


1 Domain-Aware Session Types

2 **Luís Caires** 


3 Universidade Nova de Lisboa

4 **Jorge A. Pérez** 

5 University of Groningen

6 **Frank Pfenning**

7 Carnegie Mellon University

8 **Bernardo Toninho** 

9 Universidade Nova de Lisboa

10 Abstract

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

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

25 **2012 ACM Subject Classification** Theory of computation → Process calculi; Theory of computation
26 → Type structures; Software and its engineering → Message passing

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

28 **Digital Object Identifier** 10.4230/LIPIcs.CONCUR.2019.35

29 **Related Version** Omitted definitions, proofs, and extended examples: <http://arxiv.org/abs/1907.XXXXX>.

31 **Funding** Caires and Toninho are supported by NOVA LINCS (Ref. UID/CEC/04516/2019). Pérez
32 is supported by the NWO VIDI Project No. 016.Vidi.189.046. Pfenning is supported by NSF Grant
33 No. CCF-1718267: “Enriching Session Types for Practical Concurrent Programming”.

34 1 Introduction

35 The goal of this paper is to show how existing Curry-Howard interpretations of session
36 types [9, 10] can be generalized to a *domain-aware* setting by relying on an extension of
37 linear logic with features from *hybrid logic* [42, 5]. These extended logical foundations of
38 message-passing concurrency allow us to analyze complex domain-aware concurrent systems
39 (including those governed by multiparty protocols) in a precise and principled manner.

40 Software systems typically rely on *communication* between heterogeneous services; at their
41 heart, these systems rely on message-passing protocols that combine mobility, concurrency,
42 and distribution. As distributed services are often virtualized, protocols should span diverse
43 software and hardware *domains*. These domains can have multiple interpretations, such as
44 the location where services reside, or the principals on whose behalf they act. Concurrent



© Caires et al.;

licensed under Creative Commons License CC-BY

30th International Conference on Concurrency Theory (CONCUR 2019).

Editors: Wan Fokkink and Rob van Glabbeek; Article No. 35; pp. 35:1–35:29

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

45 behavior is then increasingly *domain-aware*: a partner’s potential for interaction is influenced
 46 not only by the domains it is involved in at various protocol phases (its context), but also
 47 by *connectedness* relations among domains. Moreover, domain architectures are rarely fully
 48 specified: to aid modularity and platform independence, system participants (e.g., developers,
 49 platform vendors, service clients) often have only partial views of actual domain structures.
 50 Despite their importance in communication correctness and trustworthiness at large, the
 51 formal status of domains within *typed* models of message-passing systems remains unexplored.

52 This paper contributes to typed approaches to the analysis of domain-aware commu-
 53 nications, with a focus on *session-based concurrency*. This approach specifies the intended
 54 message-passing protocols as *session types* [29, 30, 23]. Different type theories for *binary*
 55 and *multiparty* (n -ary) protocols have been developed. In both cases, typed specifications
 56 can be conveniently coupled with π -calculus processes [36], in which so-called session chan-
 57 nels connect exactly two subsystems. Communication correctness usually results from two
 58 properties: *session fidelity* (type preservation) and *deadlock freedom* (progress). The former
 59 says that well-typed processes always evolve to well-typed processes (a safety property); the
 60 latter says that well-typed processes will never get into a stuck state (a liveness property).

61 A key motivation for this paper is the sharp contrast between (a) the growing relevance
 62 of domain-awareness in message-passing, concurrent systems and (b) the expressiveness of
 63 existing session type frameworks, binary and multiparty, which cannot adequately specify
 64 (let alone enforce) domain-related requirements. Indeed, existing session types frameworks,
 65 including those based on Curry-Howard interpretations [9, 50, 13], capture communication
 66 behavior at a level of abstraction in which even basic domain-aware assertions (e.g., “*Shipper*
 67 resides in domain *AmazonUS*”) cannot be expressed. As an unfortunate consequence, the
 68 effectiveness of the analysis techniques derived from these frameworks is rather limited.

69 To better illustrate our point, consider a common distributed design pattern: a middleware
 70 agent (*mw*) which answers requests from clients (*c1*), sometimes offloading the requests to a
 71 server (*serv*) to better manage local resource availability. In the framework of multiparty
 72 session types [31] this protocol can be represented as the global type:

$$\begin{aligned} \text{c1} \rightarrow \text{mw} : \{ \text{request}(\text{req}) . \text{mw} \rightarrow \text{c1} : \{ \text{reply}(\text{ans}) . \text{mw} \rightarrow \text{serv} : \{ \text{done} . \text{end} \} , \text{wait} . \text{mw} \rightarrow \text{serv} : \{ \text{req}(\text{data}) . \\ \text{serv} \rightarrow \text{mw} : \{ \text{reply}(\text{ans}) . \text{mw} \rightarrow \text{c1} : \{ \text{reply}(\text{ans}) . \text{end} \} \} \} \} \} \end{aligned}$$

73 The client first sends a request to the middleware, which answers back with either a *reply*
 74 message containing the answer or a *wait* message, signaling that the server will be contacted to
 75 produce the final *reply*. While this multiparty protocol captures the intended communication
 76 behavior, it does not capture that protocols for the middleware and the server often involve
 77 some form of privilege escalation or specific authentication—ensuring, e.g., that the server
 78 interaction is adequately isolated from the client, or that the escalation must precede the
 79 server interactions. These requirements simply cannot be represented in existing frameworks.

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

88 Consider the global type *Offload* $\triangleq \text{mw} \rightarrow \text{serv} : \{ \text{req}(\text{data}) . \text{serv} \rightarrow \text{mw} : \{ \text{reply}(\text{ans}) . \text{end} \} \}$
 89 in our previous example. Our framework allows us to refactor the global type above as:

$$\begin{aligned} \text{c1} \rightarrow \text{mw} : \{ \text{request}(\text{req}) . \text{mw} \rightarrow \text{c1} : \{ \text{reply}(\text{ans}) . \text{mw} \rightarrow \text{serv} : \{ \text{done} . \text{end} \} , \text{wait} . \text{mw} \rightarrow \text{serv} : \{ \text{init} . \\ \text{mw moves serv to } w_{\text{priv}} \text{ for Offload ; mw} \rightarrow \text{c1} : \{ \text{reply}(\text{ans}) . \text{end} \} \} \} \} \end{aligned}$$

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_{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 [21, 17], 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* [21, 17] we generalize the Curry-Howard interpretation of session types [9, 10] 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 [31] 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 [44] with labeled choice and explicit domain migration and communication. We write $\omega, \omega', \omega''$ to stand for a concrete domain (w, w', \dots) or a domain variable (α, α', \dots). 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, \dots , \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*

$$\begin{array}{l}
 P ::= \mathbf{0} \quad | \quad P \mid Q \quad | \quad (\nu y)P \quad | \quad x(y).P \quad | \quad x(y).P \quad | \quad !x(y).P \\
 \quad | \quad [x \leftrightarrow y] \quad | \quad x \triangleright \{l_i : P_i\}_{i \in I} \quad | \quad x \triangleleft l_i; P \\
 \quad | \quad x(y@w).P \quad | \quad x(y@w).P \quad | \quad x(w).P \quad | \quad x(\alpha).P
 \end{array}$$

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

136 The operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition) and $(\nu y)P$ (name restriction)
 137 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
 138 and proceed as P with parameter y replaced by z), and $!x(y).P$ which denotes replicated
 139 (persistent) input. The forwarding construct $[x \leftrightarrow y]$ equates x and y ; it is a primitive
 140 representation of a copycat process. The last two constructs in the second line define a
 141 labeled choice mechanism: $x \triangleright \{l_i : P_i\}_{i \in I}$ is a process that awaits some label l_j (with $j \in I$)
 142 and proceeds as P_j . Dually, the process $x \triangleleft l_i; P$ emits a label l_i and proceeds as P .

