Domain-Aware Session Types

Luís Caires
Universidade Nova de Lisboa

Jorge A. Pérez
University of Groningen

Frank Pfenning
Carnegie Mellon University

Bernardo Toninho
Universidade Nova de Lisboa

Abstract

We develop a generalization of existing Curry-Howard interpretations of (binary) session types by relying on an extension of linear logic with features from *hybrid logic*, in particular modal worlds that indicate *domains*. These worlds govern *domain migration*, subject to a parametric accessibility relation familiar from the Kripke semantics of modal logic. The result is an expressive new typed process framework for domain-aware, message-passing concurrency. Its logical foundations ensure that well-typed processes enjoy session fidelity, global progress, and termination. Typing also ensures that processes only communicate with accessible domains and so respect the accessibility relation.

Remarkably, our domain-aware framework can specify scenarios in which domain information is available only at runtime; flexible accessibility relations can be cleanly defined and statically enforced. As a specific application, we introduce domain-aware *multiparty session types*, in which global protocols can express arbitrarily nested sub-protocols via domain migration. We develop a precise analysis of these multiparty protocols by reduction to our binary domain-aware framework: complex domain-aware protocols can be reasoned about at the right level of abstraction, ensuring also the principled transfer of key correctness properties from the binary to the multiparty setting.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Type structures; Software and its engineering → Message passing

Keywords and phrases Session Types, Linear Logic, Process Calculi, Hybrid Logic

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The goal of this paper is to show how existing Curry-Howard interpretations of session types [6, 7] can be generalized to a *domain-aware* setting by relying on an extension of linear logic with features from *hybrid logic* [35, 2]. As we discuss next, these extended logical foundations of message-passing concurrency allow us to analyze complex domain-aware concurrent systems (including those governed by multiparty protocols) in a precise and principled manner.

Software systems typically rely on *communication* between heterogeneous services; at their heart, these systems rely on message-passing protocols that combine mobility, concurrency, and distribution. As distributed services are often virtualized, protocols should span diverse software and hardware *domains*. These domains can have multiple interpretations, such as the location where services reside, or the principals on whose behalf they act. Concurrent behavior is then increasingly *domain-aware*: a partner's potential for interaction is influenced not only by the domains it is involved in at various protocol phases (its context), but also by *connectedness* relations among domains. Moreover, domain architectures are rarely fully specified: to aid modularity and platform independence, system participants (e.g., developers, platform vendors, service clients) often have only partial views of actual domain structures.

Despite their importance in communication correctness and trustworthiness at large, the formal status of domains within models of message-passing systems remains unexplored.

This paper contributes to typed approaches to the analysis of domain-aware communications, with a focus on *session-based concurrency*. This approach specifies the intended message-passing protocols as *session types* [23, 24, 19]. Different type theories for *binary* and *multiparty* (n-ary) protocols have been developed. In both cases, typed specifications can be conveniently coupled with π-calculus processes [30], in which so-called session channels connect exactly two subsystems. Communication correctness usually results from two properties: *session fidelity* (type preservation) and *deadlock freedom* (progress). The former says that well-typed processes always evolve to well-typed processes (a safety property); the latter says that well-typed processes will never get into a stuck state (a liveness property).

A key motivation for this paper is the sharp contrast between (a) the growing relevance of domain-awareness in message-passing, concurrent systems and (b) the expressiveness of existing session type frameworks, binary and multiparty, which cannot adequately specify (let alone enforce) domain-related requirements. Indeed, existing session types frameworks, including those based on Curry-Howard interpretations [6, 43, 10], capture communication behavior at a level of abstraction in which even basic domain-aware assertions (e.g., "*Shipper* resides in domain `AmazonUS`") cannot be expressed. As an unfortunate consequence, the effectiveness of the analysis techniques derived from these frameworks is rather limited.

To better illustrate our point, consider a common distributed design pattern: a middleware agent (`mw`) which answers requests from clients (`cl`), sometimes offloading the requests to a server (`serv`) to better manage local resource availability. In the framework of multiparty session types [25] this can be represented as:

$$\texttt{cl} \twoheadrightarrow \texttt{mw}\!:\!\{\mathit{request}\langle \mathit{req}\rangle.\ \texttt{mw} \twoheadrightarrow \texttt{cl}\!:\!\{\ \mathit{reply}\langle \mathit{ans}\rangle.\texttt{mw} \twoheadrightarrow \texttt{serv}\!:\!\{\mathit{done}.\texttt{end}\},\ \mathit{wait}.\texttt{mw} \twoheadrightarrow \texttt{serv}\!:\!\{\mathit{req}\langle \mathit{data}\rangle.\texttt{serv} \twoheadrightarrow \texttt{mw}\!:\!\{\mathit{reply}\langle \mathit{ans}\rangle.\texttt{mw} \twoheadrightarrow \texttt{cl}\!:\!\{\mathit{reply}\langle \mathit{ans}\rangle.\texttt{end}\}\}\}\}\}$$

The client first sends a request to the middleware, which answers back with either a *reply* message containing the answer or a *wait* message, signaling that the server will be contacted to produce the final *reply*. While this multiparty protocol captures the intended communication behavior, a realistic mediation between the middleware and the server often involves some form of privilege escalation or specialized authentication—ensuring, e.g., that the server interaction is adequately isolated from the client, or that the escalation must precede the server interactions. This privilege escalation simply cannot be represented in existing frameworks.

Our work addresses this crucial limitation by generalizing Curry-Howard interpretations of session types by appealing to hybrid logic features. We develop a logically motivated typed process framework in which *worlds* from modal logics precisely and uniformly define the notion of domain in session-based concurrency. At the level of *binary* sessions, domains manifest themselves through point-to-point domain migration and communication. In *multiparty* sessions, domain migration is specified choreographically through the new construct $\mathsf{p}\ \mathsf{moves}\ \tilde{\mathsf{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$, where participant $\mathtt{p}$ leads a migration of participants $\tilde{\mathtt{q}}$ to domain (world) ω in order to perform protocol G_1, who then migrate back to perform protocol G_2.

Returning to our example, let $\mathit{Offload} \triangleq \texttt{mw} \twoheadrightarrow \texttt{serv}\!:\!\{\mathit{req}\langle \mathit{data}\rangle.\texttt{serv} \twoheadrightarrow \texttt{mw}\!:\!\{\mathit{reply}\langle \mathit{ans}\rangle.\texttt{end}\}\}$. Our framework allows us to refactor the above protocol as:

$$\texttt{cl} \twoheadrightarrow \texttt{mw}\!:\!\{\mathit{request}\langle \mathit{req}\rangle.\ \texttt{mw} \twoheadrightarrow \texttt{cl}\!:\!\{\ \mathit{reply}\langle \mathit{ans}\rangle.\texttt{mw} \twoheadrightarrow \texttt{serv}\!:\!\{\mathit{done}.\texttt{end}\},\ \mathit{wait}.\texttt{mw} \twoheadrightarrow \texttt{serv}\!:\!\{\mathit{init}.\ \texttt{mw}\ \mathsf{moves}\ \texttt{serv}\ \mathsf{to}\ w_{\texttt{priv}}\ \mathsf{for}\ \mathit{Offload}\ ;\ \texttt{mw} \twoheadrightarrow \texttt{cl}\!:\!\{\mathit{reply}\langle \mathit{ans}\rangle.\texttt{end}\}\}\}\}$$

By considering a first-class multiparty domain migration primitive at the type and process levels, we can specify that the *offload* portion of the protocol takes place after the middleware and the server *migrate* to a private domain $w_{\texttt{priv}}$, as well as ensuring that only reachable

domains can be interacted with. For instance, the type for the server that is mechanically *projected* from the protocol above ensures that the server first migrates to the private domain, communicates with the middleware, and then migrates back to its initial domain.

Perhaps surprisingly, our domain-aware *multiparty* sessions are studied within a context of logical *binary* domain-aware sessions, arising from a propositions-as-types interpretation of hybrid linear logic [17, 13], with strong *static* correctness guarantees derived from the logical nature of the system. Multiparty domain-awareness arises through an interpretation of multiparty protocols as *medium processes* [4] that orchestrate the multiparty interaction while enforcing the necessary domain-level constraints and migration steps.

Contributions The key contributions of this work are:

1. A process model with explicit domain-based migration (§ 2). We present a session π-calculus with domains that can be communicated via novel domain movement prefixes.
2. A logic-based session type discipline over domain-aware interacting processes (§ 3). Building upon an extension of linear logic with features from *hybrid logic* [17, 13] we generalize the Curry-Howard interpretation of session types [6], by interpreting *(modal) worlds* as *domains* where session behavior resides. Types include operators that specify domain *migration* and *communication*. In our system, judgments stipulate the services that processes implement *and* the domains where sessions should be present; domain mobility is governed by a parametric accessibility relation. Our type discipline statically enforces session fidelity, global progress and, notably, that communication can only happen between reachable domains, thus introducing a new layer of reasoning to session type systems.
3. A systematic study of domain-aware multiparty sessions (§ 4), extending the standard multiparty session framework of [25] with domain-aware migration and communication primitives, at both the global orchestration and local typing levels. As a specific application, our development leverages our logically motivated domain-aware *binary* sessions to give a precise semantics to multiparty sessions through a notion of a typed medium process that acts as an orchestrator of the domain-aware multiparty interaction, inheriting the strong correctness properties of typed processes but now in a multiparty setting. Crucially, we show that mediums soundly and completely encode the local behaviors of participants in a domain-aware multiparty session.

We conclude with a discussion of related work (§ 5) and concluding remarks (§ 6). Appendix A lists omitted definitions and proofs. We point the interested reader to Appendices B and C for extended examples on domain-aware multiparty and binary sessions, respectively.

2 Process Model

We introduce a variant of the synchronous π-calculus [37] with labeled choice and explicit domain migration and communication. We write $\omega, \omega', \omega''$ to stand for a concrete domain $(w, w', \ldots)$ or a domain variable $(\alpha, \alpha', \ldots)$. Domains are handled at a high-level of abstraction, with their identities being attached to session channels. Just as the π-calculus allows for communication over names and name mobility, our model also allows for domain communication and mobility. These features are justified with the typing discipline of § 3.

▶ **Definition 2.1.** *Given infinite, disjoint sets* Λ *of* names (x, y, z, u, v)*,* $\mathcal{L}$ *of labels* $l_1, l_2, \ldots$*,* $\mathcal{W}$ *of* domain tags (w, w', w'') *and* $\mathcal{V}$ *of* domain variables (α, β, γ)*, respectively, the set of* processes (P, Q, R) *is defined by*

$$
\begin{array}{rcllllllll}
P & ::= & \mathbf{0} & \mid & P \mid Q & \mid & (\boldsymbol{\nu} y)P & \mid & x\langle y\rangle.P & \mid & x(y).P & \mid & !x(y).P \\
 & \mid & [x \leftrightarrow y] & \mid & x \triangleright \{l_i : P_i\}_{i \in I} & \mid & x \triangleleft l_i; P & & & & & & \\
 & \mid & x\langle y@\omega\rangle.P & \mid & x(y@\omega).P & \mid & x\langle \omega\rangle.P & \mid & x(\alpha).P & & & &
\end{array}
$$

Domain-aware prefixes are present only in the last line. As we make precise in the typed setting of § 3, these constructs realize mobility and domain communication, in the usual sense of the π-calculus: migration to a domain is always associated to mobility with a fresh name.

The operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition) and $(\boldsymbol{\nu} y)P$ (name restriction) are standard. We then have $x\langle y\rangle.P$ (send y on x and proceed as P), $x(y).P$ (receive z on x and proceed as P with parameter y replaced by z), and $!x(y).P$ which denotes replicated (persistent) input. The forwarding construct $[x \leftrightarrow y]$ equates x and y; it is a primitive representation of a copycat process. The last two constructs in the second line define a labeled choice mechanism: $x \triangleright \{l_i : P_i\}_{i\in I}$ is a process that awaits some label l_j (with $j \in I$) and proceeds as P_j. Dually, the process $x \triangleleft l_i; P$ emits a label l_i and proceeds as P.

The first two operators in the third line define explicit domain migration: given a domain ω, $x\langle y@\omega\rangle.P$ denotes a process that is prepared to migrate the communication actions in P on endpoint x, to session y on ω. Complementarily, process $x(y@\omega).P$ signals an endpoint x to move to w, providing P with the appropriate session endpoint that is then bound to y. In a typed setting, domain movement will be always associated with a fresh session channel. Alternatively, this form of coordinated migration can be read as an explicit form of agreement (or authentication) in trusted domains. Finally, the last two operators in the third line define output and input of domains, $x\langle\omega\rangle.P$ and $x(\alpha).P$, respectively. These constructs allow for domain information to be obtained and propagated across processes dynamically.

Following [36], we abbreviate $(\boldsymbol{\nu} y)x\langle y\rangle$ and $(\boldsymbol{\nu} y)x\langle y@\omega\rangle$ as $\overline{x}\langle y\rangle$ and $\overline{x}\langle y@\omega\rangle$, respectively. In $(\boldsymbol{\nu} y)P$, $x(y).P$, and $x(y@\omega).P$ the distinguished occurrence of name y is binding with scope P. Similarly for α in $x(\alpha).P$. We identify processes up to consistent renaming of bound names and variables, writing $\equiv_\alpha$ for this congruence. $P\{x/y\}$ denotes the capture-avoiding substitution of x for y in P. While *structural congruence* $\equiv$ expresses standard identities on the basic structure of processes (cf. Appendix A.1), *reduction* expresses their behavior.

Reduction ($P \to Q$) is the binary relation defined by the rules below; it specifies the computations that a process performs on its own.

$$
\begin{array}{ll}
x\langle y\rangle.Q \mid x(z).P \to Q \mid P\{y/z\} & x\langle y\rangle.Q \mid !x(z).P \to Q \mid P\{y/z\} \mid !x(z).P \\
x\langle y@\omega\rangle.P \mid x(z@\omega').Q \to P \mid Q\{y/z\} & x\langle\omega\rangle.P \mid x(\alpha).Q \to P \mid Q\{\omega/\alpha\} \\
(\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \to P\{y/x\} & Q \to Q' \Rightarrow P \mid Q \to P \mid Q' \\
P \to Q \Rightarrow (\boldsymbol{\nu} y)P \to (\boldsymbol{\nu} y)Q & P \equiv P' \wedge P' \to Q' \wedge Q' \equiv Q \Rightarrow P \to Q \\
x \triangleleft l_j; P \mid x \triangleright \{l_i : Q_i\}_{i\in I} \to P \mid Q_j \quad (j \in I) &
\end{array}
$$

For the sake of generality, reduction allows dual endpoints with the same name to interact, independently of the domains of their subjects. The type system introduced next will ensure, among other things, *local reductions*, disallowing synchronisations among distinct domains.

3 Domain-aware Session Types via Hybrid Logic

This section develops a new domain-aware formulation of binary session types. Our system is based on a Curry-Howard interpretation of a linear variant of so-called hybrid logic, and can be seen as an extension of the interpretation of [6] to hybrid (linear) logic. Hybrid logic is often used as an umbrella term for a class of logics that extend the expressiveness of propositional logic by considering modal *worlds* as *syntactic objects* that occur in propositions.

As in [6, 7], propositions are interpreted as session types of communication channels, proofs as process typing derivations, and proof reduction as process communication. The main novelties of our approach are that we interpret worlds as domains, the hybrid connective $@_w\, A$ as the type of a session that *migrates* to an accessible domain w, and type-level

quantification over worlds $\forall\alpha.A$ and $\exists\alpha.A$ as *world communication*. We also consider a type-level operator $\downarrow\alpha.A$ (read "here") which binds the *current* domain of the session to α in A. The syntax of domain-aware session types is given in Def. 3.1, where $w, w_1, \ldots$ stand for worlds drawn from $\mathcal{W}$, and where α, β and ω, ω' are used as in the syntax of processes.

▶ **Definition 3.1** (Domain-aware Session Types). *The syntax of types* (A, B, C) *is defined by*

$$\begin{array}{rcl} A & ::= & \mathbf{1} \mid A \multimap B \mid A \otimes B \mid \&\{l_i : A_i\}_{i\in I} \mid \oplus\{l_i : A_i\}_{i\in I} \mid !A \\ & \mid & @_\omega\, A \mid \forall\alpha.A \mid \exists\alpha.A \mid \downarrow\alpha.A \end{array}$$

Types are the propositions of intuitionistic linear logic where the additives $A\&B$ and $A\oplus B$ are generalized to a labelled n-ary variant. Propositions take the standard interpretation as session types, extended with hybrid logic operators [2], where modal worlds are subject to an *accessibility relation* that is tracked by a separate context Ω. Intuitively, Ω collects direct reachability hypotheses of the form $\omega_1 \prec \omega_2$, meaning that domain ω_2 is reachable from ω_1.

