Domain-Aware Session Types

Luís Caires
Universidade Nova de Lisboa, Portugal

Jorge A. Pérez
University of Groningen, The Netherlands

Frank Pfenning
Carnegie Mellon University, Pittsburgh, PA, USA

Bernardo Toninho
Universidade Nova de Lisboa, Portugal

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.CONCUR.2019.39

Related Version Omitted proofs and extended examples: `http://arxiv.org/abs/1907.01318`.

Funding Caires and Toninho are supported by NOVA LINCS (Ref. UID/CEC/04516/2019). Pérez is supported by the NWO VIDI Project No. 016.Vidi.189.046: "Unifying Correctness for Communicating Software". Pfenning is supported by NSF Grant No. CCF-1718267: "Enriching Session Types for Practical Concurrent Programming".

1 Introduction

The goal of this paper is to show how existing Curry-Howard interpretations of session types [10, 11] can be generalized to a *domain-aware* setting by relying on an extension of linear logic with features from *hybrid logic* [40, 5]. 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 *typed* 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* [30, 31, 24]. 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 [36], 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 [10, 47, 14], 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 [32] this protocol can be represented as the global type:

$$\begin{aligned}\mathtt{cl} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{request}\langle \mathit{req}\rangle.\ \mathtt{mw} \twoheadrightarrow \mathtt{cl}{:}\{\ &\mathit{reply}\langle \mathit{ans}\rangle.\ \mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{done}.\mathtt{end}\}\ ,\ \ \mathit{wait}.\mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{req}\langle \mathit{data}\rangle.\\ &\mathtt{serv} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{mw} \twoheadrightarrow \mathtt{cl}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{end}\}\}\}\}\}\end{aligned}$$

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, it does not capture that protocols for the middleware and the server often involve some form of privilege escalation or specific authentication – ensuring, e.g., that the server interaction is adequately isolated from the client, or that the escalation must precede the server interactions. These requirements 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 $\mathtt{p}\ \mathsf{moves}\ \tilde{\mathtt{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$, where participant $\mathtt{p}$ leads a migration of participants $\tilde{\mathtt{q}}$ to domain ω in order to perform protocol G_1, who then migrate back to perform protocol G_2.

Consider the global type $\mathit{Offload} \triangleq \mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{req}\langle \mathit{data}\rangle.\mathtt{serv} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{end}\}\}$ in our previous example. Our framework allows us to refactor the global type above as:

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

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 accessible 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 [22, 18], 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* [7] 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 session type discipline for domain-aware interacting processes (§ 3). Building upon an extension of linear logic with features from *hybrid logic* [22, 18] we generalize the Curry-Howard interpretation of session types [10, 11] by interpreting *(modal) worlds* as *domains* where session behavior resides. In our system, types can specify domain *migration* and *communication*; domain mobility is governed by a parametric accessibility relation. Judgments stipulate the services used and realized by processes *and* the domains where sessions should be present. Our type discipline statically enforces session fidelity, global progress and, notably, that communication can only happen between accessible domains.
3. As a specific application, we introduce a framework of domain-aware multiparty sessions (§ 4) that uniformly extends the standard multiparty session framework of [32] with domain-aware migration and communication primitives. Our development leverages our logically motivated domain-aware *binary* sessions (§ 3) to give a precise semantics to multiparty sessions through a (typed) *medium process* that acts as an orchestrator of domain-aware multiparty interactions, lifting the strong correctness properties of typed processes to the multiparty setting. 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).

2 Process Model

We introduce a synchronous π-calculus [42] 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*

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

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 ω, 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 [41], 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. [9]), *reduction* expresses their behavior.

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

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

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 [10, 11] 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 [10, 11], propositions are interpreted as session types of communication channels, proofs as typing derivations, and proof reduction as process communication. As main novelties, here we interpret: logical worlds as *domains*; the hybrid connective $@_\omega A$ as the type of a session that *migrates* to an accessible domain ω; and type-level quantification over worlds $\forall\alpha.A$ and $\exists\alpha.A$ as *domain 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 domains 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*

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

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 [5], with worlds interpreted as domains that are explicitly subject to an *accessibility relation* (in the style of [43]) that is tracked by environment Ω. Intuitively, Ω is made up of direct accessibility hypotheses of the form $\omega_1 \prec \omega_2$, meaning that domain ω_2 is accessible from ω_1.