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

152 Following [43], we abbreviate $(\nu y)x\langle y \rangle$ and $(\nu y)x\langle y @ \omega \rangle$ as $\bar{x}\langle y \rangle$ and $\bar{x}\langle y @ \omega \rangle$, respectively.
 153 In $(\nu y)P$, $x(y).P$, and $x(y @ \omega).P$ the distinguished occurrence of name y is binding with
 154 scope P . Similarly for α in $x(\alpha).P$. We identify processes up to consistent renaming of bound
 155 names and variables, writing \equiv_α for this congruence. $P\{x/y\}$ denotes the capture-avoiding
 156 substitution of x for y in P . While *structural congruence* \equiv expresses standard identities on
 157 the basic structure of processes (cf. [?]), *reduction* expresses their behavior.

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

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

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

164 3 Domain-aware Session Types via Hybrid Logic

165 This section develops a new domain-aware formulation of binary session types. Our system
 166 is based on a Curry-Howard interpretation of a linear variant of so-called *hybrid logic*, and
 167 can be seen as an extension of the interpretation of [9, 10] to hybrid (linear) logic. Hybrid
 168 logic is often used as an umbrella term for a class of logics that extend the expressiveness of
 169 propositional logic by considering modal *worlds* as syntactic objects that occur in propositions.

170 As in [9, 10], propositions are interpreted as session types of communication channels,
 171 proofs as typing derivations, and proof reduction as process communication. As main
 172 novelties, here we interpret: logical worlds as *domains*; the hybrid connective $@_\omega A$ as the
 173 type of a session that *migrates* to an accessible domain ω ; and type-level quantification over
 174 worlds $\forall \alpha.A$ and $\exists \alpha.A$ as *domain communication*. We also consider a type-level operator

175 $\downarrow\alpha.A$ (read “here”) which binds the *current* domain of the session to α in A . The syntax of
 176 domain-aware session types is given in Def. 3.1, where w, w_1, \dots stand for domains drawn
 177 from \mathcal{W} , and where α, β and ω, ω' are used as in the syntax of processes.

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

$$179 \quad \begin{array}{l} A ::= \mathbf{1} \quad | \quad A \multimap B \quad | \quad A \otimes B \quad | \quad \&\{l_i : A_i\}_{i \in I} \quad | \quad \oplus\{l_i : A_i\}_{i \in I} \quad | \quad !A \\ \quad | \quad @_\omega A \quad | \quad \forall\alpha.A \quad | \quad \exists\alpha.A \quad | \quad \downarrow\alpha.A \end{array}$$

180 Types are the propositions of intuitionistic linear logic where the additives $A \& B$ and $A \oplus B$
 181 are generalized to a labelled n -ary variant. Propositions take the standard interpretation as
 182 session types, extended with hybrid logic operators [5], with worlds interpreted as domains
 183 that are explicitly subject to an *accessibility relation* (in the style of [45]) that is tracked
 184 by environment Ω . Intuitively, Ω is made up of direct accessibility hypotheses of the form
 185 $\omega_1 \prec \omega_2$, meaning that domain ω_2 is accessible from ω_1 .

186 Types are assigned to channel names; a *type assignment* $x:A[\omega]$ enforces the use of name
 187 x according to session A , *in the domain* ω . A *type environment* is a collection of type
 188 assignments. Besides the accessibility environment Ω just mentioned, our typing judgments
 189 consider two kinds of type environments: a *linear* part Δ and an *unrestricted* part Γ . They
 190 are subject to different structural properties: weakening and contraction principles hold for
 191 Γ but not for Δ . Empty environments are written as ‘ \cdot ’. We then consider two judgments:

$$192 \quad \text{(i) } \Omega \vdash \omega_1 \prec \omega_2 \quad \text{and} \quad \text{(ii) } \Omega; \Gamma; \Delta \vdash P :: z:A[\omega]$$

193 Judgment (i) states that ω_1 can directly access ω_2 under the hypotheses in Ω . We write
 194 \prec^* for the reflexive, transitive closure of \prec , and $\omega_1 \not\prec^* \omega_2$ when $\omega_1 \prec^* \omega_2$ does not hold.
 195 Judgment (ii) states that process P offers the session behavior specified by type A on
 196 channel z ; the session s resides at domain ω , under the accessibility hypotheses Ω , using
 197 unrestricted sessions in Γ and linear sessions in Δ . Note that each hypothesis in Γ and Δ is
 198 labeled with a specific domain. We omit Ω when it is clear from context.

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

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

$$213 \quad (\multimap R) \frac{\Omega; \Gamma; \Delta, y:A[\omega] \vdash P :: z:B[\omega]}{\Omega; \Gamma; \Delta \vdash z(y).P :: z:A \multimap B[\omega]} \quad (\otimes R) \frac{\Omega; \Gamma; \Delta_1 \vdash P :: y:A[\omega] \quad \Omega; \Gamma; \Delta_2 \vdash Q :: z:B[\omega]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash \bar{z}(y).(P \mid Q) :: z:A \otimes B[\omega]}$$

214 Dually, $A \otimes B$ denotes a session that outputs a session that will offer A and continue as B .
 215 To offer $z:A \otimes B$, we output a fresh name y with type A along z and proceed offering $z:B$.

35:6 Domain-Aware Session Types

216 The (cut) rule allows us to compose process P , which offers $x:A[\omega_2]$, with process Q ,
 217 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.,
 218 $\omega_1 \prec^* \omega_2$). We also require $\omega_1 \prec^* \Delta_1$: the domains mentioned in Δ_1 (the context for P)
 219 must be accessible from ω_1 , which follows from the transitive closure of the accessibility
 220 relation (\prec^*) using the intermediary domain ω_2 . As in [9, 10], composition binds the name x :

$$221 \quad (\text{cut}) \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega \vdash \omega_1 \prec^* \Delta_1 \quad \Omega; \Gamma; \Delta_1 \vdash P :: x:A[\omega_2] \quad \Omega; \Gamma; \Delta_2, x:A[\omega_2] \vdash Q :: z:C[\omega_1]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\nu x)(P \mid Q) :: z:C[\omega_1]}$$

222 Type **1** means that no further interaction will take place on the session; names of type **1**
 223 may be passed around as opaque values. $\&\{l_i : A_i\}_{i \in I}$ types a session channel that offers
 224 its partner a choice between the A_i behaviors, each uniquely identified by a label l_i . Dually,
 225 $\oplus\{l_i : A_i\}_{i \in I}$ types a session that selects some behavior A_i by emitting the corresponding
 226 label. For flexibility and consistency with merge-based projectability in multiparty session
 227 types, rules for choice and selection induce a standard notion of session subtyping [25].