In our setting, types are assigned to channel names; a *type assignment* $x{:}A[\omega]$ enforces the use of name x according to session A, *in the domain* ω. A *type environment* is a collection of type assignments. Besides the accessibility context Ω, our typing judgments consider two kinds of type environments, subject to different structural properties: a *linear* part Δ and an *unrestricted* part Γ. Our typing system is made up of two judgments:

$$\text{(i)}\ \ \Omega \vdash \omega_1 \prec \omega_2 \qquad \text{and} \qquad \text{(ii)}\ \ \Omega; \Gamma; \Delta \vdash P :: z{:}A[\omega]$$

Judgment (i) states that ω_1 can directly reach ω_2 under the hypotheses in Ω. We omit Ω when it is clear from context. We write $\prec^*$ for the *reflexive, transitive closure* of $\prec$, and $\omega_1 \not\prec^* \omega_2$ when $\omega_1 \prec^* \omega_2$ does not hold. Judgment (ii) states that process P offers the session behavior specified by type A on channel z, with the session being localized at domain ω, under the reachability hypotheses Ω, using *unrestricted* sessions in Γ, and *linear* sessions in Δ, where weakening and contraction principles hold for Γ but not Δ. Note that each hypothesis in Γ and Δ is labeled with a specific domain. Empty contexts are written as '$\cdot$'.

Typing Rules Most typing rules are given in Fig. 1; the rest are listed in Appendix A.3. Right rules (marked with R) specify how to *offer* a session of a given type, left rules (marked with L) define how to *use* a session. The hybrid nature of the system induces a notion of *well-formedness* of sequents: we consider a sequent $\Omega; \Gamma; \Delta \vdash P :: z : C[\omega_1]$ *well-formed* if $\Omega \vdash \omega_1 \prec^* \omega_2$ for every $x{:}A[\omega_2] \in \Delta$, which we abbreviate as $\Omega \vdash \omega_1 \prec^* \Delta$, meaning that all worlds mentioned in Δ are reachable from ω_1 (not necessarily in a single *direct* step). No such world requirement is imposed on Γ. If an end sequent is well-formed, every sequent in its proof will also be well-formed. All rules (read bottom-up) preserve this invariant; only (cut), (copy), (@R), ($\forall$L) and ($\exists$R) require explicit reachability checks, which we discuss below. This invariant statically excludes interaction between sessions in unreachable domains (cf. Theorem 3.7).

We briefly discuss some of the typing rules, first noting that we always consider processes modulo structural congruence; hence, typability is closed under $\equiv$ by definition. Type $A \multimap B$ denotes a session that inputs a session of type A and proceeds as B. To offer $z{:}A \multimap B$ at domain ω, we input y along z that will offer A at ω and proceed, now offering $z{:}B$ at ω:

$$(\multimap\mathsf{R})\ \frac{\Omega; \Gamma; \Delta, y{:}A[\omega] \vdash P :: z{:}B[\omega]}{\Omega; \Gamma; \Delta \vdash z(y).P :: z{:}A \multimap B[\omega]} \qquad (\otimes\mathsf{R})\ \frac{\Omega; \Gamma; \Delta_1 \vdash P :: y{:}A[\omega] \quad \Omega; \Gamma; \Delta_2 \vdash Q :: z{:}B[\omega]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash \overline{z}\langle y\rangle.(P \mid Q) :: z{:}A \otimes B[\omega]}$$

Dually, $A \otimes B$ denotes a session that outputs a session that will offer A and continue as B. To offer $z{:}A \otimes B$, we perform an output along z of a fresh name y, a session of type A provided by P, and proceed as Q, offering $z{:}B$.

The (cut) rule allows us to compose process P offering $x{:}A[\omega_2]$ with Q using $x{:}A[\omega_2]$ to offer $z{:}C[\omega_1]$, provided that the two worlds ω_1 and ω_2 are reachable (i.e., $\omega_1 \prec^* \omega_2$). Composition binds the name x. Note that we require domains in Δ_1, the ambient sessions of the first premise, to be accessible from ω_1, the domain of the second premise, which follows from the use of the transitive closure of accessibility $\prec^*$, using the intermediary w_2:

$$(\mathsf{cut})\ \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega \vdash \omega_1 \prec^* \Delta_1 \quad \Omega;\Gamma;\Delta_1 \vdash P :: x{:}A[\omega_2] \quad \Omega;\Gamma;\Delta_2, x{:}A[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid Q) :: z{:}C[\omega_1]}$$

Type $\mathbf{1}$ means that no further interaction will take place on the session; names of type $\mathbf{1}$ may be passed around as opaque values. $\&\{l_i : A_i\}_{i\in I}$ types a session channel that offers its partner a choice between the A_i behaviors, each uniquely identified by a label l_i. Dually, $\oplus\{l_i : A_i\}_{i\in I}$ types a session that selects some behavior A_i by emitting the corresponding label. Type $!A$ types a shared (non-linear) channel, to be used by a server for spawning an arbitrary number of new sessions (possibly none), each one conforming to type A.

Following our previous remark on well-formed sequents, the only rules that appeal to accessibility are (@R), (@L), (copy), and (cut). These conditions are directly associated with varying degrees of flexibility in terms of typability, depending on what relationship is imposed between the world to the left and to the right of the turnstile in the left rules. Notably, our system leverages the accessibility judgment to enforce that communication is only allowed between processes whose sessions are in (transitively) *reachable* domains.

The type operator $@_\omega$ realizes a *domain migration* mechanism which is specified both at the level of types and of processes via name mobility tagged with a domain name. Thus, a channel typed with $@_{\omega_2}A$ denotes that behavior A is available by first *moving to* domain ω_2, directly accessible from the current domain. More precisely, we have:

$$(@\mathsf{R})\ \frac{\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega \vdash \omega_2 \prec^* \Delta \quad \Omega;\Gamma;\Delta \vdash P :: y{:}A[\omega_2]}{\Omega;\Gamma;\Delta \vdash \overline{z}\langle y@\omega_2\rangle.P :: z{:}@_{\omega_2}A[\omega_1]} \qquad (@\mathsf{L})\ \frac{\Omega,\omega_2 \prec \omega_3;\Gamma;\Delta,\ y{:}A[\omega_3] \vdash P :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta, x{:}@_{\omega_3}A[\omega_2] \vdash x(y@\omega_3).P :: z{:}C[\omega_1]}$$

Hence, a process *offering* a behavior $z{:}@_{\omega_2}A$ at ω_1 ensures: (i) behavior A is available at ω_2 along a *fresh* session channel y that is emitted along z and (ii) ω_2 is directly reachable from ω_1. To maintain well-formedness of the sequent we also must check that all domains in Δ are still accessible from ω_2. Dually, *using* a service $x{:}@_{\omega_3}A[\omega_2]$ entails receiving a channel y that will offer behavior A at domain ω_3 (and also allowing the usage of the fact that $\omega_2 \prec \omega_3$).

Domain-quantified sessions introduce domains as *fresh* parameters to types: a particular service can be specified with the ability to refer to any existing directly reachable domain (via universal quantification) or to some *a priori* unspecified reachable domain:

$$(\forall\mathsf{R})\ \frac{\Omega,\omega_1 \prec \alpha;\Gamma;\Delta \vdash P :: z{:}A[\omega_1] \quad \alpha \notin \Omega,\Gamma,\Delta,\omega_1}{\Omega;\Gamma;\Delta \vdash z(\alpha).P :: z{:}\forall\alpha.A[\omega_1]} \qquad (\forall\mathsf{L})\ \frac{\Omega \vdash \omega_2 \prec \omega_3 \quad \Omega;\Gamma;\Delta,\ x{:}A\{\omega_3/\alpha\}[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta, x{:}\forall\alpha.A[\omega_2] \vdash x\langle\omega_3\rangle.Q :: z{:}C[\omega_1]}$$

Rule (∀R) states that a process seeking to offer $\forall\alpha.A[\omega_1]$ denotes a service that is located at domain ω_1 but that may refer to any fresh domain directly reachable from ω_1 in its specification (e.g. through the use of @). Operationally, this means that the process must be ready to receive from its client a reference to the domain being referred to in the type, which is bound to α (occurring fresh in the typing derivation). Dually, Rule (∀L) indicates that a process interacting with a service of type $x{:}\forall\alpha.A[\omega_2]$ must make concrete the domain that is directly reachable from ω_2 it wishes to use, which is achieved by the appropriate output action. Rules (∃L) and (∃R) for the existential quantifier have a dual reading.

Finally, we introduce a type-level operator $\downarrow\alpha.A$ which allows for a session type to refer to its *current* domain in a dynamic way:

$$(\downarrow\mathsf{R})\ \frac{\Omega;\Gamma;\Delta \vdash P :: z{:}A\{\omega/\alpha\}[\omega]}{\Omega;\Gamma;\Delta \vdash P :: z{:}{\downarrow}\alpha.A[\omega]} \qquad (\downarrow\mathsf{L})\ \frac{\Omega;\Gamma;\Delta, x{:}A\{\omega/\alpha\}[\omega] \vdash P :: z{:}C}{\Omega;\Gamma;\Delta, x{:}{\downarrow}\alpha.A[\omega] \vdash P :: z{:}C}$$

The typing rules that govern $\downarrow\alpha.A$ are completely symmetric and produce no action at the process level, merely instantiating the world variable α with the current domain ω of the session. As will be made clear in § 4, this connective plays a crucial role in ensuring the correctness of our analysis of multiparty domain-aware sessions in our logical setting.

By developing our type theory with a separate domain accessibility judgment, we can consider a particular accessibility relation as a *parameter* of the framework. This allows changing accessibility relations and its properties without having to alter the entire system. To consider the simplest possible accessibility relation, the only defining rule for accessibility would be Rule (whyp) in Fig. 1. To consider an accessibility relation which is an equivalence relation we would add reflexivity, transitivity, and symmetry rules to the judgment.

Discussion and Examples Crucially, our domain-aware theory is *conservative* with respect to the Curry-Howard interpretation of session types in [6, 7]: our type theory is a process interpretation of *hybridized* linear logic (whose rules can be recovered by erasing the process terms), of which [6, 7] is a special case where every session resides at the same domain.

Conversely, a fundamental consequence of our hybrid interpretation is that it *refines* the session type structure in non-trivial ways. By requiring that communication only happen between processes located at the same (or reachable) worlds we effectively introduce a new layer of reasoning to session type systems. To illustrate this feature, consider the following session type, specifying a simplified interaction between a web store and its clients:

$$\mathsf{WStore} \triangleq \mathsf{addCart} \multimap \&\{\mathit{buy} : \mathsf{Pay}\,,\ \mathit{quit} : \mathbf{1}\} \qquad \mathsf{Pay} \triangleq \mathsf{CCNum} \multimap \oplus\{\mathit{ok} : \mathsf{Rcpt} \otimes \mathbf{1}\,,\ \mathit{nok} : \mathbf{1}\}$$

The web store specification, representable in existing session type systems (e.g. [6, 43, 24]), allows clients to checkout their shopping carts by emitting a *buy* message or to *quit*. In the former case, the client pays for the purchase by sending their credit card data. Despite describing the intended communication patterns correctly, the types above do not capture the crucial fact that in a realistic setting, the client's sensitive information should only be requested after entering a secure domain. We can address this limitation, by using the type-level domain migration construct, refining the types WStore and Pay above as follows:

$$\mathsf{WStore}_{\mathtt{sec}} \triangleq \mathsf{addCart} \multimap \&\{\mathit{buy} : @_{\mathtt{sec}}\,\mathsf{Pay}, \mathtt{quit} : \mathbf{1}\} \quad \mathsf{Pay} \triangleq \mathsf{CCNum} \multimap \oplus\{\mathit{ok} : (@_{\mathtt{bnk}}\mathsf{Rcpt}) \otimes \mathbf{1}, \mathit{nok} : \mathbf{1}\}$$

$\mathsf{WStore}_{\mathtt{sec}}$ now decrees that communication behavior pertinent to type Pay should be preceded by a migration step to the trusted domain `sec`, which should be directly reachable from $\mathsf{WStore}_{\mathtt{sec}}$'s current domain. The type can also specify that the receipt must originate from a bank domain `bnk` (e.g., ensuring the client that it is not produced by the store without entering the bank domain). When considering the interactions with a client (at domain `c`) that checks out their cart, we reach a state that is typed with the judgment below (left side):

$$\mathtt{c} \prec \mathtt{ws}; \cdot; x{:}@_{\mathtt{sec}}\mathsf{Pay}[\mathtt{ws}] \vdash \mathit{Client} :: z{:}@_{\mathtt{sec}}\mathbf{1}[\mathtt{c}] \quad \mathtt{c} \prec \mathtt{ws}, \mathtt{ws} \prec \mathtt{sec}; \cdot; x'{:}\mathsf{Pay}[\mathtt{sec}] \vdash \mathit{Client}' :: z'{:}\mathbf{1}[\mathtt{sec}]$$

At this point, it is *impossible* for a (typed) client to interact with the behavior that is protected by the domain `sec`, since it is not the case that $\mathtt{c} \prec^* \mathtt{sec}$. This ensures, e.g., that a client cannot exploit the payment platform of the web store by accessing the trusted domain in unforeseen ways. The client can only communicate in the secure domain *after* the web store service has migrated accordingly, as shown by the judgment above (right side).

$$(\mathsf{whyp})\ \frac{}{\Omega, \omega_1 \prec \omega_2 \vdash \omega_1 \prec \omega_2} \qquad (\mathsf{id})\ \frac{}{\Omega; \Gamma; x{:}A[\omega] \vdash [x \leftrightarrow z] :: z{:}A[\omega]}$$

$$(\mathbf{1}\mathsf{L})\ \frac{\Omega; \Gamma; \Delta \vdash P :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}\mathbf{1}[\omega_2] \vdash P :: z{:}C[\omega_1]} \qquad (\mathbf{1}\mathsf{R})\ \frac{}{\Omega; \Gamma; \cdot \vdash \mathbf{0} :: x{:}\mathbf{1}[\omega]}$$

$$(\multimap\mathsf{R})\ \frac{\Omega; \Gamma; \Delta, y{:}A[\omega] \vdash P :: z{:}B[\omega]}{\Omega; \Gamma; \Delta \vdash z(y).P :: z{:}A \multimap B[\omega]} \qquad (\otimes\mathsf{R})\ \frac{\Omega; \Gamma; \Delta_1 \vdash P :: y{:}A[\omega] \quad \Omega; \Gamma; \Delta_2 \vdash Q :: z{:}B[\omega]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash \overline{z}\langle y\rangle.(P \mid Q) :: z{:}A \otimes B[\omega]}$$

$$(\otimes\mathsf{L})\ \frac{\Omega; \Gamma; \Delta, y{:}A[\omega_2], x{:}B[\omega_2] \vdash P :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}A \otimes B[\omega_2] \vdash x(y).P :: z{:}C[\omega_1]} \qquad (\multimap\mathsf{L})\ \frac{\Omega; \Gamma; \Delta_1 \vdash P :: y{:}A[\omega_2] \quad \Omega; \Gamma; \Delta_2,\ x{:}B[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta_1, \Delta_2, x{:}A \multimap B[\omega_2] \vdash \overline{x}\langle y\rangle.(P \mid Q) :: z{:}C[\omega_1]}$$

$$(@\mathsf{R})\ \frac{\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega \vdash \omega_2 \prec^* \Delta \quad \Omega; \Gamma; \Delta \vdash P :: y{:}A[\omega_2]}{\Omega; \Gamma; \Delta \vdash \overline{z}\langle y@\omega_2\rangle.P :: z{:}@_{\omega_2} A[\omega_1]} \qquad (@\mathsf{L})\ \frac{\Omega, \omega_2 \prec \omega_3; \Gamma; \Delta,\ y{:}A[\omega_3] \vdash P :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}@_{\omega_3} A[\omega_2] \vdash x(y@\omega_3).P :: z{:}C[\omega_1]}$$

$$(\forall\mathsf{R})\ \frac{\Omega, \omega_1 \prec \alpha; \Gamma; \Delta \vdash P :: z{:}A[\omega_1] \quad \alpha \notin \Omega, \Gamma, \Delta, \omega_1}{\Omega; \Gamma; \Delta \vdash z(\alpha).P :: z{:}\forall\alpha.A[\omega_1]} \qquad (\forall\mathsf{L})\ \frac{\Omega \vdash \omega_2 \prec \omega_3 \quad \Omega; \Gamma; \Delta,\ x{:}A\{\omega_3/\alpha\}[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}\forall\alpha.A[\omega_2] \vdash x\langle\omega_3\rangle.Q :: z{:}C[\omega_1]}$$

$$(\exists\mathsf{R})\ \frac{\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega; \Gamma; \Delta \vdash P :: z{:}A\{\omega_2/\alpha\}[\omega_1]}{\Omega; \Gamma; \Delta \vdash z\langle\omega_2\rangle.P :: z{:}\exists\alpha.A[\omega_1]} \qquad (\exists\mathsf{L})\ \frac{\Omega, \omega_2 \prec \alpha; \Gamma; \Delta,\ x{:}A[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}\exists\alpha.A[\omega_2] \vdash x(\alpha).Q :: z{:}C[\omega_1]}$$

$$(\downarrow\mathsf{R})\ \frac{\Omega; \Gamma; \Delta \vdash P :: z{:}A\{\omega/\alpha\}[\omega]}{\Omega; \Gamma; \Delta \vdash P :: z{:}\downarrow\alpha.A[\omega]} \qquad (\downarrow\mathsf{L})\ \frac{\Omega; \Gamma; \Delta, x{:}A\{\omega/\alpha\}[\omega] \vdash P :: z{:}C}{\Omega; \Gamma; \Delta, x{:}\downarrow\alpha.A[\omega] \vdash P :: z{:}C}$$

$$(\mathsf{copy})\ \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega; \Gamma, u{:}A[\omega_2]; \Delta, y{:}A[\omega_2] \vdash P :: z{:}C[\omega_1]}{\Omega; \Gamma, u{:}A[\omega_2]; \Delta \vdash \overline{u}\langle y\rangle.P :: z{:}C[\omega_1]}$$

$$(\mathsf{cut})\ \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega \vdash \omega_2 \prec^* \Delta_1 \quad \Omega; \Gamma; \Delta_1 \vdash P :: x{:}A[\omega_2] \quad \Omega; \Gamma; \Delta_2, x{:}A[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid Q) :: z{:}C[\omega_1]}$$

Figure 1 Typing Rules (Excerpt – see Appendix A.3)

Technical Results We state the main results of type safety via type preservation (Theorem 3.3) and global progress (Theorem 3.4). These results directly ensure session fidelity and deadlock-freedom. Typing also ensures that termination, i.e., processes do not exhibit infinite reduction paths (Theorem 3.5). Moreover, as a property specific to domain-aware processes, we show *domain preservation*, i.e., processes respect their domain accessibility conditions (Theorem 3.7). The complete formal development of these results relies on a *domain-aware* labeled transition system (Appendix A.2). Its definition is a straightforward generalization of the early labelled transition system for the session π-calculus.