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 environment Ω just mentioned, our typing judgments consider two kinds of type environments: a *linear* part Δ and an *unrestricted* part Γ. They are subject to different structural properties: weakening and contraction principles hold for Γ but not for Δ. Empty environments are written as '$\cdot$'. We then consider two judgments:

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

Judgment (i) states that ω_1 can directly access ω_2 under the hypotheses in Ω. 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; the session s resides at domain ω, under the accessibility hypotheses Ω, using unrestricted sessions in Γ and linear sessions in Δ. Note that each hypothesis in Γ and Δ is labeled with a specific domain. We omit Ω when it is clear from context.

Typing Rules. Selected typing rules are given in Fig. 1; see [9] for the full listing. 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: a sequent $\Omega; \Gamma; \Delta \vdash P :: z : C[\omega_1]$ is *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 domains mentioned in Δ are accessible from ω_1 (not necessarily in a single *direct* step). No such domain 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), (∀L) and (∃R) require explicit checks, which we discuss below. This invariant statically excludes interaction between sessions in accessible domains (cf. Theorem 3.7).

We briefly discuss some of the typing rules, first noting that we 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 output a fresh name y with type A along z and proceed offering $z{:}B$.

The (cut) rule allows us to compose process P, which offers $x{:}A[\omega_2]$, with process Q, which uses $x{:}A[\omega_2]$ to offer $z{:}C[\omega_1]$. We require that domain ω_2 is accessible from ω_1 (i.e., $\omega_1 \prec^* \omega_2$). We also require $\omega_1 \prec^* \Delta_1$: the domains mentioned in Δ_1 (the context for P) must be accessible from ω_1, which follows from the transitive closure of the accessibility relation ($\prec^*$) using the intermediary domain ω_2. As in [10, 11], composition binds the name x:

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

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. For flexibility and consistency with merge-based projectability in multiparty session types, rules for choice and selection induce a standard notion of session subtyping [26].

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 domain 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) *accessible* domains.

The type operator $@_\omega$ realizes a *domain migration* mechanism which is specified both at the level of types and 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:

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

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 accessible 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 accessible domain (via universal quantification) or to some *a priori* unspecified accessible domain:

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

Rule ($\forall$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 accessible 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 ($\forall$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 accessible from ω_2 it wishes to use, which is achieved by the appropriate output action. Rules ($\exists$L) and ($\exists$R) for the existential quantifier have a dual reading.

Finally, the type-level operator $\downarrow\alpha.A$ allows for a type to refer to its *current* domain:

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

The typing rules that govern $\downarrow\alpha.A$ are completely symmetric and produce no action at the process level, merely instantiating the domain 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 an explicit domain accessibility judgment, we can consider the accessibility relation as a *parameter* of the framework. This allows changing accessibility relations and their 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. Being an interpretation of *hybridized* linear logic, our domain-aware theory is *conservative* wrt the Curry-Howard interpretation of session types in [10, 11], in the following sense: the system in [10, 11] corresponds to the case where every session resides at the same domain. As in [10, 11], the sequent calculus for the underlying (hybrid) linear logic can be recovered from our typing rules by erasing processes and name assignments.

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 occurs between sessions located at the same (or accessible) domain we effectively introduce a new layer of reasoning to session type systems. To illustrate this feature, consider the following session type WStore, which specifies a simple interaction between a web store and its clients:

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

WStore 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. If a banking service (not shown) approves the transaction (via an *ok* message), a receipt is emitted. Representable in existing session type systems (e.g. [10, 47, 31]), types WStore and Pay describe the intended communications but fail to capture the crucial fact that in practice the client's sensitive information should only be requested after entering a secure domain. To address this limitation, we can use type-level domain migration to *refine* WStore and Pay:

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

$\mathsf{WStore}_{\mathsf{sec}}$ decrees that the interactions pertinent to type $\mathsf{Pay}_{\mathsf{bnk}}$ should be preceded by a migration step to the trusted domain $\mathtt{sec}$, which should be directly accessible from $\mathsf{WStore}_{\mathsf{sec}}$'s current domain. The type also specifies that the receipt must originate from a bank domain $\mathtt{bnk}$ (e.g., ensuring that the receipt is never produced by the store without entering $\mathtt{bnk}$). When considering the interactions with a client (at domain $\mathtt{c}$) that checks out their cart, we reach a state that is typed with the following judgment:

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

At this point, it is *impossible* for a (typed) client to interact with the behavior that is protected by the domain $\mathtt{sec}$, since it is not the case that $\mathtt{c} \prec^* \mathtt{sec}$. That is, no judgment of the form $\mathtt{c} \prec \mathtt{ws}; \cdot; \mathsf{Pay}_{\mathsf{bnk}}[\mathtt{sec}] \vdash \mathit{Client}' :: z{:}T[\mathtt{c}]$ is derivable. This ensures, e.g., that a

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

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

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

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

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

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

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

Figure 1 Typing Rules (Excerpt – see [9]).