228 Type $!A$ types a shared (non-linear) channel, to be used by a server for spawning an
 229 arbitrary number of new sessions (possibly none), each one conforming to type A .

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

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

$$240 \quad (\text{@R}) \frac{\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega \vdash \omega_2 \prec^* \Delta \quad \Omega; \Gamma; \Delta \vdash P :: y:A[\omega_2]}{\Omega; \Gamma; \Delta \vdash \bar{z}\langle y@_{\omega_2} \rangle.P :: z:@_{\omega_2}A[\omega_1]} \quad (\text{@L}) \frac{\Omega, \omega_2 \prec \omega_3; \Gamma; \Delta, y:A[\omega_3] \vdash P :: z:C[\omega_1]}{\Omega; \Gamma; \Delta, x:@_{\omega_3}A[\omega_2] \vdash x(y@_{\omega_3}).P :: z:C[\omega_1]}$$

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

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

$$249 \quad (\forall R) \frac{\Omega, \omega_1 \prec \alpha; \Gamma; \Delta \vdash P :: z:A[\omega_1] \quad \alpha \notin \Omega, \Gamma, \Delta, \omega_1}{\Omega; \Gamma; \Delta \vdash z(\alpha).P :: z:\forall\alpha.A[\omega_1]} \quad (\forall L) \frac{\Omega \vdash \omega_2 \prec \omega_3 \quad \Omega; \Gamma; \Delta, x:A\{\omega_3/\alpha\}[\omega_2] \vdash Q :: z:C[\omega_1]}{\Omega; \Gamma; \Delta, x:\forall\alpha.A[\omega_2] \vdash x(\omega_3).Q :: z:C[\omega_1]}$$

250 Rule ($\forall R$) states that a process seeking to offer $\forall\alpha.A[\omega_1]$ denotes a service that is located
 251 at domain ω_1 but that may refer to any fresh domain directly accessible from ω_1 in its
 252 specification (e.g. through the use of $@$). Operationally, this means that the process must be
 253 ready to receive from its client a reference to the domain being referred to in the type, which
 254 is bound to α (occurring fresh in the typing derivation). Dually, Rule ($\forall L$) indicates that a
 255 process interacting with a service of type $x:\forall\alpha.A[\omega_2]$ must make concrete the domain that

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

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

$$259 \quad (\downarrow R) \frac{\Omega; \Gamma; \Delta \vdash P :: z:A\{\omega/\alpha\}[\omega]}{\Omega; \Gamma; \Delta \vdash P :: z:\downarrow\alpha.A[\omega]} \quad (\downarrow L) \frac{\Omega; \Gamma; \Delta, x:A\{\omega/\alpha\}[\omega] \vdash P :: z:C}{\Omega; \Gamma; \Delta, x:\downarrow\alpha.A[\omega] \vdash P :: z:C}$$

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

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

270 **Discussion and Examples** Being an interpretation of *hybridized* linear logic, our domain-
 271 aware theory is *conservative* wrt the Curry-Howard interpretation of session types in [9, 10],
 272 in the following sense: the system in [9, 10] corresponds to the case where every session resides
 273 at the same domain. As in [9, 10], the sequent calculus for the underlying (hybrid) linear
 274 logic can be recovered from our typing rules by erasing processes and name assignments.

275 Conversely, a fundamental consequence of our hybrid interpretation is that it *refines* the
 276 session type structure in non-trivial ways. By requiring that communication only occurs
 277 between sessions located at the same (or accessible) domain we effectively introduce a new
 278 layer of reasoning to session type systems. To illustrate this feature, consider the following
 279 session type $WStore$, which specifies a simple interaction between a web store and its clients:

$$280 \quad WStore \triangleq \text{addCart} \multimap \&\{buy : \text{Pay}, quit : \mathbf{1}\} \quad \text{Pay} \triangleq \text{CCNum} \multimap \oplus\{ok : \text{Rcpt} \otimes \mathbf{1}, nok : \mathbf{1}\}$$

281 $WStore$ allows clients to checkout their shopping carts by emitting a *buy* message or to *quit*.
 282 In the former case, the client pays for the purchase by sending their credit card data. If
 283 a banking service (not shown) approves the transaction (via an *ok* message), a receipt is
 284 emitted. Representable in existing session type systems (e.g. [9, 50, 30]), types $WStore$ and
 285 Pay describe the intended communications but fail to capture the crucial fact that in practice
 286 the client's sensitive information should only be requested after entering a secure domain. To
 287 address this limitation, we can use type-level domain migration to *refine* $WStore$ and Pay :

$$288 \quad WStore_{\text{sec}} \triangleq \text{addCart} \multimap \&\{buy : @_{\text{sec}} \text{Pay}_{\text{bnk}}, quit : \mathbf{1}\} \\ \text{Pay}_{\text{bnk}} \triangleq \text{CCNum} \multimap \oplus\{ok : (@_{\text{bnk}} \text{Rcpt}) \otimes \mathbf{1}, nok : \mathbf{1}\}$$

289 $WStore_{\text{sec}}$ decrees that the interactions pertinent to type Pay_{bnk} should be preceded by a
 290 migration step to the trusted domain sec , which should be directly accessible from $WStore_{\text{sec}}$'s
 291 current domain. The type also specifies that the receipt must originate from a bank domain
 292 bnk (e.g., ensuring that the receipt is never produced by the store without entering bnk).
 293 When considering the interactions with a client (at domain c) that checks out their cart, we
 294 reach a state that is typed with the following judgment:

$$295 \quad c \prec \text{ws}; \cdot; x:@_{\text{sec}} \text{Pay}_{\text{bnk}}[\text{ws}] \vdash \text{Client} :: z:@_{\text{sec}} \mathbf{1}[c]$$

296 At this point, it is *impossible* for a (typed) client to interact with the behavior that is
 297 protected by the domain sec , since it is not the case that $c \prec^* \text{sec}$. That is, no judgment

$$\begin{array}{c}
\text{(whyp)} \frac{}{\Omega, \omega_1 \prec \omega_2 \vdash \omega_1 \prec \omega_2} \quad \text{(id)} \frac{}{\Omega; \Gamma; x:A[\omega] \vdash [x \leftrightarrow z] :: z:A[\omega]} \\
\text{(@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 \bar{z}(y@_{\omega_2}).P :: z:@_{\omega_2}A[\omega_1]} \quad \text{(@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]} \\
\text{(\forall 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]} \quad \text{(\forall 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(\omega_3).Q :: z:C[\omega_1]} \\
\text{(\exists 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(\omega_2).P :: z:\exists\alpha.A[\omega_1]} \quad \text{(\exists 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]} \\
\text{(\downarrow R)} \frac{\Omega; \Gamma; \Delta \vdash P :: z:A\{\omega/\alpha\}[\omega]}{\Omega; \Gamma; \Delta \vdash P :: z:\downarrow\alpha.A[\omega]} \quad \text{(\downarrow 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} \\
\text{(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 \bar{u}(y).P :: z:C[\omega_1]} \\
\text{(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 (\nu x)(P \mid Q) :: z:C[\omega_1]}
\end{array}$$

■ **Figure 1** Typing Rules (Excerpt – see [?])