Type Safety and Termination. Following [6], our proof of type preservation relies on a simulation between reductions in the session-typed π-calculus and logical proof reductions.

▶ **Lemma 3.2** (Domain Substitution). *If* $\Omega \vdash \omega_1 \prec \omega_2$ *then*

- *If* $\Omega, \omega_1 \prec \alpha, \Omega'; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ *then* $\Omega, \Omega'\{\omega_2/\alpha\}; \Gamma\{\omega_2/\alpha\}; \Delta\{\omega_2/\alpha\} \vdash P\{\omega_2/\alpha\} :: z{:}A[\omega\{\omega_2/\alpha\}]$.
- $\Omega, \alpha \prec \omega_2, \Omega'; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ *then* $\Omega, \Omega'\{\omega_1/\alpha\}; \Gamma\{\omega_1/\alpha\}; \Delta\{\omega_1/\alpha\} \vdash P\{\omega_1/\alpha\} :: z{:}A[\omega\{\omega_1/\alpha\}]$.

Safe domain communication relies on domain substitution preserving typing (Lemma 3.2).

▶ **Theorem 3.3** (Type Preservation). *If* $\Omega; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ *and* $P \to Q$ *then* $\Omega; \Gamma; \Delta \vdash Q :: z{:}A[\omega]$.

Proof (Sketch). The proof mirrors those of [6, 5, 39], relying on a series of lemmas (Appendix A.4) relating the result of dual process actions (via our LTS semantics) with typable parallel compositions through the (cut) rule. For session type constructors of [6], the results are unchanged. For the domain-aware session type constructors, the development is identical that of [5] and [39], which deal with communication of types and data terms, respectively. ◀

The proof of global progress also follows the lines of [6]: it relies on a notion of a *live* process, which intuitively consists of a process that has not yet fully carried out its ascribed session behavior, and thus is a parallel composition of processes where at least one is a non-replicated process, guarded by some action. Formally, we define $live(P)$ if and only if $P \equiv (\boldsymbol{\nu}\tilde{n})(\pi.Q \mid R)$, for some R, names $\tilde{n}$ and a non-replicated guarded process $\pi.Q$.

▶ **Theorem 3.4** (Global Progress). *If* $\Omega;\cdot;\cdot \vdash P :: x{:}\mathbf{1}[\omega]$ *and* $live(P)$ *then* $\exists Q$ *s.t.* $P \to Q$.

Note that Theorem 3.4 is without loss of generality since using the cut rules we can compose arbitrary well-typed processes together and x need not occur in P due to Rule (**1**R).

Termination (strong normalization) is a relevant property for interactive systems: while from a global perspective they are meant to run forever, at a local level participants should always react within a finite amount of time, and never engage into infinite internal behavior. We say that a process P *terminates*, noted $P \Downarrow$, if there is no infinite reduction path from P.

▶ **Theorem 3.5** (Termination). *If* $\Omega;\Gamma;\Delta \vdash P :: x{:}A[\omega]$ *then* $P \Downarrow$.

Proof (Sketch). By adapting the *linear* logical relations given in [34, 5]. For the system in § 3 without quantifiers, the logical relations correspond to those in [34], extended to carry over Ω. When considering quantifiers, the logical relations resemble those proposed for polymorphic session types in [5], noting that no impredicativity concerns are involved. ◀

Domain Preservation. As a consequence of the hybrid nature of our system, well-typed processes are guaranteed not only to faithfully perform their prescribed behavior in a deadlock-free manner, but they also do so without breaking the constraints put in place on domain reachability given by our well-formedness constraint on sequents.

▶ **Theorem 3.6.** *Let* $\mathcal{E}$ *be a derivation of* $\Omega;\Gamma;\Delta \vdash P :: z{:}A[\omega]$. *If* $\Omega;\Gamma;\Delta \vdash P :: z{:}A[\omega]$ *is well-formed then every sub-derivation in* $\mathcal{E}$ *well-formed.*

While unreachable domains can appear in Γ, such channels can never be used and thus can not appear in a well-typed process due to the restriction on the (copy) rule. Combining Theorems 3.3 and 3.6 we can then show:

▶ **Theorem 3.7.** *Let (1)* $\Omega;\Gamma;\Delta,\Delta' \vdash (\boldsymbol{\nu}x)(P \mid Q) :: z : A[\omega]$, *(2)* $\Omega;\Gamma;\Delta \vdash P :: x{:}B[\omega'']$, *and (3)* $\Omega;\Gamma;\Delta', x{:}B[\omega'] \vdash Q :: z{:}A[\omega]$. *If* $(\boldsymbol{\nu}x)(P \mid Q) \to (\boldsymbol{\nu}x)(P' \mid Q')$ *then: (a)* $\Omega;\Gamma;\Delta \vdash P' :: x'{:}B'[\omega'']$, *for some* x', B', ω''; *(b)* $\Omega;\Gamma,\Delta', x'{:}B'[\omega''] \vdash Q' :: z{:}A[\omega]$; *(c)* $\omega \prec^* \omega''$.

Theorem 3.7 characterizes reachability wrt reduction, showing that even if an ambient session changes domains, typing ensures that such a domain will be (transitively) accessible.

4 Domain-Aware Multiparty Session Types

We now shift our attention to multiparty session types [25]. We consider the standard ingredients: *global types*, *local types*, and the *projection function* that connects the two. Our global types include a new domain-aware construct, $\mathsf{p}\ \mathsf{moves}\ \widetilde{q}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$; our local types exploit the hybrid session types from Def. 3.1. Rather than defining a separate type system based on local types for the process model of § 2, our analysis of multiparty protocols extends the approach defined in [4], which uses *medium processes* to characterize correct multiparty implementations. The advantages are twofold: on the one hand, medium processes provide a precise semantics for global types; on the other hand, they enable the principled transfer of the correctness properties established in § 3 for binary sessions (type preservation, global

progress, termination, domain preservation) to the multiparty setting. Below, *participants* are ranged over by $\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots$; we write $\widetilde{\mathtt{q}}$ to denote a finite set of participants $\mathtt{q}_1, \ldots, \mathtt{q}_n$.

Besides the new domain-aware global type, our syntax of global types includes constructs from [25, 16]. We consider value passing in branching (cf. U below), fully supporting delegation. To streamline the presentation, we consider global types without recursion.

▶ **Definition 4.1** (Global and Local Types). *Define global types (G) and local types (T) as*

$$\begin{array}{lll} U & ::= & \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{str} \mid \ \ldots \mid T \\ G & ::= & \mathsf{end} \mid \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I} \mid \mathtt{p}\ \mathsf{moves}\ \widetilde{q}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2 \\ T & ::= & \mathsf{end} \mid \mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \forall\alpha.T \mid \exists\alpha.T \mid @_\alpha T \mid \downarrow\alpha.T \end{array}$$

The completed global type is denoted end. 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 $\mathtt{p}\ \mathsf{moves}\ \widetilde{\mathtt{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$ specifies the migration of participants $\mathtt{p}, \widetilde{\mathtt{q}}$ to domain ω in order to perform the *sub-protocol* G_1; this migration is lead by $\mathtt{p}$. Subsequently, all of $\mathtt{p}, \widetilde{\mathtt{q}}$ migrate from ω back to their original domains and protocol G_2 is executed. This intuition will be made precise by the medium processes for global types (cf. Def. 4.8). Notice that G_1 and G_2 may involve different sets of participants. In writing $\mathtt{p}\ \mathsf{moves}\ \widetilde{\mathtt{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$ we assume two natural conditions: (a) all migrating participants intervene in the sub-protocol (i.e., the set of participants of G_1 is exactly $\mathtt{p}, \widetilde{\mathtt{q}}$) and (b) domain ω is accessible (in the sense of $\prec$) by all these migrating participants in G_1.

▶ **Definition 4.2.** *The set of participants of G (denoted $\mathsf{part}(G)$) is defined as:* $\mathsf{part}(\mathsf{end}) = \emptyset$, $\mathsf{part}(\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}) = \{\mathtt{p}, \mathtt{q}\} \cup \bigcup_{i\in I} \mathsf{part}(G_i)$, $\mathsf{part}(\mathtt{p}\ \mathsf{moves}\ \widetilde{\mathtt{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2) = \{\mathtt{p}\} \cup \widetilde{\mathtt{q}} \cup \mathsf{part}(G_1) \cup \mathsf{part}(G_2)$. *We sometimes write $\mathtt{p} \in G$ to mean $\mathtt{p} \in \mathsf{part}(G)$.*

Global types are projected onto participants so as to obtain local types. The terminated local type is end. 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. Exploiting the domain-aware framework in § 3, we introduce four new local types. They increase the expressiveness of standard local types by specifying universal and existential quantification over domains ($\forall\alpha.T$ and $\exists\alpha.T$), migration to a specific domain ($@_\alpha T$), and a reference to the current domain ($\downarrow\alpha.T$, with α occurring in T).

We now define *(merge-based) projection* for global types [16]. To this end, we rely on a *merge* operator on local types, which in our case considers messages U.

▶ **Definition 4.3** (Merge). *We define $\sqcup$ as the commutative partial operator on base and local types such that* $\mathsf{bool} \sqcup \mathsf{bool} = \mathsf{bool}$ *(and analogously for other base types), and*

1. $T \sqcup T = T$, *where T is one of the following:* end, $\mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$, $@_\omega T$, $\forall\alpha.T$, *or* $\exists\alpha.T$;
2. $\mathtt{p}?\{l_k\langle U_k\rangle.T_k\}_{k\in K} \sqcup \mathtt{p}?\{l'_j\langle U'_j\rangle.T'_j\}_{j\in J} =$
   $\mathtt{p}?\big(\{l_k\langle U_k\rangle.T_k\}_{k\in K\setminus J} \cup \{l'_j\langle U'_j\rangle.T'_j\}_{j\in J\setminus K} \cup \{l_l\langle U_l \sqcup U'_l\rangle.(T_l \sqcup T'_l)\}_{l\in K\cap J}\big)$

and is undefined otherwise.

Therefore, for $U_1 \sqcup U_2$ to be defined there are two options: (a) U_1 and U_2 are identical base, terminated, selection, or "hybrid" local 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.

To define projection and medium processes for the new global type $\mathtt{p}\ \mathsf{moves}\ \widetilde{\mathtt{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$, we require ways of "fusing" local types and processes. The intent is to capture in a single (sequential) specification the behavior of two distinct (sequential) specifications, i.e., those corresponding to protocols G_1 and G_2. For local types, we have the following definition:

▶ **Definition 4.4** (Local Type Fusion). *The* fusion *of T_1 and T_2, written $T_1 \circ T_2$, is given by:*

$$
\begin{array}{lcllcl}
\mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \circ T & = & \mathtt{p}!\{l_i\langle U_i\rangle.(T_i \circ T)\}_{i\in I} & \mathsf{end} \circ T & = & T \\
\mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \circ T & = & \mathtt{p}?\{l_i\langle U_i\rangle.(T_i \circ T)\}_{i\in I} & (\exists\alpha.T_1) \circ T & = & \exists\alpha.(T_1 \circ T) \\
(\forall\alpha.T_1) \circ T & = & \forall\alpha.(T_1 \circ T) & (@_\alpha T_1) \circ T & = & @_\alpha(T_1 \circ T) \\
(\downarrow\alpha.T_1) \circ T & = & \downarrow\alpha.(T_1 \circ T) & & &
\end{array}
$$

This way, e.g., if $T_1 = \exists\alpha.@_\alpha\, \mathtt{p}?\{l_1\langle\mathsf{Int}\rangle.\mathsf{end}\,, l_2\langle\mathsf{Bool}\rangle.\mathsf{end}\}$ and $T_2 = @_\omega\, \mathtt{q}!\{l\langle\mathsf{Str}\rangle.\mathsf{end}\}$, then $T_1 \circ T_2 = \exists\alpha.@_\alpha\, \mathtt{p}?\{l_1\langle\mathsf{Int}\rangle.@_\omega\, \mathtt{q}!\{l\langle\mathsf{Str}\rangle.\mathsf{end}\}\,, l_2\langle\mathsf{Bool}\rangle.@_\omega\, \mathtt{q}!\{l\langle\mathsf{Str}\rangle.\mathsf{end}\}\}$. We can now define:

▶ **Definition 4.5** (Merge-based Projection [16]). *Let G be a global type. The* merge-based projection *of G under participant $\mathtt{r}$, denoted $G\upharpoonright\mathtt{r}$, is defined as $\mathsf{end}\upharpoonright\mathtt{r} = \mathsf{end}$ and*

- $$\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}\upharpoonright\mathtt{r} = \begin{cases} \mathtt{p}!\{l_i\langle U_i\rangle.G_i\upharpoonright\mathtt{r}\}_{i\in I} & \text{if } \mathtt{r} = \mathtt{p} \\ \mathtt{p}?\{l_i\langle U_i\rangle.G_i\upharpoonright\mathtt{r}\}_{i\in I} & \text{if } \mathtt{r} = \mathtt{q} \\ \sqcup_{i\in I}\, G_i\upharpoonright\mathtt{r} & \text{otherwise } (\sqcup \text{ as in Def. 4.3}) \end{cases}$$
- $$(\mathtt{p}\ \mathsf{moves}\ \widetilde{\mathtt{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\,;\, G_2)\upharpoonright\mathtt{r} = \begin{cases} \downarrow\beta.(\exists\alpha.@_\alpha\, G_1\upharpoonright\mathtt{r}) \circ @_\beta\, G_2\upharpoonright\mathtt{r} & \text{if } \mathtt{r} = \mathtt{p} \\ \downarrow\beta.(\forall\alpha.@_\alpha\, G_1\upharpoonright\mathtt{r}) \circ @_\beta\, G_2\upharpoonright\mathtt{r} & \text{if } \mathtt{r} \in \widetilde{\mathtt{q}} \\ G_2\upharpoonright\mathtt{r} & \text{otherwise} \end{cases}$$

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

The projection for the type $\mathtt{p}\ \mathsf{moves}\ \widetilde{\mathtt{q}}\ \mathsf{to}\ w\ \mathsf{for}\ G_1\,;\, G_2$ is one of the key points in our analysis. The local type for $\mathtt{p}$, the leader of the migration, starts by binding the identity of its current domain (say, $\omega_\mathtt{p}$) to β. Then, the (fresh) domain ω is communicated, and there is a migration step to ω, which is where protocol $G_1\upharpoonright\mathtt{p}$ will be performed. Finally, there is a migration step from ω back to $\omega_\mathtt{p}$; once there, the protocol $G_2\upharpoonright\mathtt{p}$ will be performed. The local type for all of $\mathtt{q}_i \in \widetilde{\mathtt{q}}$ follows accordingly: they expect ω from $\mathtt{p}$; the migration from their original domains to ω (and back) is as for $\mathtt{p}$. For participants in G_1, the fusion on local types (Def. 4.4) defines a local type that includes the actions for G_1 but also for G_2, if any: a participant in G_1 need not be involved in G_2. Interestingly, the resulting local types $\downarrow\beta.(\exists\alpha.@_\alpha\, G_1\upharpoonright\mathtt{p}) \circ @_\beta\, G_2\upharpoonright\mathtt{p}$ and $\downarrow\beta.(\forall\alpha.@_\alpha\, G_1\upharpoonright\mathtt{q}_i) \circ @_\beta\, G_2\upharpoonright\mathtt{q}_i$ define a precise combination of hybrid connectives whereby each migration step is bound by a quantifier or the current domain.

The following notion of *well-formedness* for global types is standard:

▶ **Definition 4.6** (Well-Formed Global Types [25]). *We say that global type G is* well-formed (WF, in the following) *if the projection $G\upharpoonright\mathtt{r}$ is defined for all $\mathtt{r} \in G$.*

Analyzing Global Types via Medium Processes A *medium process* is a well-typed process from § 2 that captures the communication behavior of the domain-aware global types of Def. 4.1. Here we define medium processes and establish two fundamental characterization results for them (Theorems 4.11 and 4.12). We shall consider names *indexed by participants*: given a name c and a participant $\mathtt{p}$, we use $c_\mathtt{p}$ to denote the name along which the session behavior of $\mathtt{p}$ will be made available. This way, if $\mathtt{p} \neq \mathtt{q}$ then $c_\mathtt{p} \neq c_\mathtt{q}$.