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

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

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 termination, i.e., processes do not exhibit infinite reduction paths (Theorem 3.5). We note that in the presence of termination, our progress result ensures that communication actions are always guaranteed to take place. 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 formal development of these results relies on a *domain-aware* labeled transition system [9], defined as a simple generalization of the early labelled transition system for the session π-calculus given in [10, 11].

Type Safety and Termination. Following [10, 11], 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). *Suppose* $\Omega \vdash \omega_1 \prec \omega_2$. *Then we have:*

- *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 [10, 11, 8, 44], relying on a series of lemmas relating the result of dual process actions (via our LTS semantics) with typable parallel compositions through the (cut) rule [9]. For session type constructors of [10], the results are unchanged. For the domain-aware session type constructors, the development is identical that of [8] and [44], which deal with communication of types and data terms, respectively. ◀

Following [10, 11], the proof of global progress 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 ($\mathbf{1}$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 [38, 39, 8]. For the system in § 3 without quantifiers, the logical relations correspond to those in [38, 39], extended to carry over Ω. When considering quantifiers, the logical relations resemble those proposed for polymorphic session types in [8], 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 accessibility 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 inaccessible 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 that even if a session in the environment changes domains, typing ensures that such a domain will be (transitively) accessible:

▶ **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) $\omega \prec^* \omega''$.*

4 Domain-Aware Multiparty Session Types

We now shift our attention to multiparty session types [32]. 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, p moves $\widetilde{q}$ to ω 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 [7], 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 [32, 21]. 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*

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

The completed global type is denoted $\mathtt{end}$. 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 $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2$ 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 $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2$ 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 (via $\prec$) by all these migrating participants in G_1. While subprotocols and session delegation may appear as similar, delegation supports a different idiom altogether, and has no support for domain awareness. Unlike delegation, with subprotocols we can specify a point where some of the participants perform a certain protocol *within the same multiparty session* and then return to the main session as an ensemble.

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

Global types are projected onto participants so as to obtain local types. The terminated local type is $\mathtt{end}$. 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 [21]. 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:* $\mathtt{end}$, $\mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$, $@_\omega T$, $\forall\alpha.T$, *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 global type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2$, 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, which safely appends a local type to another:

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

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

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

▶ **Definition 4.5** (Merge-based Projection [21]). *Let G be a global type. The* merge-based projection *of G under participant $\mathtt{r}$, denoted $G{\restriction}\mathtt{r}$, is defined as $\mathsf{end}{\restriction}\mathtt{r} = \mathsf{end}$ and*

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

When no side condition holds, the map is undefined.

The projection for the type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$ 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 $G_1{\restriction}\mathtt{p}$ will be performed. Finally, there is a migration step from ω back to $\omega_\mathtt{p}$; once there, the protocol $G_2{\restriction}\mathtt{p}$ will be performed. The local type for all of $\mathsf{q}_i \in \widetilde{\mathsf{q}}$ 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 $\downarrow\beta.(\exists\alpha.@_\alpha\,G_1{\restriction}\mathtt{p}) \circ @_\beta\,G_2{\restriction}\mathtt{p}$ and $\downarrow\beta.(\forall\alpha.@_\alpha\,G_1{\restriction}\mathsf{q}_i) \circ @_\beta\,G_2{\restriction}\mathsf{q}_i$ 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 [32]). *We say that global type G is* well-formed (WF, in the following) *if the projection $G{\restriction}\mathtt{r}$ 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 append or 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 $\pi \in \{c(y), c\langle\omega\rangle, c(\alpha), c\langle y@\omega\rangle, c(y@\omega), c \triangleleft l\}$) :*

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

and is undefined otherwise.

The previous definition suffices to define a medium process (or simply *medium*), which uses indexed names to uniformly capture 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 *of G, denoted $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, is defined as:*

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

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 $\mathtt{q}$. We omit the forwarding and value exchange when the interaction does not involve a value payload.