298 of the form $c \prec \text{ws}; \cdot; \text{Pay}_{\text{bnk}}[\text{sec}] \vdash \text{Client}' :: z:T[c]$ is derivable. This ensures, e.g., that a
299 client cannot exploit the payment platform of the web store by accessing the trusted domain
300 in unforeseen ways. The client can only communicate in the secure domain *after* the web
301 store service has migrated accordingly, as shown by the judgment

$$302 \quad c \prec \text{ws}, \text{ws} \prec \text{sec}; \cdot; x':\text{Pay}_{\text{bnk}}[\text{sec}] \vdash \text{Client}' :: z':1[\text{sec}].$$

303
304 **Technical Results** We state the main results of type safety via type preservation (The-
305 orem 3.3) and global progress (Theorem 3.4). These results directly ensure session fidelity
306 and deadlock-freedom. Typing also ensures termination, i.e., processes do not exhibit infinite
307 reduction paths (Theorem 3.5). We note that in the presence of termination, our progress
308 result ensures that communication actions are always guaranteed to take place. Moreover, as
309 a property specific to domain-aware processes, we show *domain preservation*, i.e., processes
310 respect their domain accessibility conditions (Theorem 3.7). The formal development of
311 these results relies on a *domain-aware* labeled transition system [?], defined as a simple
312 generalization of the early labelled transition system for the session π -calculus given in [9, 10].

313 **Type Safety and Termination.** Following [9, 10], our proof of type preservation relies on
314 a simulation between reductions in the session-typed π -calculus and logical proof reductions.

315 ► **Lemma 3.2** (Domain Substitution). *Suppose $\Omega \vdash \omega_1 \prec \omega_2$. Then we have:*

- 316 ■ *If $\Omega, \omega_1 \prec \alpha, \Omega'; \Gamma; \Delta \vdash P :: z:A[\omega]$ then*
317 $\Omega, \Omega'\{\omega_2/\alpha\}; \Gamma\{\omega_2/\alpha\}; \Delta\{\omega_2/\alpha\} \vdash P\{\omega_2/\alpha\} :: z:A[\omega\{\omega_2/\alpha\}].$
- 318 ■ *$\Omega, \alpha \prec \omega_2, \Omega'; \Gamma; \Delta \vdash P :: z:A[\omega]$ then*
319 $\Omega, \Omega'\{\omega_1/\alpha\}; \Gamma\{\omega_1/\alpha\}; \Delta\{\omega_1/\alpha\} \vdash P\{\omega_1/\alpha\} :: z:A[\omega\{\omega_1/\alpha\}].$

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

321 ► **Theorem 3.3** (Type Preservation). *If $\Omega; \Gamma; \Delta \vdash P :: z:A[\omega]$ and $P \rightarrow Q$ then*
322 $\Omega; \Gamma; \Delta \vdash Q :: z:A[\omega].$

323 **Proof (Sketch).** The proof mirrors those of [9, 10, 8, 46], relying on a series of lemmas
 324 relating the result of dual process actions (via our LTS semantics) with typable parallel
 325 compositions through the (cut) rule [?]. For session type constructors of [9], the results are
 326 unchanged. For the domain-aware session type constructors, the development is identical
 327 that of [8] and [46], which deal with communication of types and data terms, respectively. ◀

328 Following [9, 10], the proof of global progress relies on a notion of a *live* process, which
 329 intuitively consists of a process that has not yet fully carried out its ascribed session behavior,
 330 and thus is a parallel composition of processes where at least one is a non-replicated process,
 331 guarded by some action. Formally, we define $live(P)$ if and only if $P \equiv (\nu \tilde{n})(\pi.Q \mid R)$, for
 332 some R , names \tilde{n} and a non-replicated guarded process $\pi.Q$.

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

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

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

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

341 **Proof (Sketch).** By adapting the *linear* logical relations given in [40, 41, 8]. For the system
 342 in §3 without quantifiers, the logical relations correspond to those in [40, 41], extended to
 343 carry over Ω . When considering quantifiers, the logical relations resemble those proposed for
 344 polymorphic session types in [8], noting that no impredicativity concerns are involved. ◀

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

349 ▶ **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
 350 well-formed then every sub-derivation in \mathcal{E} well-formed.*

351 While inaccessible domains can appear in Γ , such channels can never be used and thus
 352 can not appear in a well-typed process due to the restriction on the (copy) rule. Combining
 353 Theorems 3.3 and 3.6 we can then show that even if a session in the environment changes
 354 domains, typing ensures that such a domain will be (transitively) accessible:

355 ▶ **Theorem 3.7.** *Let (1) $\Omega; \Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: z : A[\omega]$, (2) $\Omega; \Gamma; \Delta \vdash P :: x:B[\omega'']$,
 356 and (3) $\Omega; \Gamma; \Delta', x:B[\omega'] \vdash Q :: z:A[\omega]$. If $(\nu x)(P \mid Q) \rightarrow (\nu x)(P' \mid Q')$ then: (a) $\Omega; \Gamma; \Delta \vdash$
 357 $P' :: x':B'[\omega'']$, for some x', B', ω'' ; (b) $\Omega; \Gamma, \Delta', x':B'[\omega''] \vdash Q' :: z:A[\omega]$; (c) $\omega \prec^* \omega''$.*

358 4 Domain-Aware Multiparty Session Types

359 We now shift our attention to multiparty session types [31]. We consider the standard
 360 ingredients: *global types*, *local types*, and the *projection function* that connects the two. Our
 361 global types include a new domain-aware construct, \mathfrak{p} moves \tilde{q} to ω for $G_1; G_2$; our local types
 362 exploit the hybrid session types from Def. 3.1. Rather than defining a separate type system
 363 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 $\mathbf{p}, \mathbf{q}, \mathbf{r}, \dots$; we write $\tilde{\mathbf{q}}$ to denote a finite set of participants $\mathbf{q}_1, \dots, \mathbf{q}_n$.