To define mediums, we need to fuse sequential processes just as Def. 4.4 fuses local types:

▶ **Definition 4.7** (Fusion of Processes). *We define $\circ$ as the partial operator on well-typed processes such that (with $\pi \in \{c(y), c\langle\omega\rangle, c(\alpha), c\langle y@\omega\rangle, c(y@\omega), c \triangleleft l\}$) :*

$$
\begin{array}{lcllcl}
c\langle y\rangle.([u\leftrightarrow y] \mid P) \circ Q & \triangleq & c\langle y\rangle.([u\leftrightarrow y] \mid (P \circ Q)) & \mathbf{0} \circ Q & \triangleq & Q \\
c \triangleright \{l_i : P_i\}_{i\in I} \circ Q & \triangleq & c \triangleright \{l_i : (P_i \circ Q)\}_{i\in I} & (\pi.P) \circ Q & \triangleq & \pi.(P \circ Q)
\end{array}
$$

and is undefined otherwise.

The previous definition suffices for the definition of medium processes, given next. Using indexed names, a medium process uniformly captures the behavior of a global type:

▶ **Definition 4.8** (Medium Process). *Let G be a global type (cf. Def. 4.1), $\tilde{c}$ be a set of indexed names, and $\tilde{\omega}$ a set of domains. The* medium process *of G (or simply* medium*), denoted $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, is defined as follows:*

$$\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) = \begin{cases} \mathbf{0} & \text{if } G = \mathtt{end} \\ c_{\mathtt{p}} \triangleright \left\{ l_i : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_i; \overline{c_{\mathtt{q}}}\langle v\rangle.([u \leftrightarrow v] \mid \mathsf{M}^{\tilde{\omega}}[\![G_i]\!](\tilde{c})) \right\}_{i\in I} & \text{if } G = \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I} \\ c_{\mathtt{p}}(\alpha).c_{\mathtt{q}_1}\langle\alpha\rangle.\cdots.c_{\mathtt{q}_n}\langle\alpha\rangle. & \text{if } G = \mathtt{p}\ \mathsf{moves}\ \mathtt{q}_1,\ldots,\mathtt{q}_n\ \mathsf{to}\ w\ \mathsf{for}\ G_1\,;\,G_2 \\ \quad c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}_1}(y_{\mathtt{q}_1}@\alpha).\cdots.c_{\mathtt{q}_n}(y_{\mathtt{q}_n}@\alpha). & \\ \quad\quad \mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\ldots,\alpha/\omega_{\mathtt{q}_n}\}}[\![G_1]\!](\tilde{y}) \circ & \\ \quad\quad\quad (y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}_1}(m_{\mathtt{q}_1}@\omega_{\mathtt{q}_1}).\cdots.y_{\mathtt{q}_n}(m_{\mathtt{q}_n}@\omega_{\mathtt{q}_n}). & \\ \quad\quad\quad\quad \mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})) & \end{cases}$$

where $\mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{c}) \circ \mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{c})$ is as in Def. 4.7.

The medium for $G = \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}$ exploits four prefixes to mediate in the interaction between the implementations of $\mathtt{p}$ and $\mathtt{q}$: the first two prefixes (on name $c_{\mathtt{p}}$) capture the label selected by $\mathtt{p}$ and the subsequently received value; the third and fourth prefixes (on name $c_{\mathtt{q}}$) propagate the choice and forward the value sent by $\mathtt{p}$ to $\mathtt{q}$.

The medium for $G = \mathtt{p}\ \mathsf{moves}\ \mathtt{q}_1,\ldots,\mathtt{q}_n\ \mathsf{to}\ w\ \mathsf{for}\ G_1\,;\,G_2$ showcases the expressivity and convenience of our domain-aware process framework. In this case, the medium's behavior takes place through the following steps: First, $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ inputs a domain identifier (say, ω) from $\mathtt{p}$ which is forwarded to $\mathtt{q}_1,\ldots,\mathtt{q}_n$, the other participants of G_1. Secondly, the roles $\mathtt{p},\mathtt{q}_1,\ldots,\mathtt{q}_n$ migrate from their domains $\omega_{\mathtt{p}},\omega_{\mathtt{q}_1}\ldots,\omega_{\mathtt{q}_n}$ to ω. At this point, the medium for G_1 can execute, keeping track the current domain ω for all participants. Finally, the participants of G_1 migrate back to their original domains and the medium for G_2 executes.

Recalling the domain-aware global type of § 1, we produce its medium process:

$$\begin{array}{l} c_{\mathtt{cl}} \triangleright \left\{ \mathit{request} : c_{\mathtt{cl}}(r).c_{\mathtt{mw}} \triangleleft \mathit{request}; \overline{c_{\mathtt{mw}}}\langle v\rangle.([r \leftrightarrow v] \mid \right. \\ c_{\mathtt{mw}} \triangleright \left\{ \mathit{reply} : c_{\mathtt{mw}}(a).c_{\mathtt{cl}} \triangleleft \mathit{reply}; \overline{c_{\mathtt{cl}}}\langle n\rangle.([a \leftrightarrow n] \mid c_{\mathtt{mw}} \triangleright \left\{ \mathit{done} : c_{\mathtt{serv}} \triangleleft \mathit{done}; \mathbf{0} \right\}), \right. \\ \quad \mathit{wait} : c_{\mathtt{mw}} \triangleright \left\{ \mathit{init} : c_{\mathtt{serv}} \triangleleft \mathit{init}; c_{\mathtt{mw}}(w_{\mathtt{priv}}).c_{\mathtt{serv}}\langle w_{\mathtt{priv}}\rangle. \right. \\ \quad\quad c_{\mathtt{mw}}(y_{\mathtt{mw}}@w_{\mathtt{priv}}).c_{\mathtt{serv}}(y_{\mathtt{serv}}@w_{\mathtt{priv}}).\mathsf{M}^{\tilde{w_{\mathtt{priv}}}}[\![\mathit{Offload}]\!](y_{\mathtt{mw}}, y_{\mathtt{serv}}) \circ \\ \quad\quad (y_{\mathtt{mw}}(z_{\mathtt{mw}}@w_{\mathtt{mw}}).y_{\mathtt{serv}}(z_{\mathtt{serv}}@w_{\mathtt{serv}}). \\ \quad\quad z_{\mathtt{mw}} \triangleright \left\{ \mathit{reply} : z_{\mathtt{mw}}(a).c_{\mathtt{cl}} \triangleleft \mathit{reply}; \overline{c_{\mathtt{cl}}}\langle n\rangle.([a \leftrightarrow n] \mid \mathbf{0}) \right\}) \left. \left. \left. \right\} \right\} \right) \left. \right\} \end{array}$$

The medium ensures the client's domain remains fixed through the entire interaction, regardless of whether the middleware chooses to interact with the server. This showcases how our medium transparently manages domain migration of participants.

Characterization Results We state results that offer a sound and complete account of the relationship between: (i) a global type G (and its local types), (ii) its medium process $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, and (iii) process implementations for the participants $\{\mathtt{p}_1,\ldots,\mathtt{p}_n\}$ of G. In a nutshell, these results say that the typeful composition of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ with processes for each $\mathtt{p}_1,\ldots,\mathtt{p}_n$ (well-typed in the system of § 3) performs the intended global type. Crucially, these processes reside in distinct domains and can be independently developed, guided by their local type—they need not know about the medium's existence or structure. The results generalize those in [4] to the domain-aware setting. Given a global type G with $\mathsf{part}(G) = \{\mathtt{p}_1,\ldots,\mathtt{p}_n\}$, below we write $\mathsf{npart}(G)$ to denote the set of indexed names $\{c_{\mathtt{p}_1},\ldots,c_{\mathtt{p}_n}\}$. We define:

▶ **Definition 4.9** (Compositional Typing). *We say $\Omega;\Gamma;\Delta \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z{:}C$ is a* compositional typing *if: (i) it is a valid typing derivation; (ii) $\mathsf{npart}(G) \subseteq dom(\Delta)$; and (iii) $C = \mathbf{1}$.*

A compositional typing says that $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ depends on behaviors associated to each participant of G; it also specifies that $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ does not offer any behaviors of its own.

The following definition relates binary session types and local types: the main difference is that the former do not mention participants. Below, B ranges over base types ($\mathsf{bool}, \mathsf{nat}, \ldots$).

▶ **Definition 4.10** (Local Types→Binary Types). *Mapping $\langle\!\langle \cdot \rangle\!\rangle$ from local types T (Def. 4.1) into binary types A (Def. 3.1) is inductively defined as $\langle\!\langle \mathsf{end} \rangle\!\rangle = \langle\!\langle B \rangle\!\rangle = \mathbf{1}$ and*

$$\begin{array}{lcllcl} \langle\!\langle \mathsf{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \rangle\!\rangle & = & \oplus\{l_i : \langle\!\langle U_i \rangle\!\rangle \otimes \langle\!\langle T_i \rangle\!\rangle\}_{i\in I} & \langle\!\langle \forall\alpha.T \rangle\!\rangle & = & \forall\alpha.\langle\!\langle T \rangle\!\rangle \\ \langle\!\langle \mathsf{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \rangle\!\rangle & = & \&\{l_i : \langle\!\langle U_i \rangle\!\rangle \multimap \langle\!\langle T_i \rangle\!\rangle\}_{i\in I} & \langle\!\langle \exists\alpha.T \rangle\!\rangle & = & \exists\alpha.\langle\!\langle T \rangle\!\rangle \\ \langle\!\langle @_\omega T \rangle\!\rangle & = & @_\omega\langle\!\langle T \rangle\!\rangle & \langle\!\langle \downarrow\alpha.T \rangle\!\rangle & = & \downarrow\alpha.\langle\!\langle T \rangle\!\rangle \end{array}$$

Our first characterization result ensures that well-formedness of a global type G guarantees the typability of its medium $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ using binary session types. Hence, it ensures that multiparty protocols can be analyzed by composing the medium with independently obtained, well-typed implementations for each protocol participant. Crucially, the resulting well-typed process will inherit all correctness properties ensured by binary typability established in § 3.

▶ **Theorem 4.11** (Global Types → Typed Mediums). *If G is WF with $\mathsf{part}(G) = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ then $\Omega; \Gamma; c_{\mathsf{p}_1} : \langle\!\langle G \upharpoonright \mathsf{p}_1 \rangle\!\rangle[\omega_1], \ldots, c_{\mathsf{p}_n} : \langle\!\langle G \upharpoonright \mathsf{p}_n \rangle\!\rangle[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ is a compositional typing, for some Ω, Γ, with $\tilde{\omega} = \omega_1, \ldots, \omega_n$. We assume that $\omega_i \prec \omega_m$ for all $i \in \{1, \ldots, n\}$ (the medium's domain is accessible by all), and that $i \neq j$ implies $\omega_i \neq \omega_j$.*

Proof. By induction on the structure of G; see Appendix A.6. ◀

The second characterization result, given next, is the converse of Theorem 4.11: binary typability precisely delineates the session interactions that underlie well-formed multiparty protocols. We need an auxiliary definition to relate local types based on their branching operators and the "here" operators that govern migration. First, we wish to relate T_1 and T_2 if there exists a T' such that $T_1 \sqcup T' = T_2$ (cf. Def. 4.3). Second, we wish to relate T_1 and T_2 if (i) $T_1 = T'$ and $T_2 = \downarrow\alpha.T'$ and α does not occur in T'; but also if (ii) $T_1 = \downarrow\alpha.T'$ and $T_2 = T'\{\omega/\alpha\}$. Below, this relation on local types is denoted $T_1 \preceq^{\sqcup}_{\downarrow} T_2$ (cf. Appendix A.5).

▶ **Theorem 4.12** (Well-Typed Mediums → Global Types). *Let G be a global type (cf. Def. 4.1). If $\Omega; \Gamma; c_{\mathsf{p}_1} : A_1[\omega_1], \ldots, c_{\mathsf{p}_n} : A_n[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ is a compositional typing then $\exists T_1, \ldots, T_n$ such that $G \upharpoonright \mathsf{p}_j \preceq^{\sqcup}_{\downarrow} T_j$ and $\langle\!\langle T_j \rangle\!\rangle = A_j$, for all $\mathsf{p}_j \in \mathsf{part}(G)$.*

Proof. By induction on the structure of G, using $\preceq^{\sqcup}_{\downarrow}$ as in Def. A.7; see Appendix A.6. ◀

The above theorems offer a *static guarantee* that connects multiparty protocols and well-typed processes. They can be used to establish also *dynamic guarantees* relating the behavior of a global type G and that of its associated set of *multiparty systems* (i.e., the typeful composition of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ with processes for each of $\mathsf{p}_i \in \mathsf{part}(G)$). These dynamic guarantees can be easily obtained by combining Theorems 4.11 and 4.12 with the approach in [4].

5 Related Work

Our work contributes to the logical foundations of concurrency, a line of research developed by Caires and Pfenning [6], Dal Lago and Di Giamberardino [29], Wadler [43], and others. Medium-based analyzes of multiparty sessions in a logical setting were developed in [4] and later used in an account of classical multiparty sessions in an extended linear logic [10].

Several process calculi with distributed features have been put forward. Two salient proposals are the Ambient calculus [12], in which processes may move across *ambients*—abstractions of administrative domains, and the *distributed π-calculus* (Dpi) [22], which

extends the π-calculus with flat locations, local communication, and process migration. While domains in our model may be read as locations, their abstract and parametric nature admits various alternative readings (e.g. administrative domains, security-related levels), leveraging the partial view of the domain hierarchy.

Concerning type systems for such calculi, e.g., those in [11, 3], typing is used to specify security and communication-oriented properties in terms of ambient movement. Their work does not cover issues of structured interaction, which is central in our work. Garralda et al. [20] integrate binary sessions in an Ambient calculus, ensuring that session communication is undisturbed by ambient mobility steps. This contrasts with our work, where typing ensures that both migrations and communication are safely described by (extended) session types.

Demangeon and Honda [14] study multiparty sessions with nested protocols. Their nesting construct is similar to our global migration construct, which also introduces nesting. Their work focuses on modularity in choreographic programming and is not concerned with domains nor domain migration. As such, their nested protocols can have *local* participants and may be parameterized on data from previous actions. We conjecture that our medium-based approach can accommodate local participants in a similar way. Data parameterization can be transposed to our logical setting via dependent session types [39, 42]. Asynchrony and recursive behaviors can also be integrated by exploiting existing logical foundations [18, 41].

Balzer et al. [1] overlay a notion of world and accessibility on a system of *shared* session types to ensure deadlock-freedom. Their work differs substantially from ours, being closer to partial-order-based typing for deadlock-freedom [28, 33]: they instantiate accessibility as a partial-order, equip sessions with multiple worlds and are not conservative wrt linear logic.

We highlight the works [21, 26, 15] that study runtime monitoring of contracts in session-based systems. Mediums can be seen as monitors that enforce the communication and domain migrations specified in domain-aware multiparty sessions. The study of contract enforcing mediums (e.g. enforcing refinements or limited trust) is interesting future work.

Finally, extensions of session types with access control and information flow analyses have been proposed by Capecchi et al. [9, 8]. Our enforcement of communication across connected domains introduces some high-level similarities with information flow analyses. Establishing the precise relationship with such works is an interesting item of future work.

6 Concluding Remarks

We have proposed a Curry-Howard interpretation of hybrid linear logic as domain-aware session types. Our development generalizes the interpretation put forward in [6], leading to a typing discipline with enhanced expressiveness and strong correctness properties such as global progress, termination, and session fidelity for well-typed domain-aware processes. Domain-awareness is realized at both the process and type-level, accounting for scenarios where domain information can be only determined at runtime. We also leverage a *parametric* domain accessibility relation for added flexibility. Our system statically rules out processes that communicate with unreachable domains, which is beyond the scope of previous works.

As an application of our framework, we presented the first systematic study of domain-awareness in a *multiparty* setting, considering multiparty sessions with domain-aware migration and communication whose semantics is given by a typed (binary) medium process that orchestrates the multiparty protocol. Embedded in a fully distributed domain structure, our medium is shown to strongly encode domain-aware multiparty sessions; it naturally allows us to transpose the correctness properties of our logical development to the multiparty setting.

References

1 Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, pages 611–639, 2019.

2 Torben Braüner and Valeria de Paiva. Intuitionistic hybrid logic. *J. of App. Log.*, 4:231–255, 2006.

3 Michele Bugliesi and Giuseppe Castagna. Behavioural typing for safe ambients. *Comput. Lang.*, 28(1):61–99, 2002.

4 Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, pages 74–95, 2016. Extended version at `https://sites.google.com/a/jorgeaperez.net/www/publications/medium16long.pdf`.

5 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *ESOP*, volume 7792 of *LNCS*. Springer, 2013.

6 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.

7 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. URL: `https://doi.org/10.1017/S0960129514000218`, `doi:10.1017/S0960129514000218`.

8 Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Information flow safety in multiparty sessions. In Bas Luttik and Frank Valencia, editors, *EXPRESS*, volume 64 of *EPTCS*, pages 16–30, 2011.

9 Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session types for access and information flow control. In *CONCUR*, volume 6269 of *LNCS*, pages 237–252. Springer, 2010.