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:

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

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 [7] 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*

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

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* $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ *then* $\Omega; \Gamma; c_{\mathtt{p}_1}{:}\langle\!\langle G{\restriction}\mathtt{p}_1\rangle\!\rangle[\omega_1], \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G{\restriction}\mathtt{p}_n\rangle\!\rangle[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ *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* $\omega_i \neq \omega_j$.

The second characterization result, given next, is the converse of Theorem 4.11: binary typability precisely delineates the interactions that underlie well-formed multiparty protocols. We need an auxiliary relation on local types, written $\preceq^{\sqcup}_{\downarrow}$, that relates types with branching and "here" type operators, which have silent process interpretations (cf. Figure 1 and [9]). First, we have $T_1 \preceq^{\sqcup}_{\downarrow} T_2$ if there is a T' such that $T_1 \sqcup T' = T_2$ (cf. Def. 4.3). Second, we have $T_1 \preceq^{\sqcup}_{\downarrow} T_2$ 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 $T_2 = T'\{^{\omega}/\alpha\}$. (See [9] for a formal definition of $\preceq^{\sqcup}_{\downarrow}$).

▶ **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* $G{\restriction}\mathtt{p}_j \preceq^{\sqcup}_{\downarrow} T_j$ *and* $\langle\!\langle T_j\rangle\!\rangle = A_j$, *for all* $\mathtt{p}_j \in \mathsf{part}(G)$.

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 $\mathtt{p}_i \in \mathsf{part}(G)$). These dynamic guarantees can be easily obtained by combining Theorems 4.11 and 4.12 with the approach in [7].

5 Related Work

There is a rich history of works on the logical foundations of concurrency (see, e.g., [4, 27, 1, 3]), which has been extended to session-based concurrency by Wadler [47], Dal Lago and Di Giamberardino [35], and others. Medium-based analyses of multiparty sessions were developed in [7] and used in an account of multiparty sessions in an extended classical linear logic [14].

Two salient calculi with distributed features are the Ambient calculus [16], in which processes move across *ambients* (abstractions of administrative domains), and the *distributed π-calculus* (Dpi) [29], which extends the π-calculus with flat locations, local communication, and process migration. While domains in our model may be read as locations, this is just one specific interpretation; they admit various alternative readings (e.g., administrative domains, security-related levels), leveraging the partial view of the domain hierarchy. Type systems for Ambient calculi such as [15, 6] enforce security and communication-oriented properties in terms of ambient movement but do not cover issues of structured interaction, central in our work. Garralda et al. [25] integrate binary sessions in an Ambient calculus, ensuring that session protocols are undisturbed by ambient mobility. In contrast, our type system ensures that both migration and communication are safe and, for the first time in such a setting, satisfy global progress (i.e., session protocols never jeopardize migration and vice-versa).

The multiparty sessions with nested protocols of Demangeon and Honda [19] include a nesting construct that is similar to our new global type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$, which also introduces nesting. The focus in [19] is on modularity in choreographic programming; domains nor domain migration are not addressed. The nested protocols in [19] can have *local* participants and may be parameterized on data from previous actions. We conjecture that our approach can accommodate local participants in a similar way. Data parameterization can be transposed to our logical setting via dependent session types [44, 46]. Asynchrony and recursive behaviors can also be integrated by exploiting existing logical foundations [23, 45].

Balzer et al. [2] overlay a notion of world and accessibility on a system of *shared* session types to ensure deadlock-freedom. Their work differs substantially from ours: they instantiate accessibility as a partial-order, equip sessions with multiple worlds and are not conservative wrt linear logic, being closer to partial-order-based typings for deadlock-freedom [34, 37].

6 Concluding Remarks

We developed a Curry-Howard interpretation of hybrid linear logic as domain-aware session types. Present in processes and types, domain-awareness can account for scenarios where domain information is only determined at runtime. The resulting type system features strong correctness properties for well-typed processes (session fidelity, global progress, termination). Moreover, by leveraging a *parametric* accessibility relation, it rules out processes that communicate with inaccessible domains, thus going 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.

Our work opens up interesting avenues for future work. Mediums can be seen as *monitors* that enforce the specification of a domain-aware multiparty session. We plan to investigate contract-enforcing mediums building upon works such as [28, 33, 20], which study runtime monitoring in session-based systems. Our enforcement of communication across accessible domains suggests high-level similarities with information flow analyses in multiparty sessions (cf. [13, 12, 17]), but does not capture the directionality needed to model such analyses outright. It would be insightful to establish the precise relationship with such prior works.