Besides the new domain-aware global type, our syntax of global types includes constructs from [31, 20]. 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*

$$\begin{aligned} U &::= \text{bool} \mid \text{nat} \mid \text{str} \mid \dots \mid T \\ G &::= \text{end} \mid \mathbf{p} \rightarrow \mathbf{q} : \{l_i \langle U_i \rangle . G_i\}_{i \in I} \mid \mathbf{p} \text{ moves } \tilde{\mathbf{q}} \text{ to } \omega \text{ for } G_1 ; G_2 \\ T &::= \text{end} \mid \mathbf{p} ? \{l_i \langle U_i \rangle . T_i\}_{i \in I} \mid \mathbf{p} ! \{l_i \langle U_i \rangle . T_i\}_{i \in I} \mid \forall \alpha . T \mid \exists \alpha . T \mid @_{\alpha} T \mid \downarrow \alpha . T \end{aligned}$$

The completed global type is denoted end . Given a finite I and pairwise different labels, $\mathbf{p} \rightarrow \mathbf{q} : \{l_i \langle U_i \rangle . G_i\}_{i \in I}$ specifies that by choosing label l_i , participant \mathbf{p} may send a message of type U_i to participant \mathbf{q} , and then continue as G_i . We decree $\mathbf{p} \neq \mathbf{q}$, so reflexive interactions are disallowed. The global type $\mathbf{p} \text{ moves } \tilde{\mathbf{q}} \text{ to } \omega \text{ for } G_1 ; G_2$ specifies the migration of participants $\mathbf{p}, \tilde{\mathbf{q}}$ to domain ω in order to perform the *sub-protocol* G_1 ; this migration is lead by \mathbf{p} . Subsequently, all of $\mathbf{p}, \tilde{\mathbf{q}}$ 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 $\mathbf{p} \text{ moves } \tilde{\mathbf{q}} \text{ to } \omega \text{ for } G_1 ; G_2$ 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 $\mathbf{p}, \tilde{\mathbf{q}}$) 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 $\text{part}(G)$) is defined as: $\text{part}(\text{end}) = \emptyset$, $\text{part}(\mathbf{p} \rightarrow \mathbf{q} : \{l_i \langle U_i \rangle . G_i\}_{i \in I}) = \{\mathbf{p}, \mathbf{q}\} \cup \bigcup_{i \in I} \text{part}(G_i)$, $\text{part}(\mathbf{p} \text{ moves } \tilde{\mathbf{q}} \text{ to } \omega \text{ for } G_1 ; G_2) = \{\mathbf{p}\} \cup \tilde{\mathbf{q}} \cup \text{part}(G_1) \cup \text{part}(G_2)$. We sometimes write $\mathbf{p} \in G$ to mean $\mathbf{p} \in \text{part}(G)$.*

Global types are projected onto participants so as to obtain local types. The terminated local type is end . The local type $\mathbf{p} ? \{l_i \langle U_i \rangle . T_i\}_{i \in I}$ denotes an offer of a set of labeled alternatives; the local type $\mathbf{p} ! \{l_i \langle U_i \rangle . T_i\}_{i \in I}$ 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 [20]. 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 $\text{bool} \sqcup \text{bool} = \text{bool}$ (and analogously for other base types), and*

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

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

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

► **Definition 4.5** (Merge-based Projection [20]). *Let G be a global type. The merge-based projection of G under participant \mathbf{r} , denoted $G \upharpoonright \mathbf{r}$, is defined as $\text{end} \upharpoonright \mathbf{r} = \text{end}$ and*

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

When no side condition holds, the map is undefined.

The projection for the type \mathbf{p} moves $\tilde{\mathbf{q}}$ to ω for $G_1 ; G_2$ is one of the key points in our analysis. The local type for \mathbf{p} , the leader of the migration, starts by binding the identity of its current domain (say, $\omega_{\mathbf{p}}$) to β . Then, the (fresh) domain ω is communicated, and there is a migration step to ω , which is where protocol $G_1 \upharpoonright \mathbf{p}$ will be performed. Finally, there is a migration step from ω back to $\omega_{\mathbf{p}}$; once there, the protocol $G_2 \upharpoonright \mathbf{p}$ will be performed. The local type for all of $\mathbf{q}_i \in \tilde{\mathbf{q}}$ follows accordingly: they expect ω from \mathbf{p} ; the migration from their original domains to ω (and back) is as for \mathbf{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 $\downarrow \beta. (\exists \alpha. @_\alpha G_1 \upharpoonright \mathbf{p}) \circ @_\beta G_2 \upharpoonright \mathbf{p}$ and $\downarrow \beta. (\forall \alpha. @_\alpha G_1 \upharpoonright \mathbf{q}_i) \circ @_\beta G_2 \upharpoonright \mathbf{q}_i$ 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 [31]). *We say that global type G is well-formed (WF, in the following) if the projection $G \upharpoonright \mathbf{r}$ is defined for all $\mathbf{r} \in G$.*

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 \mathbf{p} , we use $c_{\mathbf{p}}$ to denote the name along which the session behavior of \mathbf{p} will be made available. This way, if $\mathbf{p} \neq \mathbf{q}$ then $c_{\mathbf{p}} \neq c_{\mathbf{q}}$. To define mediums, we need to append or fuse sequential processes, just as Def. 4.4 fuses local types:

35:12 Domain-Aware Session Types

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

$$449 \quad \begin{aligned} c\langle y \rangle.([u \leftrightarrow y] \mid P) \circ Q &\triangleq c\langle y \rangle.([u \leftrightarrow y] \mid (P \circ Q)) & \mathbf{0} \circ Q &\triangleq Q \\ c \triangleright \{l_i : P_i\}_{i \in I} \circ Q &\triangleq c \triangleright \{l_i : (P_i \circ Q)\}_{i \in I} & (\pi.P) \circ Q &\triangleq \pi.(P \circ Q) \end{aligned}$$

450 and is undefined otherwise.

451 The previous definition suffices to define a medium process (or simply *medium*), which uses
 452 indexed names to uniformly capture the behavior of a global type:

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

$$455 \quad \left\{ \begin{array}{ll} \mathbf{0} & \text{if } G = \text{end} \\ c_p \triangleright \{l_i : c_p(u).c_q \triangleleft l_i; \overline{c_q}(v).([u \leftrightarrow v] \mid M^{\tilde{\omega}}[[G_i]](\tilde{c}))\}_{i \in I} & \text{if } G = p \rightarrow q: \{l_i \langle U_i \rangle . G_i\}_{i \in I} \\ c_p(\alpha).c_{q_1} \langle \alpha \rangle. \dots .c_{q_n} \langle \alpha \rangle. & \text{if } G = p \text{ moves } q_1, \dots, q_n \text{ to } w \text{ for } G_1; G_2 \\ c_p(y_p @ \alpha).c_{q_1}(y_{q_1} @ \alpha). \dots .c_{q_n}(y_{q_n} @ \alpha). & \\ M^{\tilde{\omega}}\{\alpha/\omega_p, \dots, \alpha/\omega_{q_n}\}[[G_1]](\tilde{y}) \circ & \\ (y_p(m_p @ \omega_p).y_{q_1}(m_{q_1} @ \omega_{q_1}). \dots .y_{q_n}(m_{q_n} @ \omega_{q_n}). & \\ M^{\tilde{\omega}}[[G_2]](\tilde{m})) & \end{array} \right.$$

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

457 The medium for $G = p \rightarrow q: \{l_i \langle U_i \rangle . G_i\}_{i \in I}$ exploits four prefixes to mediate in the
 458 interaction between the implementations of p and q : the first two prefixes (on name c_p)
 459 capture the label selected by p and the subsequently received value; the third and fourth
 460 prefixes (on name c_q) propagate the choice and forward the value sent by p to q . We omit
 461 the forwarding and value exchange when the interaction does not involve a value payload.