10 Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 33:1–33:15, 2016.

11 Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the ambient calculus. *Inf. Comput.*, 177(2):160–194, 2002.

12 Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

13 Kaustuv Chaudhuri, Carlos Olarte, Elaine Pimentel, and Joëlle Despeyroux. Hybrid Linear Logic, revisited. *Mathematical Structures in Computer Science*, 2019. URL: `https://hal.inria.fr/hal-01968154`.

14 Romain Demangeon and Kohei Honda. Nested protocols in session types. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, pages 272–286, 2012. URL: `https://doi.org/10.1007/978-3-642-32940-1_20`, `doi:10.1007/978-3-642-32940-1\_20`.

15 Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: distributed dynamic verification with multiparty session types and python. *Formal Methods in System Design*, 46(3):197–225, 2015. URL: `https://doi.org/10.1007/s10703-014-0218-8`, `doi:10.1007/s10703-014-0218-8`.

16 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages,*

and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II, volume 7966 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-39212-2_18, doi:10.1007/978-3-642-39212-2_18.

17 Joëlle Despeyroux, Carlos Olarte, and Elaine Pimentel. Hybrid and subexponential linear logics. *Electr. Notes Theor. Comput. Sci.*, 332:95–111, 2017.

18 Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, pages 228–242, 2012.

19 Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview. In *WS-FM 2009*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010.

20 Pablo Garralda, Adriana B. Compagnoni, and Mariangiola Dezani-Ciancaglini. Bass: boxed ambients with safe sessions. In Annalisa Bossi and Michael J. Maher, editors, *PPDP*, pages 61–72. ACM, 2006.

21 Hannah Gommerstadt, Limin Jia, and Frank Pfenning. Session-typed concurrent contracts. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 771–798, 2018.

22 Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002.

23 Kohei Honda. Types for dynamic interaction. In *CONCUR*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.

24 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP'98*, LNCS. Springer, 1998.

25 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008. URL: https://doi.org/10.1145/1328438.1328472, doi:10.1145/1328438.1328472.

26 Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 582–594, 2016.

27 Limin Jia and David Walker. Modal proofs as distributed programs (extended abstract). In *ESOP*, volume 2986 of *LNCS*, pages 219–233. Springer, 2004.

28 Naoki Kobayashi. A new type system for deadlock-free processes. In *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, pages 233–247, 2006.

29 Ugo Dal Lago and Paolo Di Giamberardino. Soft session types. In *EXPRESS*, volume 64 of *EPTCS*, pages 59–73, 2011.

30 Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, part I/II. *Inf. Comput.*, 100(1):1–77, 1992.

31 Tom Murphy. *Modal Types for Mobile Code*. PhD thesis, Carnegie Mellon University, 2008.

32 Tom Murphy, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *LICS*, pages 286–295. IEEE Computer Society, 2004.

33 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 72:1–72:10, 2014.

34 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In *ESOP*, volume 7211 of *LNCS*, pages 539–558. Springer, 2012.

35 Jason Reed. Hybridizing a logical framework. *Electr. Notes Theor. Comput. Sci.*, 174(6):135–148, 2007.

36 Davide Sangiorgi. pi-calculus, internal mobility, and agent-passing calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996.

37 Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

38 Alex Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.

39 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In *Proc. of PPDP '11*, pages 161–172, New York, NY, USA, 2011. ACM. URL: `http://doi.acm.org/10.1145/2003476.2003499`, `doi:http://doi.acm.org/10.1145/2003476.2003499`.

40 Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In Lars Birkedal, editor, *FoSSaCS*, volume 7213 of *LNCS*, pages 346–360. Springer, 2012.

41 Bernardo Toninho, Luís Caires, and Frank Pfenning. Corecursion and non-divergence in session-typed processes. In *Trustworthy Global Computing - 9th International Symposium, TGC 2014, Rome, Italy, September 5-6, 2014. Revised Selected Papers*, pages 159–175, 2014.

42 Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 128–145, 2018.

43 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014.

A Appendix

A.1 Structural Congruence

▶ **Definition A.1.** Structural congruence *($P \equiv Q$) is the least congruence relation on processes such that*

$$P \mid \mathbf{0} \equiv P \qquad P \equiv_\alpha Q \Rightarrow P \equiv Q \qquad (\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0} \qquad [x \leftrightarrow y] \equiv [y \leftrightarrow x] \qquad P \mid Q \equiv Q \mid P$$
$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad x \notin fn(P) \Rightarrow P \mid (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \mid Q)$$
$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$$

A.2 Labeled Transition System

Some technical results rely on labeled transitions rather than on reduction. To characterize the interactions of a well-typed process with its environment, we extend the early labeled transition system (LTS) for the π-calculus [37] with labels and transition rules for choice, migration, and forwarding constructs. A transition $P \xrightarrow{\lambda} Q$ denotes that P may evolve to Q by performing the action represented by label λ. Transition labels are defined below:

$$\lambda \quad ::= \quad \tau \mid x(y) \mid x(w) \mid x.l \mid x.y@\omega \mid \overline{x\,y} \mid \overline{x\langle y\rangle} \mid \overline{x\,w} \mid \overline{x.l} \mid \overline{x.y@\omega}$$

Actions are name input $x(y)$, domain input $x(w)$, the offers $x.\mathsf{inl}$ and $x.\mathsf{inr}$, migration $x.y@\omega$ and their matching co-actions, respectively the output $\overline{x\,y}$ and bound output $\overline{x\langle y\rangle}$ actions, the domain output $\overline{x\,w}$, label selections $\overline{x.l}$ and $\overline{x.l}$, and domain migration $\overline{x.y@\omega}$. Both the bound output $\overline{x\langle y\rangle}$ and migration action $\overline{x.y@\omega}$ denote extrusion of a fresh name y along x. Internal action is denoted by τ. In general, an action requires a matching co-action in the environment to enable progress.

$$\text{(id)}\ (\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \xrightarrow{\tau} P\{y/x\} \qquad \text{(n.out)}\ x\langle y\rangle.P \xrightarrow{\overline{x\ y}} P \qquad \text{(n.in)}\ x(y).P \xrightarrow{x(z)} P\{z/y\}$$

$$\text{(d.out)}\ x\langle w\rangle.P \xrightarrow{\overline{x\ w}} P \qquad \text{(d.in)}\ x(\alpha).P \xrightarrow{x(w)} P\{w/\alpha\}$$

$$\text{(move)}\ x\langle y@w\rangle.P \xrightarrow{\overline{x.y@\omega}} (\boldsymbol{\nu} y)P \qquad \text{(move}')\ x(z@w).P \xrightarrow{x.y@\omega} P\{y/z\}$$

$$\text{(par)}\ \frac{P \xrightarrow{\lambda} Q}{P \mid R \xrightarrow{\lambda} Q \mid R} \qquad \text{(com)}\ \frac{P \xrightarrow{\overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{(res)}\ \frac{P \xrightarrow{\lambda} Q}{(\boldsymbol{\nu} y)P \xrightarrow{\lambda} (\boldsymbol{\nu} y)Q} \qquad \text{(open)}\ \frac{P \xrightarrow{\overline{x\ y}} Q}{(\boldsymbol{\nu} y)P \xrightarrow{\overline{x\langle y\rangle}} Q}$$

$$\text{(close)}\ \frac{P \xrightarrow{\overline{x\langle y\rangle}} P' \quad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} y)(P' \mid Q')} \qquad \text{(rep)}\ !x(y).P \xrightarrow{x(z)} P\{z/y\} \mid !x(y).P$$

$$\text{(l.out)}\ x \triangleleft l_i; P \xrightarrow{\overline{x.l_i}} P \qquad \text{(l.in)}\ x \triangleright \{l_i : P_i\}_{i \in I} \xrightarrow{x.l_i} P_i$$

Figure 2 Labeled Transition System.

▶ **Definition A.2** (Labeled Transition System). *The relation* labeled transition *($P \xrightarrow{\lambda} Q$) is defined by the rules in Fig. 2, subject to the side conditions: in rule* (res)*, we require $y \notin fn(\lambda)$; in rule* (par)*, we require $bn(\lambda) \cap fn(R) = \emptyset$; in rule* (close)*, we require $y \notin fn(Q)$. We omit the symmetric versions of rules* (par)*,* (com)*, and* (close)*.*

We write $subj(\lambda)$ for the subject of the action λ, that is, the channel along which the action takes place. Weak transitions are defined as usual. Let us write $\rho_1\rho_2$ for the composition of relations ρ_1, ρ_2 and $\Longrightarrow$ for the reflexive, transitive closure of $\xrightarrow{\tau}$. Notation $\overset{\lambda}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\lambda}\Longrightarrow$ (given $\lambda \neq \tau$) and $\overset{\tau}{\Longrightarrow}$ stands for $\Longrightarrow$. We recall basic facts about reduction, structural congruence, and labeled transition: closure of labeled transitions under structural congruence, and coincidence of τ-labeled transition and reduction [37]: (1) if $P \equiv \xrightarrow{\lambda} Q$ then $P \xrightarrow{\lambda} \equiv Q$; (2) $P \to Q$ iff $P \xrightarrow{\tau} \equiv Q$.

A.3 Omitted Typing Rules

$$(\&\mathsf{R})\ \frac{\Omega;\Gamma;\Delta \vdash P_1 :: x{:}A_1[\omega] \quad \ldots \quad \Omega;\Gamma;\Delta \vdash P_n :: x{:}A_n[\omega]}{\Omega;\Gamma;\Delta \vdash x \triangleright \{l_i : P_i\}_{i\in I} :: z{:}\&\{l_i : A_i\}_{i\in I}[\omega]}$$

$$(\&\mathsf{L}_1)\ \frac{\Gamma;\Delta, x{:}A[\omega_2] \vdash P :: z{:}C[\omega_1]}{\Gamma;\Delta, x{:}\,\&\{l_i : A\}_{\{i\}}[\omega_2] \vdash x \triangleleft l_i; P :: z{:}C[\omega_1]}$$

$$(\&\mathsf{L}_2)\ \frac{\Gamma;\Delta, x{:}\,\&\{l_i{:}A_i\}_{i\in I}[\omega_2] \vdash P :: z{:}C[\omega_1] \quad k \notin I}{\Gamma;\Delta, x{:}\,\&\{l_j{:}A_j\}_{j\in I\cup\{k\}}[\omega_2] \vdash P :: z{:}C[\omega_1]}$$

$$(\oplus\mathsf{R}_1)\ \frac{\Gamma;\Delta \vdash P :: x{:}A[\omega]}{\Gamma;\Delta \vdash x \triangleleft l_i; P :: x{:}\,\oplus\{l_i : A\}_{\{i\}}[\omega]} \qquad (\oplus\mathsf{R}_2)\ \frac{\Gamma;\Delta \vdash P :: x{:}\,\oplus\{l_i : A_i\}_{i\in I}[\omega] \quad k \notin I}{\Gamma;\Delta \vdash P :: x{:}\,\oplus\{l_j : A_j\}_{j\in I\cup\{k\}}[\omega]}$$

$$(\oplus\mathsf{L})\ \frac{\Omega;\Gamma;\Delta, x{:}A_1[\omega_2] \vdash Q_1 :: z{:}C[\omega_1] \quad \ldots \quad \Omega;\Gamma;\Delta, x{:}A_n[\omega_2] \vdash Q_n :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta, x{:}\oplus\{l_i : A_i\}_{i\in I}[\omega_2] \vdash x \triangleright \{l_i : Q_i\}_{i\in I} :: z{:}C[\omega_1]}$$

$$(!\mathsf{L})\ \frac{\Omega;\Gamma, u{:}A[\omega_2];\Delta \vdash P :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta, x{:}!A[\omega_2] \vdash x(u).P :: z{:}C[\omega_1]} \qquad (!\mathsf{R})\ \frac{\Omega;\Gamma;\cdot \vdash Q :: y{:}A[\omega]}{\Omega;\Gamma;\cdot \vdash \overline{x}\langle u\rangle.!u(y).Q :: x{:}!A[\omega]}$$

$$(\mathsf{cut}^!)\ \frac{\Omega;\Gamma;\cdot \vdash P :: x{:}A[\omega_1] \quad \Omega;\Gamma, u{:}A[\omega_1];\Delta \vdash Q :: z{:}C[\omega_2]}{\Omega;\Gamma;\Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid Q) :: z{:}C[\omega_2]}$$

A.4 Additional Lemmas for Type Preservation

The development of type preservation extends that of [6] to account for domain communication and migration. The proof mainly relies on a series of reduction lemmas (one per session type connective that produces observable process actions) that relate process actions with parallel composition through the (cut) rule, which correspond to logical proof reductions. For instance, the reduction lemma for $\otimes$ is:

▶ **Lemma A.3** (Reduction Lemma – $\otimes$). *Assume*
(a) $\Omega;\Gamma;\Delta_1 \vdash P :: x{:}A_1 \otimes A_2[\omega]$ *with* $P \xrightarrow{\overline{(\nu y)x\langle y\rangle}} P'$*; and*
(b) $\Gamma;\Delta_2, x{:}A_1 \otimes A_2[\omega] \vdash Q :: z{:}C[\omega']$ *with* $Q \xrightarrow{x(y)} Q'$.
Then: $\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P' \mid Q') :: z{:}C[\omega']$

These lemmas carry over straightforwardly from [6]. The new lemmas are:

▶ **Lemma A.4** (Reduction Lemma - $\forall$). *Assume*
(a) $\Omega;\Gamma;\Delta_1 \vdash P :: x{:}\forall\alpha.A[\omega_2]$ *with* $P \xrightarrow{x(w_3)} P'$ *and*
(b) $\Omega;\Gamma;\Delta_2, x{:}\forall\alpha.A[\omega_2] \vdash Q :: z{:}C[\omega_1]$ *with* $Q \xrightarrow{\overline{x\,w_3}} Q'$.
Then: $\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P' \mid Q') :: z{:}C[\omega_1]$

▶ **Lemma A.5** (Reduction Lemma - $\exists$). *Assume*
(a) $\Omega;\Gamma;\Delta_1 \vdash P :: x{:}\exists\alpha.A[\omega_2]$ *with* $P \xrightarrow{\overline{x\,w_3}} P'$ *and*
(b) $\Omega;\Gamma;\Delta_2, x{:}\exists\alpha.A[\omega_2] \vdash Q :: z{:}C[\omega_1]$ *with* $Q \xrightarrow{x(w_3)} Q'$.
Then: $\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P' \mid Q') :: z{:}C[\omega_1]$

▶ **Lemma A.6** (Reduction Lemma - @). *Assume*
(a) $\Omega;\Gamma;\Delta_1 \vdash P :: x{:}@_\omega A[\omega']$ *with* $P \xrightarrow{\overline{x.y@\omega}} P'$ *and*
(b) $\Omega;\Gamma;\Delta_2, x{:}@_\omega A[\omega'] \vdash Q :: z{:}C[\omega'']$ *with* $Q \xrightarrow{x.y@\omega} Q'$.
Then: $\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P' \mid Q') :: z{:}C[\omega'']$

The proofs of the lemmas above follow by simultaneous induction on the two given typing derivations, with Lemmas A.4 and A.5 making use of Lemma 3.2. This development is essentially that of [5] and [39] which consider an extension of the core propositional system of [6] with communication of types (i.e. polymorphism) and communication of data (i.e. value dependencies). By appealing to such lemmas, we can establish type preservation for our system.

