuhl, 2015.
12. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP'13*. Springer, 2013.
13. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP'98*, LNCS. Springer, 1998.
14. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.
15. H. Huttel et al. Foundations of session types and behavioural contracts. *ACM Computing Surveys*, 2016. To appear.
16. J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *Proc. of POPL 2015*, pages 221–232. ACM, 2015.
17. S. McIlvenna, M. Dumas, and M. T. Wynn. Synthesis of orchestrators from service choreographies. In *APCCM*, volume 96 of *CRPIT*. Australian Computer Society, 2009.
18. L. Padovani. Deadlock and lock freedom in the linear π-calculus. In *Proc. of CSL-LICS'14*, pages 72:1–72:10. ACM, 2014. Extended version at `http://hal.archives-ouvertes.fr/hal-00932356v2/document`.
19. J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Inf. Comput.*, 239:254–302, 2014.
20. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. CUP, 2001.
21. B. Toninho, L. Caires, and F. Pfenning. Corecursion and Non-Divergence in Session Types. In *TGC 2014*, volume 8902 of *LNCS*, pages 159–175. Springer, 2014.
22. P. Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014.