References

1 Samson Abramsky. Computational Interpretations of Linear Logic. *Theor. Comput. Sci.*, 111(1&2):3–57, 1993. `doi:10.1016/0304-3975(93)90181-R`.

2 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.

3 Emmanuel Beffara. A Concurrent Model for Linear Logic. *Electr. Notes Theor. Comput. Sci.*, 155:147–168, 2006. `doi:10.1016/j.entcs.2005.11.055`.

4 Gianluigi Bellin and Philip J. Scott. On the pi-Calculus and Linear Logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. `doi:10.1016/0304-3975(94)00104-9`.

5 Torben Braüner and Valeria de Paiva. Intuitionistic Hybrid Logic. *J. of App. Log.*, 4:231–255, 2006.

6 Michele Bugliesi and Giuseppe Castagna. Behavioural typing for safe ambients. *Comput. Lang.*, 28(1):61–99, 2002. `doi:10.1016/S0096-0551(02)00008-5`.

7 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 with proofs at `https://sites.google.com/a/jorgeaperez.net/www/publications/medium16long.pdf`.

8 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. `doi:10.1007/978-3-642-37036-6_19`.

9 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Domain-Aware Session Types (Extended Version). *CoRR*, abs/1907.01318, 2019. `arXiv:1907.01318`.

10 Luís Caires and Frank Pfenning. Session Types as Intuitionistic Linear Propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010. `doi:10.1007/978-3-642-15375-4_16`.

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

12 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. `doi:10.4204/EPTCS.64.2`.

13 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. `doi:10.1007/978-3-642-15375-4_17`.

14 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.

15 Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the Ambient Calculus. *Inf. Comput.*, 177(2):160–194, 2002. `doi:10.1006/inco.2001.3121`.

16 Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000. `doi:10.1016/S0304-3975(99)00231-5`.

17 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Asp. Comput.*, 28(4):669–696, 2016. doi:10.1007/s00165-016-0381-3.

18 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.

19 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. doi:10.1007/978-3-642-32940-1_20.

20 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. doi:10.1007/s10703-014-0218-8.

21 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. doi:10.1007/978-3-642-39212-2_18.

22 Joëlle Despeyroux, Carlos Olarte, and Elaine Pimentel. Hybrid and Subexponential Linear Logics. *Electr. Notes Theor. Comput. Sci.*, 332:95–111, 2017.

23 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.

24 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. doi:10.1007/978-3-642-14458-5_1.

25 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. doi:10.1145/1140335.1140344.

26 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

27 Jean-Yves Girard and Yves Lafont. Linear Logic and Lazy Computation. In *TAPSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development*, pages 52–66, 1987. doi:10.1007/BFb0014972.

28 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.

29 Matthew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents. *Inf. Comput.*, 173(1):82–120, 2002. doi:10.1006/inco.2001.3089.

30 Kohei Honda. Types for Dynamic Interaction. In *CONCUR*, volume 715 of *LNCS*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

31 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP'98*, LNCS. Springer, 1998. doi:10.1007/BFb0053567.

32 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. doi:10.1145/1328438.1328472.

33 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.

34 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.

35 Ugo Dal Lago and Paolo Di Giamberardino. Soft Session Types. In *EXPRESS*, volume 64 of *EPTCS*, pages 59–73, 2011. `doi:10.4204/EPTCS.64.5`.

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

37 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.

38 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. `doi:10.1007/978-3-642-28869-2_27`.

39 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Inf. Comput.*, 239:254–302, 2014. `doi:10.1016/j.ic.2014.08.001`.

40 Jason Reed. Hybridizing a Logical Framework. *Electr. Notes Theor. Comput. Sci.*, 174(6):135–148, 2007.

41 Davide Sangiorgi. pi-Calculus, Internal Mobility, and Agent-Passing Calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996. `doi:10.1016/0304-3975(96)00075-8`.

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

43 Alex K. Simpson. *The proof theory and semantics of intuitionistic modal logic.* PhD thesis, University of Edinburgh, UK, 1994. URL: `http://hdl.handle.net/1842/407`.

44 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. `doi:10.1145/2003476.2003499`.

45 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.

46 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.

47 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. `doi:10.1017/S095679681400001X`.