A.5 Pre-congruence on Local Types

The following definition is used in the proof of Thm. 4.12:

▶ **Definition A.7.** *We define $\preceq^{\sqcup}_{\downarrow}$ as the least pre-congruence relation on local types such that*

$$T_1 \preceq^{\sqcup}_{\downarrow} T_1 \sqcup T_2 \qquad T_1 \preceq^{\sqcup}_{\downarrow} T_2 \Rightarrow \downarrow\alpha.T_1 \preceq^{\sqcup}_{\downarrow} T_2\{^{\omega}/_{\alpha}\} \qquad T \preceq^{\sqcup}_{\downarrow} \downarrow\alpha.T \quad \text{if } \alpha \text{ does not occur in } T$$

A.6 Proofs of Medium Characterization

The proof of Theorem 4.11 relies on the following auxiliary proposition:

▶ **Proposition A.8.** *Let*

1. $\Omega;\Gamma;\Delta_1 \vdash \mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{y}) :: z : \mathbf{1}[\omega_m]$, *with* $dom(\Delta_1) = \{y_{\mathtt{p}}, y_{\mathtt{q}_1}, \ldots, y_{\mathtt{q}_n}\}$
2. $\Omega;\Gamma;\Delta_2 \vdash y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}_1}(m_{\mathtt{q}_1}@\omega_{\mathtt{q}_1}).\cdots.y_{\mathtt{q}_n}(m_{\mathtt{q}_n}@\omega_{\mathtt{q}_n}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m}) :: z : \mathbf{1}[\omega_m]$

be two compositional typings. Then

$$\Omega;\Gamma;\Delta_1 \circ \Delta_2 \vdash \mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{y}) \circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}_1}(m_{\mathtt{q}_1}@\omega_{\mathtt{q}_1}).\cdots.y_{\mathtt{q}_n}(m_{\mathtt{q}_n}@\omega_{\mathtt{q}_n}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m}) :: z : \mathbf{1}[\omega_m]$$

is a compositional typing, where the typing environment $\Delta_1 \circ \Delta_2$ is defined as follows:

$$\Delta_1 \circ \Delta_2(c) = \begin{cases} c : \langle\!\langle T \rangle\!\rangle[\omega] & \text{if } c \in dom(\Delta_j) \text{ and } c \notin dom(\Delta_i), \text{ with } i,j \in \{1,2\}, i \neq j \\ c : \langle\!\langle T_1 \circ T_2 \rangle\!\rangle[\omega] & \text{if } c : \langle\!\langle T_1 \rangle\!\rangle[\omega] \in \Delta_1 \text{ and } c : \langle\!\langle T_2 \rangle\!\rangle[\omega] \in \Delta_2 \end{cases}$$

Proof (Sketch). We must prove the existence of a typing derivation for the resulting fused process. We start by observing that the compositional typing for $\mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{y})$ ensures that its associated derivation will contain one or more occurrences of the sequent

$$\Omega;\Gamma;y_{\mathtt{p}} : \mathbf{1}[\omega_{\mathtt{p}}], y_{\mathtt{q}_1} : \mathbf{1}[\omega_{\mathtt{q}_1}], \ldots, y_{\mathtt{q}_n} : \mathbf{1}[\omega_{\mathtt{q}_n}] \vdash \mathbf{0} :: z : \mathbf{1}[\omega_m] \tag{1}$$

corresponding to one or more occurrences of `end` in G_1 (possible because of labeled choices in G_1): indeed, by Def. 4.8 we have $\mathsf{M}^{\tilde{\omega}}[\![\mathtt{end}]\!](\tilde{y}) = \mathbf{0}$ and by Def. 4.10 we have $\langle\!\langle \mathtt{end} \rangle\!\rangle = \mathbf{1}$. Observe that the compositional typing for

$$y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}_1}(m_{\mathtt{q}_1}@\omega_{\mathtt{q}_1}).\cdots.y_{\mathtt{q}_n}(m_{\mathtt{q}_n}@\omega_{\mathtt{q}_n}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m}) \tag{2}$$

ensures that $\{y_{\mathtt{p}}, y_{\mathtt{q}_1}, \ldots, y_{\mathtt{q}_n}\} \subseteq dom(\Delta_2)$, i.e., Δ_2 contains judgements for at least the names in $dom(\Delta_1)$—it may also contain other judgments, corresponding to participants that intervene in G_2 but not in the sub-protocol G_1. Given this, the compositional typing for the fused process

$$\mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{y}) \circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}_1}(m_{\mathtt{q}_1}@\omega_{\mathtt{q}_1}).\cdots.y_{\mathtt{q}_n}(m_{\mathtt{q}_n}@\omega_{\mathtt{q}_n}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})$$

is obtained by "stacking up" the typing derivation for (2) exactly on the occurrences of sequents of the form (1) in the typing derivation for $\mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{y})$. This is fully consistent with definitions of fusion for processes (Def. 4.7) and local types (Def. 4.4): the former decrees that $\mathbf{0} \circ P = P$ whereas the latter decrees that $\mathtt{end} \circ T = T$. In the resulting "stacked" typing derivation, the types for $y_{\mathtt{p}}, y_{\mathtt{q}_1}, \ldots, y_{\mathtt{q}_n}$ that correspond to the behavior of $\mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{y})$ can be derived exactly as in the derivation of the first assumption, now starting from the types $\Delta_2(y_{\mathtt{p}}), \Delta_2(y_{\mathtt{q}_1}), \ldots, \Delta_2(y_{\mathtt{q}_n})$ rather than from $\mathbf{1}$. ◀

▶ **Theorem 4.11** (Global Types → Typed Mediums). *If G is WF with* $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ *then* $\Omega; \Gamma; c_{\mathtt{p}_1}{:}\langle\!\langle G\upharpoonright \mathtt{p}_1 \rangle\!\rangle[\omega_1], \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G\upharpoonright \mathtt{p}_n \rangle\!\rangle[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ *is a compositional typing, for some* Ω, Γ, *with* $\tilde{\omega} = \omega_1, \ldots, \omega_n$. *We assume that* $\omega_i \prec \omega_m$ *for all* $i \in \{1, \ldots, n\}$ *(the medium's domain is accessible by all), and that* $i \neq j$ *implies* $\omega_i \neq \omega_j$.

Proof. By induction on the structure of G. There are three cases. The base case, $G = \mathtt{end}$, is immediate as there are no participants.

The case $G = \mathtt{p}_1 \twoheadrightarrow \mathtt{p}_2{:}\{l_i\langle U_i\rangle.G^i\}_{i\in I}$ is exactly as in [4], but we report it here for the sake of completeness. By the well-formedness assumption (Def. 4.6), local types $G\upharpoonright \mathtt{p}_1, \ldots, G\upharpoonright \mathtt{p}_n$ are all defined. Writing $\mathtt{p}$ and $\mathtt{q}$ instead of $\mathtt{p}_1$ and $\mathtt{p}_2$, by Def. 4.5 we have:

$$G\upharpoonright \mathtt{p} \;=\; \mathtt{p}!\{l_i\langle U_i\rangle.G^i\upharpoonright \mathtt{p}\}_{i\in I} \tag{3}$$

$$G\upharpoonright \mathtt{q} \;=\; \mathtt{p}?\{l_i\langle U_i\rangle.G^i\upharpoonright \mathtt{q}\}_{i\in I} \tag{4}$$

$$G\upharpoonright \mathtt{p}_j \;=\; \sqcup_{i\in I}\, G^i\upharpoonright \mathtt{p}_j \quad \text{for every } j \in \{3, \ldots, n\} \tag{5}$$

We need to show that, for some Ω and Γ,

$$\Omega; \Gamma; c_{\mathtt{p}}{:}\langle\!\langle G\upharpoonright \mathtt{p} \rangle\!\rangle[\omega_{\mathtt{p}}], c_{\mathtt{q}}{:}\langle\!\langle G\upharpoonright \mathtt{q} \rangle\!\rangle[\omega_{\mathtt{q}}], \Delta \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m] \tag{6}$$

is a compositional typing, with $D = c_{\mathtt{p}_3}{:}\langle\!\langle G\upharpoonright \mathtt{p}_3 \rangle\!\rangle[\omega_{\mathtt{p}_3}], \cdots, c_{\mathtt{p}_n}{:}\langle\!\langle G\upharpoonright \mathtt{p}_n \rangle\!\rangle[\omega_{\mathtt{p}_n}]$.

Without loss of generality, we detail the case $I = \{1, 2\}$. By Def. 4.8, we have:

$$\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) = c_{\mathtt{p}} \triangleright \begin{cases} l_1 : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_1; \overline{c_{\mathtt{q}}}\langle v\rangle.([u \leftrightarrow v] \mid \mathsf{M}^{\tilde{\omega}}[\![G^1]\!](\tilde{c})) \\ l_2 : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_2; \overline{c_{\mathtt{q}}}\langle v\rangle.([u \leftrightarrow v] \mid \mathsf{M}^{\tilde{\omega}}[\![G^2]\!](\tilde{c})) \end{cases}$$

and by combining (3) and (4) with Def. 4.10 we have:

$$\langle\!\langle G\upharpoonright \mathtt{p} \rangle\!\rangle = \oplus\{l_1 : \langle\!\langle U_1 \rangle\!\rangle \otimes \langle\!\langle G^1\upharpoonright \mathtt{p} \rangle\!\rangle\,,\; l_2 : \langle\!\langle U_2 \rangle\!\rangle \otimes \langle\!\langle G^2\upharpoonright \mathtt{p} \rangle\!\rangle\}_{i\in I}$$

$$\langle\!\langle G\upharpoonright \mathtt{q} \rangle\!\rangle = \&\{l_1 : \langle\!\langle U_1 \rangle\!\rangle \multimap \langle\!\langle G^1\upharpoonright \mathtt{q} \rangle\!\rangle\,,\; l_2 : \langle\!\langle U_2 \rangle\!\rangle \multimap \langle\!\langle G^2\upharpoonright \mathtt{q} \rangle\!\rangle\}_{i\in I}$$

Now, by assumption G is WF; then, by construction, both G^1 and G^2 are WF too. Therefore, by using IH twice we may infer that both

$$\Omega; \Gamma; c_{\mathtt{p}}{:}\langle\!\langle G^1\upharpoonright \mathtt{p} \rangle\!\rangle[\omega_{\mathtt{p}}],\; c_{\mathtt{q}}{:}\langle\!\langle G^1\upharpoonright \mathtt{q} \rangle\!\rangle[\omega_{\mathtt{q}}],\; \Delta_1 \vdash \mathsf{M}^{\tilde{\omega}}[\![G^1]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m] \tag{7}$$

$$\Omega; \Gamma; c_{\mathtt{p}}{:}\langle\!\langle G^2\upharpoonright \mathtt{p} \rangle\!\rangle[\omega_{\mathtt{p}}],\; c_{\mathtt{q}}{:}\langle\!\langle G^2\upharpoonright \mathtt{q} \rangle\!\rangle[\omega_{\mathtt{q}}],\; \Delta_2 \vdash \mathsf{M}^{\tilde{\omega}}[\![G^2]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m] \tag{8}$$

are compositional typings, for any Ω and Γ, with $\Delta_1 = c_{\mathtt{p}_3}{:}\langle\!\langle G^1 \upharpoonright \mathtt{p}_3 \rangle\!\rangle[\omega_{\mathtt{p}_3}], \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G^1 \upharpoonright \mathtt{p}_n \rangle\!\rangle[\omega_{\mathtt{p}_n}]$ and $\Delta_2 = c_{\mathtt{p}_3}{:}\langle\!\langle G^2\upharpoonright \mathtt{p}_3 \rangle\!\rangle[\omega_{\mathtt{p}_3}], \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G^2\upharpoonright \mathtt{p}_n \rangle\!\rangle[\omega_{\mathtt{p}_n}]$.

Now, to obtain a compositional typing for $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, we must consider that Δ_1 and Δ_2 may not be identical. This is due to the merge-based well-formedness assumption, which admits non identical behaviors in branches G^1 and G^2 in the case of (local) branching types.

We proceed by induction on k, defined as the size of Δ_1 and Δ_2 (note that $k = n - 2$).

1. (Case $k = 0$): Then $\Delta_1 = \Delta_2 = \emptyset$ and $\mathtt{p}$ and $\mathtt{q}$ are the only participants in G. Let us write $A_{\mathtt{q}}$ to stand for the session type

$$\&\{l_1 : \langle\!\langle U_1 \rangle\!\rangle \multimap \langle\!\langle G^1\upharpoonright \mathtt{q} \rangle\!\rangle\,,\; l_2 : \langle\!\langle U_2 \rangle\!\rangle \multimap \langle\!\langle G^2\upharpoonright \mathtt{q} \rangle\!\rangle\}$$

Based on (7) and (8), following the derivation in Fig. 3, we may derive typings

$$\Omega; \Gamma; c_{\mathtt{p}} : \langle\!\langle U_1 \rangle\!\rangle \otimes \langle\!\langle G^1\upharpoonright \mathtt{p} \rangle\!\rangle[\omega_{\mathtt{p}}],\; c_{\mathtt{q}}{:}A_{\mathtt{q}}[\omega_{\mathtt{q}}] \vdash c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_1; \overline{c_{\mathtt{q}}}\langle v\rangle.([u \leftrightarrow v] \mid \mathsf{M}^{\tilde{\omega}}[\![G^1]\!](\tilde{c})) :: z : \mathbf{1}[\omega_m] \tag{9}$$

$$\Omega; \Gamma; c_{\mathtt{p}} : \langle\!\langle U_2 \rangle\!\rangle \otimes \langle\!\langle G^2\upharpoonright \mathtt{p} \rangle\!\rangle[\omega_{\mathtt{p}}],\; c_{\mathtt{q}}{:}A_{\mathtt{q}}[\omega_{\mathtt{q}}] \vdash c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_2; \overline{c_{\mathtt{q}}}\langle v\rangle.([u \leftrightarrow v] \mid \mathsf{M}^{\tilde{\omega}}[\![G^2]\!](\tilde{c})) :: z : \mathbf{1}[\omega_m] \tag{10}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Omega;\Gamma;u{:}\langle\!\langle U_1\rangle\!\rangle[\omega_\mathtt{p}]\vdash[u\leftrightarrow v]::v:\langle\!\langle U_1\rangle\!\rangle[\omega_\mathtt{p}]}\,(\mathsf{id})\quad\dfrac{}{\Omega;\Gamma;c_\mathtt{p}{:}\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\langle\!\langle G^1\upharpoonright\mathtt{q}\rangle\!\rangle[\omega_\mathtt{q}],\,\Delta_1\vdash\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c)::z:\mathbf{1}[\omega_m]}\,(\mathbf{1}\mathsf{L})}{\Omega;\Gamma;u:\langle\!\langle U_1\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{p}:\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\langle\!\langle U_1\rangle\!\rangle\multimap\langle\!\langle G^1\upharpoonright\mathtt{q}\rangle\!\rangle[\omega_\mathtt{q}],\Delta_1\vdash\overline{c_\mathtt{q}}\langle v\rangle.([u\leftrightarrow v]\mid\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c))::z:\mathbf{1}[\omega_m]}\,(\multimap\mathsf{L})}{\Omega;\Gamma;u:\langle\!\langle U_1\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{p}:\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\,\&\{l_1:\langle\!\langle U_1\rangle\!\rangle\multimap\langle\!\langle G^1\upharpoonright\mathtt{q}\rangle\!\rangle\}_{\{1\}}[\omega_\mathtt{q}],\Delta_1\vdash c_\mathtt{q}\triangleleft l_1;\overline{c_\mathtt{q}}\langle v\rangle.([u\leftrightarrow v]\mid\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c))::z:\mathbf{1}[\omega_m]}\,(\&\mathsf{L}_1)}{\Omega;\Gamma;u:\langle\!\langle U_1\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{p}:\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\,\&\{l_i:(\langle\!\langle U_i\rangle\!\rangle\multimap\langle\!\langle G^i\upharpoonright\mathtt{q}\rangle\!\rangle)\}_{i\in I}[\omega_\mathtt{q}],\Delta_1\vdash c_\mathtt{q}\triangleleft l_1;\overline{c_\mathtt{q}}\langle v\rangle.([u\leftrightarrow v]\mid\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c))::z:\mathbf{1}[\omega_m]}\,(\&\mathsf{L}_2)}{\Omega;\Gamma;c_\mathtt{p}:\langle\!\langle U_1\rangle\!\rangle\otimes\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\,\&\{l_i:(\langle\!\langle U_i\rangle\!\rangle\multimap\langle\!\langle G^i\upharpoonright\mathtt{q}\rangle\!\rangle)\}_{i\in I}[\omega_\mathtt{q}],\Delta_1\vdash c_\mathtt{p}(u).c_\mathtt{q}\triangleleft l_1;\overline{c_\mathtt{q}}\langle v\rangle.([u\leftrightarrow v]\mid\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c))::z:\mathbf{1}[\omega_m]}\,(\otimes\mathsf{L})$$

Figure 3 Derivation for $c_\mathtt{p}(u).c_\mathtt{q}\triangleleft l_1;\overline{c_\mathtt{q}}\langle v\rangle.([u\leftrightarrow v]\mid\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c))$.

Then, using (9) and (10) we may derive, using Rule ($\oplus\mathsf{L}$):

$$\Omega;\Gamma;c_\mathtt{p}:\&\{l_1:\langle\!\langle U_1\rangle\!\rangle\otimes\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle\,,\;l_2:\langle\!\langle U_2\rangle\!\rangle\otimes\langle\!\langle G^2\upharpoonright\mathtt{p}\rangle\!\rangle\}[\omega_\mathtt{p}],\,c_\mathtt{q}{:}A_\mathtt{q}[\omega_\mathtt{p}]\vdash\mathsf{M}^{\tilde\omega}[\![G]\!](\tilde c)::z:\mathbf{1}[\omega_m]$$

and the thesis follows.