462 The medium for $G = p \text{ moves } q_1, \dots, q_n \text{ to } w \text{ for } G_1; G_2$ showcases the expressivity and
 463 convenience of our domain-aware process framework. In this case, the medium's behavior
 464 takes place through the following steps: First, $M^{\tilde{\omega}}[[G]](\tilde{c})$ inputs a domain identifier (say, ω)
 465 from p which is forwarded to q_1, \dots, q_n , the other participants of G_1 . Secondly, the roles
 466 p, q_1, \dots, q_n migrate from their domains $\omega_p, \omega_{q_1}, \dots, \omega_{q_n}$ to ω . At this point, the medium
 467 for G_1 can execute, keeping track the current domain ω for all participants. Finally, the
 468 participants of G_1 migrate back to their original domains and the medium for G_2 executes.

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

$$\begin{aligned} c_{c1} \triangleright \{ & \text{request} : c_{c1}(r).c_{mw} \triangleleft \text{request}; \overline{c_{mw}}(v).([r \leftrightarrow v] \mid \\ & c_{mw} \triangleright \{ \text{reply} : c_{mw}(a).c_{c1} \triangleleft \text{reply}; \overline{c_{c1}}(n).([a \leftrightarrow n] \mid c_{mw} \triangleright \{ \text{done} : c_{serv} \triangleleft \text{done}; \mathbf{0} \}), \\ & \text{wait} : c_{c1} \triangleleft \text{wait}; c_{mw} \triangleright \{ \text{init} : c_{serv} \triangleleft \text{init}; c_{mw}(w_{priv}).c_{serv}(w_{priv}). \\ & c_{mw}(y_{mw} @ w_{priv}).c_{serv}(y_{serv} @ w_{priv}).M^{w_{priv}}[[\text{Offload}]](y_{mw}, y_{serv}) \circ \\ & (y_{mw}(z_{mw} @ w_{mw}).y_{serv}(z_{serv} @ w_{serv}). \\ & z_{mw} \triangleright \{ \text{reply} : z_{mw}(a).c_{c1} \triangleleft \text{reply}; \overline{c_{c1}}(n).([a \leftrightarrow n] \mid \mathbf{0}) \} \} \} \} \end{aligned}$$

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

473 **Characterization Results** We state results that offer a sound and complete account of the
 474 relationship between: (i) a global type G (and its local types), (ii) its medium process
 475 $M^{\tilde{\omega}}[[G]](\tilde{c})$, and (iii) process implementations for the participants $\{p_1, \dots, p_n\}$ of G . In a
 476 nutshell, these results say that the typeful composition of $M^{\tilde{\omega}}[[G]](\tilde{c})$ with processes for each

477 p_1, \dots, p_n (well-typed in the system of §3) performs the intended global type. Crucially, these
 478 processes reside in distinct domains and can be independently developed, guided by their local
 479 type—they need not know about the medium’s existence or structure. The results generalize
 480 those in [7] to the domain-aware setting. Given a global type G with $\text{part}(G) = \{p_1, \dots, p_n\}$,
 481 below we write $\text{npart}(G)$ to denote the set of indexed names $\{c_{p_1}, \dots, c_{p_n}\}$. We define:

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

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

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

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

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

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

496 ► **Theorem 4.11 (Global Types \rightarrow Typed Mediums).** *If G is WF with $\text{part}(G) = \{p_1, \dots, p_n\}$*
 497 *then $\Omega; \Gamma; c_{p_1} : \langle\langle G \upharpoonright p_1 \rangle\rangle[\omega_1], \dots, c_{p_n} : \langle\langle G \upharpoonright p_n \rangle\rangle[\omega_n] \vdash M^{\tilde{\omega}}[G](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ is a compositional*
 498 *typing, for some Ω, Γ , with $\tilde{\omega} = \omega_1, \dots, \omega_n$. We assume that $\omega_i \prec \omega_m$ for all $i \in \{1, \dots, n\}$*
 499 *(the medium’s domain is accessible by all), and that $i \neq j$ implies $\omega_i \neq \omega_j$.*

500 The second characterization result, given next, is the converse of Theorem 4.11: binary
 501 typability precisely delineates the interactions that underlie well-formed multiparty protocols.
 502 We need an auxiliary relation on local types, written $\preceq_{\downarrow}^{\perp}$, that relates types with branching
 503 and “here” type operators, which have silent process interpretations (cf. Figure 1 and [?]).
 504 First, we have $T_1 \preceq_{\downarrow}^{\perp} T_2$ if there is a T' such that $T_1 \sqcup T' = T_2$ (cf. Def. 4.3). Second,
 505 we have $T_1 \preceq_{\downarrow}^{\perp} T_2$ if (i) $T_1 = T'$ and $T_2 = \downarrow \alpha . T'$ and α does not occur in T' ; but also if
 506 (ii) $T_1 = \downarrow \alpha . T'$ and $T_2 = T'\{\omega/\alpha\}$. (See [?] for a formal definition of $\preceq_{\downarrow}^{\perp}$).

507 ► **Theorem 4.12 (Well-Typed Mediums \rightarrow Global Types).** *Let G be a global type (cf. Def. 4.1).*
 508 *If $\Omega; \Gamma; c_{p_1} : A_1[\omega_1], \dots, c_{p_n} : A_n[\omega_n] \vdash M^{\tilde{\omega}}[G](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ is a compositional typing then*
 509 *$\exists T_1, \dots, T_n$ such that $G \upharpoonright p_j \preceq_{\downarrow}^{\perp} T_j$ and $\langle\langle T_j \rangle\rangle = A_j$, for all $p_j \in \text{part}(G)$.*

510 The above theorems offer a *static guarantee* that connects multiparty protocols and well-typed
 511 processes. They can be used to establish also *dynamic guarantees* relating the behavior
 512 of a global type G and that of its associated set of *multiparty systems* (i.e., the typeful
 513 composition of $M^{\tilde{\omega}}[G](\tilde{c})$ with processes for each of $p_i \in \text{part}(G)$). These dynamic guarantees
 514 can be easily obtained by combining Theorems 4.11 and 4.12 with the approach in [7].