2. (Case $k>0$): Then there exists a participant $\mathtt{p}_k$, types $B_1=\langle\!\langle G^1\upharpoonright\mathtt{p}_k\rangle\!\rangle$, $B_2=\langle\!\langle G^2\upharpoonright\mathtt{p}_k\rangle\!\rangle$ and environments Δ_1',Δ_2' such that $\Delta_1=c_{\mathtt{p}_k}{:}B_1[\omega_{\mathtt{p}_k}],\Delta_1'$ and $\Delta_2=c_{\mathtt{p}_k}{:}B_2[\omega_{\mathtt{p}_k}],\Delta_2'$.
By induction hypothesis, there is a compositional typing starting from

$$\Omega;\Gamma;c_\mathtt{p}{:}\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\langle\!\langle G^1\upharpoonright\mathtt{q}\rangle\!\rangle[\omega_\mathtt{q}],\,\Delta_1'\vdash\mathsf{M}^{\tilde\omega}[\![G^1]\!](\tilde c)::z:\mathbf{1}[\omega_m]$$
$$\Omega;\Gamma;c_\mathtt{p}{:}\langle\!\langle G^2\upharpoonright\mathtt{p}\rangle\!\rangle[\omega_\mathtt{p}],\,c_\mathtt{q}{:}\langle\!\langle G^2\upharpoonright\mathtt{q}\rangle\!\rangle[\omega_\mathtt{q}],\,\Delta_2'\vdash\mathsf{M}^{\tilde\omega}[\![G^2]\!](\tilde c)::z:\mathbf{1}[\omega_m]$$

resulting into

$$\Omega;\Gamma;c_\mathtt{p}:\&\{l_1:\langle\!\langle U_1\rangle\!\rangle\otimes\langle\!\langle G^1\upharpoonright\mathtt{p}\rangle\!\rangle\,,\;l_2:\langle\!\langle U_2\rangle\!\rangle\otimes\langle\!\langle G^2\upharpoonright\mathtt{p}\rangle\!\rangle\}[\omega_\mathtt{p}],\,c_\mathtt{q}{:}A_\mathtt{q}[\omega_\mathtt{q}],\Delta_1'\vdash\mathsf{M}^{\tilde\omega}[\![G]\!](\tilde c)::z:\mathbf{1}[\omega_m]$$

since $\Delta_1'=\Delta_2'$.
To extend the typing derivation to Δ_1 and Δ_2, we proceed by a case analysis on the shape of B_1 and B_2. We aim to show that either (a) B_1 and B_2 are already identical base or session types or (b) that typing allows us to transform them into identical types. We rely on the definition of $\sqcup$ (Def. 4.3). There are two main sub-cases:

a. Case $B_1\neq\&\{l_h:A_h\}_{h\in H}$: Then, since Def. 4.3 decrees $T\sqcup T=T$ and the fact that merge-based well-definedness depends on $\sqcup$, we may infer $B_2=B_1$. Hence, $\Delta_1=\Delta_2$ and the desired derivation is obtained as in the base case.

b. Case $B_1=\&\{l_h:A_h\}_{h\in H}$: This is the interesting case: even if merge-based well-formedness of G ensures that both B_1 and B_2 are selection types, they may not be identical. If B_1 and B_2 are identical then we proceed as in previous sub cases. Otherwise, then due to $\sqcup$ there are some finite number of labeled alternatives in B_1 but not in B_2 and/or viceversa. Also, Def. 4.3 ensures that common options (if any) are identical in both branches. We may then use Rule ($\&\mathsf{L}_2$) to "complement" occurrences of types B_1 and B_2 in (7) and (8) as appropriate to make them coincide and achieve identical typing. This rule is silent; as labels are finite, this completing task is also finite, and results into $\Delta_1=\Delta_2$.

Finally, we have the case $G=\mathtt{p}\,\mathsf{moves}\,\mathtt{q}_1,\ldots,\mathtt{q}_n\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$. Without loss of generality, we consider the global type $G=\mathtt{p}\,\mathsf{moves}\,\mathtt{q}\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$, i.e., the type in which the sub-protocol G_1 only involves two participants, namely $\mathtt{p}$ and $\mathtt{q}$. By the well-formedness

assumption, local types $G\upharpoonright \mathtt{p}$, $G\upharpoonright \mathtt{q}$, $G\upharpoonright \mathtt{p}_3, \ldots, G\upharpoonright \mathtt{p}_n$ are all defined. In particular, by Def. 4.5 we have:

$$G\upharpoonright \mathtt{p} \;=\; \downarrow\beta.(\exists\alpha.@_\alpha\, G_1\upharpoonright \mathtt{p}) \circ @_\beta\, G_2\upharpoonright \mathtt{p} \tag{11}$$

$$G\upharpoonright \mathtt{q} \;=\; \downarrow\beta.(\forall\alpha.@_\alpha\, G_1\upharpoonright \mathtt{q}) \circ @_\beta\, G_2\upharpoonright \mathtt{q} \tag{12}$$

We need to show that, for some Γ,

$$\Omega;\Gamma;c_{\mathtt{p}}{:}\langle\!\langle G\upharpoonright \mathtt{p}\rangle\!\rangle[\omega_{\mathtt{p}}],\ c_{\mathtt{q}}{:}\langle\!\langle G\upharpoonright \mathtt{q}\rangle\!\rangle[\omega_{\mathtt{q}}],\ \Delta\{\tilde{c}/\tilde{m}\} \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m] \tag{13}$$

with $\Delta\{\tilde{c}/\tilde{m}\} = c_{\mathtt{p}_3}{:}\langle\!\langle G\upharpoonright \mathtt{p}_3\rangle\!\rangle[\omega_3],\ \ldots,\ c_{\mathtt{p}_n}{:}\langle\!\langle G\upharpoonright \mathtt{p}_n\rangle\!\rangle[\omega_n]$ is a compositional typing. By Def. 4.8, we have:

$$\begin{aligned}\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) =& c_{\mathtt{p}}(\alpha).c_{\mathtt{q}}\langle\alpha\rangle.c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}}(y_{\mathtt{q}}@\alpha).\\ & \mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y}) \circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})\end{aligned} \tag{14}$$

and by combining (11) and (12) with Def. 4.10 we have:

$$\langle\!\langle G\upharpoonright \mathtt{p}\rangle\!\rangle \;=\; \downarrow\beta.\exists\alpha.@_\alpha\langle\!\langle G_1\upharpoonright \mathtt{p} \circ @_\beta\, G_2\upharpoonright \mathtt{p}\rangle\!\rangle$$

$$\langle\!\langle G\upharpoonright \mathtt{q}\rangle\!\rangle \;=\; \downarrow\beta.\forall\alpha.@_\alpha\langle\!\langle G_1\upharpoonright \mathtt{q} \circ @_\beta\, G_2\upharpoonright \mathtt{q}\rangle\!\rangle$$

Now, by assumption G is WF; then, by construction both G_1 and G_2 are WF too. Therefore, by using IH twice we may infer that both

$$\Omega;\Gamma;y_{\mathtt{p}}{:}\langle\!\langle G_1\upharpoonright \mathtt{p}\rangle\!\rangle[\alpha],\ y_{\mathtt{q}}{:}\langle\!\langle G_1\upharpoonright \mathtt{q}\rangle\!\rangle[\alpha] \vdash \mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y}) :: z : \mathbf{1}[\omega_m] \tag{15}$$

$$\Omega;\Gamma;m_{\mathtt{p}}{:}\langle\!\langle G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\omega_{\mathtt{p}}],\ m_{\mathtt{q}}{:}\langle\!\langle G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\omega_{\mathtt{q}}],\ \Delta \vdash \mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m}) :: z : \mathbf{1}[\omega_m] \tag{16}$$

are compositional typings, for any Ω, Γ, with Δ as above. Using Rule (@L) twice, assuming $\alpha \prec \omega_1$ and $\alpha \prec \omega_2$, from (16) we can derive:

$$\Omega;\Gamma;y_{\mathtt{p}}{:}@_{\omega_1}\,\langle\!\langle G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\alpha],\ y_{\mathtt{q}}{:}@_{\omega_2}\,\langle\!\langle G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\alpha],\ \Delta \vdash y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{p}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m}) \tag{17}$$

Using Prop. A.8 on (15) and (17) we obtain:

$$\begin{aligned}&\Omega;\Gamma;y_{\mathtt{p}}{:}\langle\!\langle G_1\upharpoonright \mathtt{p} \circ @_{\omega_1}\, G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\alpha],\ y_{\mathtt{q}}{:}\langle\!\langle G_1\upharpoonright \mathtt{q} \circ @_{\omega_2}\, G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\alpha],\ \Delta\\ &\vdash \mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y}) \circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})\end{aligned} \tag{18}$$

We then have the following derivation, which completes the proof for this case:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(18)}{\Omega,\omega_1\prec\alpha,\omega_2\prec\alpha;\Gamma;y_{\mathtt{p}}{:}\langle\!\langle G_1\upharpoonright \mathtt{p}\circ@_{\omega_1}G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\alpha],\,c_{\mathtt{q}}{:}@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{q}\circ@_{\omega_2}G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\omega_2],\,\Delta \vdash c_{\mathtt{q}}(y_{\mathtt{q}}@\alpha).\mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y})\circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})}\,(@\mathsf{L})}{\Omega,\omega_1\prec\alpha,\omega_2\prec\alpha;\Gamma;c_{\mathtt{p}}{:}@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{p}\circ@_{\omega_1}G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\omega_1],\,y_{\mathtt{q}}{:}@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{q}\circ@_{\omega_2}G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\omega_2],\,\Delta \vdash c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}}(y_{\mathtt{q}}@\alpha).\mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y})\circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})}\,(@\mathsf{L})}{\Omega,\omega_1\prec\alpha,\omega_2\prec\alpha;\Gamma;c_{\mathtt{p}}{:}@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{p}\circ@_{\omega_1}G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\omega_1],\,c_{\mathtt{q}}{:}\forall\alpha.@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{q}\circ@_{\omega_2}G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\omega_2],\,\Delta \vdash c_{\mathtt{q}}\langle\alpha\rangle.c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}}(y_{\mathtt{q}}@\alpha).\mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y})\circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})}\,(\forall\mathsf{L})}{\Omega;\Gamma;c_{\mathtt{p}}{:}\exists\alpha.@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{p}\circ@_{\omega_1}G_2\upharpoonright \mathtt{p}\rangle\!\rangle[\omega_1],\,c_{\mathtt{q}}{:}\forall\alpha.@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{q}\circ@_{\omega_2}G_2\upharpoonright \mathtt{q}\rangle\!\rangle[\omega_2],\,\Delta \vdash c_{\mathtt{p}}(\alpha).c_{\mathtt{q}}\langle\alpha\rangle.c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}}(y_{\mathtt{q}}@\alpha).\mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y})\circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})}\,(\exists\mathsf{L})}{\Omega;\Gamma;c_{\mathtt{p}}{:}\downarrow\beta.(\exists\alpha.@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{p}\circ@_{\omega_1}G_2\upharpoonright \mathtt{p}\rangle\!\rangle)[\omega_1],\,c_{\mathtt{q}}{:}\downarrow\beta.(\forall\alpha.@_\alpha.\,\langle\!\langle G_1\upharpoonright \mathtt{q}\circ@_{\omega_2}G_2\upharpoonright \mathtt{q}\rangle\!\rangle)[\omega_2],\,\Delta \vdash c_{\mathtt{p}}(\alpha).c_{\mathtt{q}}\langle\alpha\rangle.c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}}(y_{\mathtt{q}}@\alpha).\mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}},\alpha/\omega_{\mathtt{q}}\}}[\![G_1]\!](\tilde{y})\circ y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}}(m_{\mathtt{q}}@\omega_{\mathtt{q}}).\mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})}\,(\downarrow\mathsf{L})\times 2$$

◀

▶ **Theorem 4.12** (Well-Typed Mediums → Global Types)**.** *Let G be a global type (cf. Def. 4.1). If $\Omega;\Gamma;c_{\mathtt{p}_1}{:}A_1[\omega_1],\ldots,c_{\mathtt{p}_n}{:}A_n[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ is a compositional typing then $\exists T_1,\ldots,T_n$ such that $G\upharpoonright \mathtt{p}_j \preceq^{\sqcup}_{\downarrow} T_j$ and $\langle\!\langle T_j \rangle\!\rangle = A_j$, for all $\mathtt{p}_j \in \mathsf{part}(G)$.*

Proof (Sketch). By induction on the structure of G, exploiting the definition of $\preceq^{\sqcup}_{\downarrow}$ (cf. Def. A.7). There are three cases:

1. (Case $G = \mathtt{end}$): Then $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) = \mathbf{0}$, $\mathsf{part}(G) = \emptyset$, and the thesis follows vacuously. Notice that from the assumption $\Omega;\Gamma;\cdot \vdash \mathsf{M}[\![G]\!]$ and Rule (**1**L) we may derive

$$\Omega;\Gamma;c_{\mathtt{r}_j}{:}\mathbf{1} \vdash \mathsf{M}[\![G]\!] :: z : \mathbf{1}[\omega_m]$$

   for any $\mathtt{r}_j$. In that case, observe that Def. 4.5 decrees that $\mathtt{end}\upharpoonright \mathtt{r}_j = \mathtt{end}$, for any $\mathtt{r}_j$. The thesis holds, for $\langle\!\langle \mathtt{end} \rangle\!\rangle = \mathbf{1}$ (cf. Def. 4.10).
2. (Case $G = \mathtt{p}_1 \twoheadrightarrow \mathtt{p}_2{:}\{l_i\langle U_i\rangle.G^i\}_{i\in I}$) This case follows by typing inversion on the structure of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, using the derivation presented in the second case of the proof of Thm. 4.11. The main aspect to consider are the possible uses of the silent Rule ($\&\mathsf{L}_2$). To prove the correspondence between the types of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ for $c_{\mathtt{p}_2}$ and $G\upharpoonright \mathtt{p}_2$, we exploit the first axiom of Def. A.7 to appropriately handle/prune silently added alternatives in the branching.
3. (Case $G = \mathtt{p}\,\mathsf{moves}\,\mathtt{q}_1,\ldots,\mathtt{q}_n\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$) This case also follows by typing inversion on the structure of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, using the derivation presented in the third case of the proof of Thm. 4.11. The main aspect to consider for the intended correspondence is that Rule ($\downarrow\mathsf{L}$) silently induces type constructs of the form $\downarrow\alpha.A$ within the types for $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, or . To obtain the correspondence, we exploit the second and third axioms of Def. A.7, which make explicit required "here" operators in the type and remove spurious "heres", respectively.

◀

B Extended Examples

B.1 Negotiation Procedure

This example is adapted from [14], consisting of a negotiation procedure *Nego* between two participants of a three-party interaction. The negotiation consists of an agreement on a contract: one participant specifies a request, while the other offers a corresponding contract. The first participant may either accept the contract and end the protocol or make a counter-offer. For the sake of conciseness we assume that the counter-offer is accepted:

$$\begin{aligned} \mathit{Nego}_{\mathtt{p},\mathtt{q}} \triangleq\ & \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{\mathit{ask}\langle \mathit{terms}\rangle.\ \mathtt{q} \twoheadrightarrow \mathtt{p}{:}\{\mathit{proposition}\langle \mathit{contract}_1\rangle. \\ & \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{\mathit{accept}.\mathtt{end},\ \mathit{counter}\langle \mathit{contract}_1\rangle.\ \mathtt{q} \twoheadrightarrow \mathtt{p}{:}\{\mathit{accept}.\mathtt{end}\}\}\}\} \end{aligned}$$

The main protocol consists of a client, an agent and an instrument, each initially in their own domains. The client first sends a request to the agent for some instrument they wish to use. The agent connects to the instrument which acknowledges when available. The agent then enters the negotiation sub-protocol with the client (via protocol *Nego*), by having both agent and the client migrate to domain d_n. This movement models the trusted setting at which the agent and the client coexist in order to successfully negotiate the instrument usage. After the negotiation stage is complete, both the client and the instrument migrate to a common domain d_i to perform the rest of the protocol, which for the sake of conciseness we model with the client either aborting the interaction or sending a command to the instrument and then receiving back the appropriate result:

$$\begin{aligned}&\mathsf{client} \twoheadrightarrow \mathsf{agent}:\{\ req\langle coord\rangle.\ \mathsf{agent} \twoheadrightarrow \mathsf{instr}:\{connect.\ \mathsf{instr} \twoheadrightarrow \mathsf{agent}:\{available.\ \mathsf{agent} \twoheadrightarrow \mathsf{client}:\{ack.\\ &\qquad\qquad \mathsf{agent\ moves\ client\ to}\ d_n\ \mathsf{for}\ Nego_{\mathsf{agent,client}}\ ;\\ &\qquad\qquad \mathsf{client\ moves\ instr\ to}\ d_i\ \mathsf{for\ client} \twoheadrightarrow \mathsf{instr}:\{\ abort.\mathtt{end},\\ &\qquad\qquad command\langle code\rangle.\mathsf{instr} \twoheadrightarrow \mathsf{client}:\{result\langle data\rangle.\mathtt{end}\ \}\}\}\}\}\ ;\ \mathtt{end}\end{aligned}$$

By leveraging our notion of medium, we can make explicit the fact that the three participants are distributed agents, each located at independent domains that can access the medium substrate (i.e. the domain of the medium). Through the medium-orchestrated interaction, the use of domain migration primitives enables us to explicitly model the various domain movement steps that the participants must follow to implement the protocol. This is in sharp contrast with more traditional approaches to multiparty protocols [25], where such domain specific notions are implicit. The medium for the global type above is (we assume that the three participants initially reside at worlds $w_{\mathtt{client}}, w_{\mathtt{agent}}, w_{\mathtt{instr}}$, respectively):

$$\begin{aligned}&c_{\mathtt{client}} \triangleright \Big\{ req : c_{\mathtt{client}}(u).c_{\mathtt{agent}} \triangleleft req; \overline{c_{\mathtt{agent}}}\langle v\rangle.([u \leftrightarrow v] \mid\\ &\quad c_{\mathtt{agent}} \triangleright \Big\{ connect : c_{\mathtt{instr}} \triangleleft connect; c_{\mathtt{instr}} \triangleright \Big\{ available : c_{\mathtt{agent}} \triangleleft available;\\ &\quad c_{\mathtt{agent}} \triangleright \Big\{ ack : c_{\mathtt{client}} \triangleleft ack; c_{\mathtt{agent}}(d_n).c_{\mathtt{client}}\langle d_n\rangle.c_{\mathtt{agent}}(y_{\mathtt{agent}}@d_n).c_{\mathtt{client}}(y_{\mathtt{client}}@d_n).\\ &\quad \mathsf{M}^{\tilde{d_n}}[\![Nego]\!](y_{\mathtt{agent}}, y_{\mathtt{client}}) \circ (y_{\mathtt{agent}}(z_{\mathtt{agent}}@w_{\mathtt{agent}}).y_{\mathtt{client}}(z_{\mathtt{client}}@w_{\mathtt{client}}).\\ &\qquad z_{\mathtt{client}}(d_i).c_{\mathtt{instr}}\langle d_i\rangle.z_{\mathtt{client}}(y_{\mathtt{client}}@d_i).c_{\mathtt{instr}}(y_{\mathtt{instr}}@d_i).\\ &\qquad y_{\mathtt{client}} \triangleright \Big\{ abort : y_{\mathtt{instr}} \triangleleft abort; \mathbf{0},\\ &\qquad\qquad command : y_{\mathtt{client}}(c).y_{\mathtt{instr}} \triangleleft command; \overline{y_{\mathtt{instr}}}\langle d\rangle.([c \leftrightarrow d] \mid\\ &\qquad\qquad y_{\mathtt{instr}} \triangleright \Big\{ result : y_{\mathtt{instr}}(data).y_{\mathtt{client}} \triangleleft result; \overline{y_{\mathtt{client}}}\langle r\rangle.([data \leftrightarrow r] \mid\\ &\qquad\qquad y_{\mathtt{client}}(z_{\mathtt{client}}@w_{\mathtt{client}}).y_{\mathtt{instr}}(z_{\mathtt{instr}}@w_{\mathtt{instr}}).\mathbf{0})\Big\}\Big)\Big\}\Big\}\Big\}\Big)\Big\}\end{aligned}$$

The first two lines of the medium definition correspond to the initial exchange between the client, the agent and the instrument. The actions after the emission of label *ack* to the client model the migration protocol: the agent emits the domain identifier to the medium, which then forwards it to the client and receives from both participants the session handles $y_{\mathtt{agent}}$ and $y_{\mathtt{client}}$, located at d_n. After the migration takes place, the medium orchestrates the negotation between the agent and the client using the new session handles.

After the negotiation, the agent and the client migrate back to their initial domains $w_{\mathtt{agent}}$ and $w_{\mathtt{client}}$, respectively, and the interactions between the client and the instrument take place: the client and the instrument migrate to d_i, sending to the medium the session handles $y_{\mathtt{client}}$ and $y_{\mathtt{instr}}$, followed by the client emitting an *abort* or *command* message which is forwarded to the instrument. In the latter case, the instrument forwards the result to the client. Finally, both the client and the instrument migrate back to their initial domains.

B.2 Domain-aware Middleware

A common design pattern in distributed computing is the notion of a middleware agent which answers requests from clients, sometimes offloading the requests to some server (e.g. to better manage local resource availability). The mediation between the middleware and the server often involves some form of privilege escalation or specialized authentication, which we can now model via domain migration. We first represent a simple offloading protocol between the middleware $\mathtt{p}$ and the server $\mathtt{q}$:

$$Offload_{\mathtt{p,q}} \triangleq \mathtt{p} \twoheadrightarrow \mathtt{q}:\{req\langle data\rangle.\mathtt{q} \twoheadrightarrow \mathtt{p}:\{reply\langle ans\rangle.\mathtt{end}\}\}$$

The global interaction is represented by the following global type:

$$\begin{aligned}&\mathtt{client} \twoheadrightarrow \mathtt{mw}:\{request\langle req\rangle.\\ &\mathtt{mw} \twoheadrightarrow \mathtt{client}:\{\ reply\langle ans\rangle.\mathtt{mw} \twoheadrightarrow \mathtt{server}:\{done.\mathtt{end}\},\\ &\qquad\qquad wait.\mathtt{mw} \twoheadrightarrow \mathtt{server}:\{init.\mathtt{mw}\ \mathsf{moves}\ \mathtt{server}\ \mathsf{to}\ w_{\mathtt{priv}}\ \mathsf{for}\ Offload_{\mathtt{mw,server}}\ ;\\ &\qquad\qquad \mathtt{mw} \twoheadrightarrow \mathtt{client}:\{reply\langle ans\rangle.\mathbf{0}\}\}\}\end{aligned}$$

The medium for this protocol is given by:

$$
\begin{aligned}
&c_{\texttt{client}} \triangleright \{ \mathit{request} : c_{\texttt{client}}(r).c_{\texttt{mw}} \triangleleft \mathit{request}; \overline{c_{\texttt{mw}}}\langle v\rangle.([r \leftrightarrow v] \mid \\
&c_{\texttt{mw}} \triangleright \{\ \mathit{reply} : c_{\texttt{mw}}(a).c_{\texttt{client}} \triangleleft \mathit{reply}; \overline{c_{\texttt{client}}}\langle n\rangle.([a \leftrightarrow n] \mid c_{\texttt{mw}} \triangleright \{ \mathit{done} : c_{\texttt{server}} \triangleleft \mathit{done}; \mathbf{0}\}), \\
&\qquad \mathit{wait} : c_{\texttt{mw}} \triangleright \{ \mathit{init} : c_{\texttt{server}} \triangleleft \mathit{init}; c_{\texttt{mw}}(w_{\texttt{priv}}).c_{\texttt{server}}\langle w_{\texttt{priv}}\rangle. \\
&\qquad\qquad c_{\texttt{mw}}(y_{\texttt{mw}}@w_{\texttt{priv}}).c_{\texttt{server}}(y_{\texttt{server}}@w_{\texttt{priv}}).\mathsf{M}^{\tilde{w_{\texttt{priv}}}}[\![\mathit{Offload}]\!](y_{\texttt{mw}}, y_{\texttt{server}}) \circ \\
&\qquad\qquad (y_{\texttt{mw}}(z_{\texttt{mw}}@w_{\texttt{mw}}).y_{\texttt{server}}(z_{\texttt{server}}@w_{\texttt{server}}). \\
&\qquad\qquad z_{\texttt{mw}} \triangleright \{ \mathit{reply} : z_{\texttt{mw}}(a).c_{\texttt{client}} \triangleleft \mathit{reply}; \overline{c_{\texttt{client}}}\langle n\rangle.([a \leftrightarrow n] \mid \mathbf{0})\})\}\})\}
\end{aligned}
$$

Notice how the client's domain remains fixed throughout the entire interaction, regardless of whether or not the middleware chooses to interact with the server to fulfil the client request.

B.3 A Secure Communication Domain

Our previous examples explore the use of domains in a general distributed setting. Another interesting aspect of our domain-aware typing discipline is that communication can only take place between reachable domains. In scenarios where participants are each situated in distinct domains, domain movement also governs the ability of participants to interact. For instance, consider the following protocol excerpt:

$$
\begin{aligned}
&\texttt{client} \twoheadrightarrow \texttt{store}{:}\{\mathit{purchase} : \texttt{store}\ \mathsf{moves}\ \mathsf{bank}\ \mathsf{to}\ \texttt{sec}\ \mathsf{for}\ \mathit{SecurePay}\,; \\
&\qquad\qquad \texttt{store} \twoheadrightarrow \texttt{client}{:}\{\mathit{success}\langle \mathit{receipt}\rangle.\texttt{end}, \mathit{fail}.\texttt{end}\}\}
\end{aligned}
$$

The protocol above is part of the interaction between an online store and its clients, where after some number of exchanges the client decides to purchase the contents of their shopping cart. Upon receiving the *purchase* message, the store is meant to enter a secure domain `sec` so it can communicate with the bank role to exchange potentially sensitive data. In a setting where the client, store and bank exist at different domains and where only the store domain can reach the bank domain, our domain-aware typing discipline ensures that no direct communication between the bank and the client domain is possible, with the entirety of the data flows between the client and the bank (via the medium) being captured by the type and medium specification.

C Examples of Domain-aware Binary Sessions

In this section we present two examples that further illustrate the novel features of our domain-aware framework for session communications.

C.1 E-Commerce Example

We revisit the web store example of § 3. Recall the refined web store session type:

$$
\begin{aligned}
\mathsf{WStore}_{\mathsf{sec}} &\triangleq \mathsf{addCart} \multimap \&\{\mathsf{buy} : @_{\mathsf{sec}}\,\mathsf{Payment}, \mathtt{quit} : \mathbf{1}\} \\
\mathsf{Payment} &\triangleq \mathsf{CCNumber} \multimap \oplus\{\mathtt{ok} : (@_{\mathsf{bnk}}\mathsf{Receipt}) \otimes \mathbf{1}, \mathtt{nok} : \mathbf{1}\} \\
\mathsf{Bank} &\triangleq \mathsf{CCNumber} \multimap \oplus\{\mathtt{ok} : \mathsf{Receipt} \otimes \mathbf{1}, \mathtt{nok} : \mathbf{1}\}
\end{aligned}
$$

The process that implements the bank interface is to be accessible from the domain of the web store, moving to a secure domain `bnk` before receiving and validating the payment information. Thus, the bank process typing can be specified with the following judgment, recalling that the domain of the web store is `ws`:

$$\texttt{ws} \prec \texttt{bnk}; \cdot; \cdot \vdash B :: @_{\mathsf{bnk}}\,\mathsf{Bank}[\texttt{ws}]$$

The web store will then use the bank interface to fulfill its interface:

$$\cdot;\cdot;b{:}@_{\mathtt{bnk}}\,\mathsf{Bank}[\mathtt{ws}] \vdash \mathsf{Store} :: z{:}\mathsf{WStore}_{\mathtt{sec}}[\mathtt{ws}]$$

The $@_w$ type constructor allows us to express a very precise form of coordinated domain migration. For instance, typing ensures that in order for the store to produce an output of the form $@_{\mathtt{bnk}}\mathsf{Payment}$ it must first have interacted with the bank domain: in order to produce an output of $@_{\mathtt{bnk}}$, it must be the case that $\mathtt{ws} \prec \mathtt{bnk}$, which is only known to the store process *after* interacting with the bank domain. Alternatively, consider the $\mathsf{WStore}_{\mathtt{sec}}$ service interacting with a client process along channel x, each in their own (reachable) domains, $\mathtt{c}$ and $\mathtt{ws}$, respectively. Our framework ensures that interactions between the client and the web store enjoy session fidelity, progress, and termination guarantees. Concerning domain-awareness, by assuming the client chooses to buy his product selection, we reach a state that is typed as follows:

$$\mathtt{c} \prec \mathtt{ws};\cdot;x{:}@_{\mathtt{sec}}\mathsf{Payment}[\mathtt{ws}] \vdash \mathit{Client} :: z{:}@_{\mathtt{sec}}\mathbf{1}[\mathtt{c}]$$

At this point, it is *impossible* for a (typed) client to interact with the behavior that is protected by the trusted domain $\mathtt{sec}$, since it is not the case that $\mathtt{c} \prec^* \mathtt{sec}$. This ensures, e.g., that a client cannot exploit the payment platform of the web store by accessing the trusted domain in unforeseen ways. Formally, no typing derivation of $\mathtt{c} \prec \mathtt{ws};\cdot;\mathsf{Payment}[\mathtt{sec}] \vdash \mathit{Client} :: z{:}@_{\mathtt{sec}}\mathbf{1}[\mathtt{c}]$ exists (Theorem 3.6). The client can only communicate in the secure domain *after* the web store service has migrated accordingly:

$$\begin{array}{l}\mathtt{c} \prec \mathtt{ws}, \mathtt{ws} \prec \mathtt{sec};\cdot;x'{:}\mathsf{Payment}[\mathtt{sec}] \vdash \mathit{Client}' :: z'{:}\mathbf{1}[\mathtt{sec}] \\ \text{where} \quad \mathit{Client} \triangleq x(x'@\mathtt{sec}).\overline{z}\langle z'@\mathtt{sec}\rangle.\mathit{Client}'\end{array}$$

It is inconvenient (and potentially error-prone) for the payment domain to be hardwired in the type. We can solve this issue via existential quantification as shown in the introduction.

$$\mathsf{WStore}_{\exists} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : \exists\alpha.\, @_\alpha\,\mathsf{Payment}, \mathtt{quit} : \mathbf{1}\}$$

As long as accessibility is irreflexive and antisymmetric, the server-provided payment domain w will not be able to interact with the initial $\mathtt{public}$ domain of the interaction except as specified in the $\mathsf{Payment}$ type.

Alternatively, the server can let the client choose a payment domain by using universal quantification. Compliant server code will only be able to communicate in the client-provided payment domain since the process must be parametric in α.

$$\mathsf{WStore}_{\forall} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : \forall\alpha.\, @_\alpha\,\mathsf{Payment}, \mathtt{quit} : \mathbf{1}\}$$

C.2 Spatial Distribution as in $\lambda 5$

Murphy et al. [32] have proposed a Curry-Howard interpretation of the intuitionistic modal logic S5 [38] to model distributed computation with worlds as explicit loci for computation. Accessibility between worlds was assumed to be reflexive, transitive, and symmetric because each host on a network should be reachable from any other host. Murphy [31] later generalized this to hybrid logic, an idea also present in [27], so that propositions can explicitly refer to worlds. Computation in this model was decidedly *sequential*, and a concurrent extension was proposed as future work. Moreover, the system presented some difficulties in the presence of disjunction, requiring a so-called *action at a distance* without an explicit communication visible in the elimination rule for disjunction.

The present system not only generalizes $\lambda 5$ to permit concurrency through session-typed linearity, but also solves the problem of action at a distance because all communication is explicit in the processes. Due to this issue in the original formulation of $\lambda 5$, we will not attempt here to give a full, computationally adequate interpretation of $\lambda 5$ in HILL (which would generalize [40]), but instead explain the spatially distributed computational interpretation of HILL directly.

- $c : \Box A$. Channel c offering A can be used in any domain.
- $c : \Diamond A$. Channel c is offering A in some (hidden) domain.

These are mapped into HILL (choosing a fresh α each time) with

$$\begin{array}{rcl} \Box A & = & \forall \alpha.@_\alpha A \\ \Diamond A & = & \exists \alpha.@_\alpha A \end{array}$$

A process $P :: c{:}\Box A[w_1]$ will therefore receive, along c, a world w_2 reachable from w_1 and then move to w_2, offering A in domain w_2. Conversely, a process $P :: c{:}\Diamond A[w_1]$ will send a world w_2 along c and then move to w_2, offering A in domain w_2. Processes using such channels will behave dually.

We can now understand the computational interpretation of some of the characteristic axioms of S5, keeping in mind that for this application accessibility is reflexive, transitive, and symmetric (we make no distinction between direct reachability or reachability requiring multiple hops).

- $\mathsf{K}_\Diamond :: z{:}\Box(A \multimap B) \multimap \Diamond A \multimap \Diamond B[w_0]$.
  Here, A is offered along some c_1 at some unknown domain α reachable from w_0. Move the offer of $\Box(A \multimap B)[w_0]$ to α as c_2 and send it c_1 to obtain $c_2 : B[\alpha]$. We abstract this as $\Diamond B[w_0]$, which is possible since $w_0 \prec \alpha$. Intuitively, this axiom captures the fact that a session transformer $A \multimap B$ that can be used in *any* domain may be combined with a session A offered in *some* domain in order to produce a session behavior B, itself in some hidden domain.

$$\begin{array}{rcl} \mathsf{K}_\Diamond & \triangleq & z(x).z(y).y(\alpha).y(c_1@\alpha).x\langle\alpha\rangle.x(c_2@\alpha). \\ & & \overline{c_2}\langle v\rangle.([c_1 \leftrightarrow v] \mid z\langle\alpha\rangle.\overline{z}\langle c_3@\alpha\rangle.[c_2 \leftrightarrow c_3]) \end{array}$$

- $\mathsf{T} :: z{:}\Box A \multimap A[w_0]$.
  Given a session x that offers $\Box A$ at w_0, we offer A at w_0 by appealing to reflexivity, and thus $w_0 \prec w_0$. This means that we can then receive from x a fresh channel $c_1@w_0$, obtaining an ambient session of type A at domain w_0 which we then forward along z. The T axiom captures the fact that a mobile session can in fact be accessed "anywhere".

$$\mathsf{T} \quad \triangleq \quad z(x).x\langle w_0\rangle.x(c_1@w_0).[c_1 \leftrightarrow z]$$

- $\mathsf{5} :: z{:}\Diamond A \multimap \Box\Diamond A[w_0]$
  Given a session x offering $\Diamond A$ at w_0, this means that x is offering the behavior A at some reachable but unknown domain β. We can use this session to provide somewhat of a "link" session, that allows any other domain α to also reach the behavior A at this reachable, unknown, domain. We do this by first receiving α along z, identifying the domain reachable from w_0 which we wish to link to β. We then send along z the fresh session c_1 located at α, along which we shall provide the connection. We may then receive from x the identity of β and a session c_2 located in this domain, send the identity along c_1 and a fresh session c_3, located at β (only possible due to the accessibility relation being an equivalence) which we then forward from c_2 as needed.

$$\begin{array}{rcl} \mathsf{5} & \triangleq & z(x).z(\alpha).z\langle c_1@\alpha\rangle.x(\beta).x(c_2@\beta). \\ & & c_1\langle\beta\rangle.c_1\langle c_3@\beta\rangle.[c_2 \leftrightarrow c_3] \end{array}$$

997 Other axioms can be given similarly straightforward interpretations and process realizations.