515 **5 Related Work**

516 There is a rich history of works on the logical foundations of concurrency (see, e.g., [4, 26, 1, 3]),
 517 which has been extended to session-based concurrency by Wadler [50], Dal Lago and Di

518 Giamberardino [35], and others. Medium-based analyses of multiparty sessions were developed
 519 in [7] and used in an account of multiparty sessions in an extended classical linear logic [13].

520 Two salient calculi with distributed features are the Ambient calculus [15], in which
 521 processes move across *ambients* (abstractions of administrative domains), and the *distributed*
 522 π -calculus (DP1) [28], which extends the π -calculus with flat locations, local communication,
 523 and process migration. While domains in our model may be read as locations, this is just one
 524 specific interpretation; they admit various alternative readings (e.g., administrative domains,
 525 security-related levels), leveraging the partial view of the domain hierarchy. Type systems
 526 for Ambient calculi such as [14, 6] enforce security and communication-oriented properties in
 527 terms of ambient movement but do not cover issues of structured interaction, central in our
 528 work. Garralda et al. [24] integrate binary sessions in an Ambient calculus, ensuring that
 529 session protocols are undisturbed by ambient mobility. In contrast, our type system ensures
 530 that both migration and communication are safe and, for the first time in such a setting,
 531 satisfy global progress (i.e., session protocols never jeopardize migration and vice-versa).

532 The multiparty sessions with nested protocols of Demangeon and Honda [18] include
 533 a nesting construct that is similar to our new global type \mathbf{p} moves \tilde{q} to w for $G_1 ; G_2$, which
 534 also introduces nesting. The focus in [18] is on modularity in choreographic programming;
 535 domains nor domain migration are not addressed. The nested protocols in [18] can have *local*
 536 participants and may be parameterized on data from previous actions. We conjecture that
 537 our approach can accommodate local participants in a similar way. Data parameterization
 538 can be transposed to our logical setting via dependent session types [46, 49]. Asynchrony and
 539 recursive behaviors can also be integrated by exploiting existing logical foundations [22, 48].

540 Balzer et al. [2] overlay a notion of world and accessibility on a system of *shared* session
 541 types to ensure deadlock-freedom. Their work differs substantially from ours: they instantiate
 542 accessibility as a partial-order, equip sessions with multiple worlds and are not conservative
 543 wrt linear logic, being closer to partial-order-based typings for deadlock-freedom [34, 39].

544 **6** Concluding Remarks

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

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

557 Our work opens up interesting avenues for future work. Mediums can be seen as *monitors*
 558 that enforce the specification of a domain-aware multiparty session. We plan to investigate
 559 contract-enforcing mediums building upon works such as [27, 32, 19], which study runtime
 560 monitoring in session-based systems. Our enforcement of communication across accessible
 561 domains suggests high-level similarities with information flow analyses in multiparty sessions
 562 (cf. [12, 11, 16]), but does not capture the directionality needed to model such analyses
 563 outright. It would be insightful to establish the precise relationship with such prior works.

564 — **References** —

- 565 **1** Samson Abramsky. Computational interpretations of linear logic. *Theor. Comput. Sci.*,
566 111(1&2):3–57, 1993. URL: [https://doi.org/10.1016/0304-3975\(93\)90181-R](https://doi.org/10.1016/0304-3975(93)90181-R), doi:10.
567 1016/0304-3975(93)90181-R.
- 568 **2** Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for
569 shared session types. In *Programming Languages and Systems - 28th European Symposium on*
570 *Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and*
571 *Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*,
572 pages 611–639, 2019.
- 573 **3** Emmanuel Beffara. A concurrent model for linear logic. *Electr. Notes Theor. Comput. Sci.*,
574 155:147–168, 2006. URL: [https://doi.org/10.1016/j.](https://doi.org/10.1016/j.entcs.2005.11.055)
575 [entcs.2005.11.055](https://doi.org/10.1016/j.entcs.2005.11.055), doi:10.1016/j.
576 [entcs.2005.11.055](https://doi.org/10.1016/j.entcs.2005.11.055).
- 577 **4** Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *Theor. Comput. Sci.*,
578 135(1):11–65, 1994. URL: [https://doi.org/10.1016/0304-3975\(94\)00104-9](https://doi.org/10.1016/0304-3975(94)00104-9), doi:10.1016/
579 0304-3975(94)00104-9.
- 580 **5** Torben Braüner and Valeria de Paiva. Intuitionistic hybrid logic. *J. of App. Log.*, 4:231–255,
581 2006.
- 582 **6** Michele Bugliesi and Giuseppe Castagna. Behavioural typing for safe ambients. *Comput.*
583 *Lang.*, 28(1):61–99, 2002.
- 584 **7** Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory,
585 and beyond. In *Formal Techniques for Distributed Objects, Components, and Systems - 36th*
586 *IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International*
587 *Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete,*
588 *Greece, June 6-9, 2016, Proceedings*, pages 74–95, 2016. Extended version with proofs at
589 <https://sites.google.com/a/jorgeaperez.net/www/publications/medium16long.pdf>.
- 590 **8** Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism
591 and parametricity in session-based communication. In *ESOP*, volume 7792 of *LNCS*. Springer,
592 2013.
- 593 **9** Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*,
594 volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
- 595 **10** Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session
596 types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. URL: [https:](https://doi.org/10.1017/S0960129514000218)
597 [//doi.org/10.1017/S0960129514000218](https://doi.org/10.1017/S0960129514000218), doi:10.1017/S0960129514000218.
- 598 **11** Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Information flow safety
599 in multiparty sessions. In Bas Luttik and Frank Valencia, editors, *EXPRESS*, volume 64 of
600 *EPTCS*, pages 16–30, 2011.
- 601 **12** Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session
602 types for access and information flow control. In *CONCUR*, volume 6269 of *LNCS*, pages
603 237–252. Springer, 2010.
- 604 **13** Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler.
605 Coherence generalises duality: A logical explanation of multiparty session types. In *27th*
606 *International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec*
607 *City, Canada*, pages 33:1–33:15, 2016.
- 608 **14** Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the ambient calculus. *Inf.*
609 *Comput.*, 177(2):160–194, 2002.
- 610 **15** Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213,
611 2000.
- 612 **16** Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-adaptation
613 and secure information flow in multiparty communications. *Formal Asp. Comput.*,
614 28(4):669–696, 2016. URL: <https://doi.org/10.1007/s00165-016-0381-3>, doi:10.1007/
s00165-016-0381-3.

- 615 **17** Kaustuv Chaudhuri, Carlos Olarte, Elaine Pimentel, and Joëlle Despeyroux. Hybrid Linear
616 Logic, revisited. *Mathematical Structures in Computer Science*, 2019. URL: [https://hal.
617 inria.fr/hal-01968154](https://hal.inria.fr/hal-01968154).
- 618 **18** Romain Demangeon and Kohei Honda. Nested protocols in session types. In *CONCUR
619 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon
620 Tyne, UK, September 4-7, 2012. Proceedings*, pages 272–286, 2012. URL: [https://doi.org/
621 10.1007/978-3-642-32940-1_20](https://doi.org/10.1007/978-3-642-32940-1_20), doi:10.1007/978-3-642-32940-1_20.
- 622 **19** Romain Demangeon, Kohei Honda, Raymond Hu, Romyana Neykova, and Nobuko Yoshida.
623 Practical interruptible conversations: distributed dynamic verification with multiparty session
624 types and python. *Formal Methods in System Design*, 46(3):197–225, 2015. URL: [https:
625 //doi.org/10.1007/s10703-014-0218-8](https://doi.org/10.1007/s10703-014-0218-8), doi:10.1007/s10703-014-0218-8.
- 626 **20** Pierre-Malo Deniérou and Nobuko Yoshida. Multiparty compatibility in communicating
627 automata: Characterisation and synthesis of global session types. In Fedor V. Fomin,
628 Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages,
629 and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12,
630 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages
631 174–186. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-39212-2_18, doi:
632 10.1007/978-3-642-39212-2_18.
- 633 **21** Joëlle Despeyroux, Carlos Olarte, and Elaine Pimentel. Hybrid and subexponential linear
634 logics. *Electr. Notes Theor. Comput. Sci.*, 332:95–111, 2017.
- 635 **22** Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut reduction in linear
636 logic as asynchronous session-typed communication. In *Computer Science Logic (CSL'12) -
637 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September
638 3-6, 2012, Fontainebleau, France*, pages 228–242, 2012.
- 639 **23** Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview.
640 In *WS-FM 2009*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010.
- 641 **24** Pablo Garralda, Adriana B. Compagnoni, and Mariangiola Dezani-Ciancaglini. Bass: boxed
642 ambients with safe sessions. In Annalisa Bossi and Michael J. Maher, editors, *PPDP*, pages
643 61–72. ACM, 2006.
- 644 **25** Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*,
645 42(2-3):191–225, 2005. URL: <https://doi.org/10.1007/s00236-005-0177-z>, doi:10.1007/
646 s00236-005-0177-z.
- 647 **26** Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In *TAPSOFT'87: Pro-
648 ceedings of the International Joint Conference on Theory and Practice of Software Development*,
649 pages 52–66, 1987. URL: <https://doi.org/10.1007/BFb0014972>, doi:10.1007/BFb0014972.
- 650 **27** Hannah Gommerstadt, Limin Jia, and Frank Pfenning. Session-typed concurrent contracts.
651 In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP
652 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software,
653 ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 771–798, 2018.
- 654 **28** Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf.
655 Comput.*, 173(1):82–120, 2002.
- 656 **29** Kohei Honda. Types for dynamic interaction. In *CONCUR*, volume 715 of *LNCS*, pages
657 509–523. Springer, 1993.
- 658 **30** Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type
659 discipline for structured communication-based programming. In *ESOP'98*, LNCS. Springer,
660 1998.
- 661 **31** Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types.
662 In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-
663 SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco,
664 California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008. URL: [https://doi.org/
665 10.1145/1328438.1328472](https://doi.org/10.1145/1328438.1328472), doi:10.1145/1328438.1328472.

- 666 32 Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment
667 for higher-order session types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT*
668 *Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA,*
669 *January 20 - 22, 2016*, pages 582–594, 2016.
- 670 33 Limin Jia and David Walker. Modal proofs as distributed programs (extended abstract). In
671 *ESOP*, volume 2986 of *LNCS*, pages 219–233. Springer, 2004.
- 672 34 Naoki Kobayashi. A new type system for deadlock-free processes. In *CONCUR 2006 -*
673 *Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August*
674 *27-30, 2006, Proceedings*, pages 233–247, 2006.
- 675 35 Ugo Dal Lago and Paolo Di Giamberardino. Soft session types. In *EXPRESS*, volume 64 of
676 *EPTCS*, pages 59–73, 2011.
- 677 36 Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, part I/II.
678 *Inf. Comput.*, 100(1):1–77, 1992.
- 679 37 Tom Murphy. *Modal Types for Mobile Code*. PhD thesis, Carnegie Mellon University, 2008.
- 680 38 Tom Murphy, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda
681 calculus for distributed computing. In *LICS*, pages 286–295. IEEE Computer Society, 2004.
- 682 39 Luca Padovani. Deadlock and lock freedom in the linear π -calculus. In *Joint Meeting of the*
683 *Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-*
684 *Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14,*
685 *Vienna, Austria, July 14 - 18, 2014*, pages 72:1–72:10, 2014.
- 686 40 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations
687 for session-based concurrency. In *ESOP*, volume 7211 of *LNCS*, pages 539–558. Springer, 2012.
- 688 41 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations
689 and observational equivalences for session-based concurrency. *Inf. Comput.*, 239:254–302, 2014.
690 URL: <https://doi.org/10.1016/j.ic.2014.08.001>, doi:10.1016/j.ic.2014.08.001.
- 691 42 Jason Reed. Hybridizing a logical framework. *Electr. Notes Theor. Comput. Sci.*, 174(6):135–
692 148, 2007.
- 693 43 Davide Sangiorgi. π -calculus, internal mobility, and agent-passing calculi. *Theor. Comput.*
694 *Sci.*, 167(1&2):235–274, 1996.
- 695 44 Davide Sangiorgi and David Walker. *The π -calculus: A Theory of Mobile Processes*. Cambridge
696 University Press, New York, NY, USA, 2001.
- 697 45 Alex K. Simpson. *The proof theory and semantics of intuitionistic modal logic*. PhD thesis,
698 University of Edinburgh, UK, 1994. URL: <http://hdl.handle.net/1842/407>.
- 699 46 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic
700 linear type theory. In *Proc. of PPDP '11*, pages 161–172, New York, NY, USA, 2011. ACM.
701 URL: <http://doi.acm.org/10.1145/2003476.2003499>, doi:[http://doi.acm.org/10.1145/](http://doi.acm.org/10.1145/2003476.2003499)
702 [2003476.2003499](http://doi.acm.org/10.1145/2003476.2003499).
- 703 47 Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In
704 Lars Birkedal, editor, *FoSSaCS*, volume 7213 of *LNCS*, pages 346–360. Springer, 2012.
- 705 48 Bernardo Toninho, Luís Caires, and Frank Pfenning. Corecursion and non-divergence in
706 session-typed processes. In *Trustworthy Global Computing - 9th International Symposium,*
707 *TGC 2014, Rome, Italy, September 5-6, 2014. Revised Selected Papers*, pages 159–175, 2014.
- 708 49 Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In *Foundations*
709 *of Software Science and Computation Structures - 21st International Conference, FOSSACS*
710 *2018, Held as Part of the European Joint Conferences on Theory and Practice of Software,*
711 *ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 128–145, 2018.
- 712 50 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014.