Session Types as Intuitionistic Linear Propositions

Luís Caires[1] and Frank Pfenning[2]

[1] CITI and Departamento de Informática, FCT, Universidade Nova de Lisboa
[2] Department of Computer Science, Carnegie Mellon University

Abstract. Several type disciplines for π-calculi have been proposed in which linearity plays a key role, even if their precise relationship with pure linear logic is still not well understood. In this paper, we introduce a type system for the π-calculus that exactly corresponds to the standard sequent calculus proof system for dual intuitionistic linear logic. Our type system is based on a new interpretation of linear propositions as session types, and provides the first purely logical account of all (both shared and linear) features of session types. We show that our type discipline is useful from a programming perspective, and ensures session fidelity, absence of deadlocks, and a tight operational correspondence between π-calculus reductions and cut elimination steps.

1 Introduction

Linear logic has been intensively explored in the analysis of π-calculus models for communicating and mobile system, given its essential ability to deal with resources, effects, and non-interference. The fundamental way it provides for analyzing notions of sharing versus uniqueness, captured by the exponential "!", seems to have been a source of inspiration for Milner when introducing replication in the π-calculus [22]. Following the early works of Abramsky [1], several authors have exploited variants of π-calculi to express proof reductions (*e.g.*, [5]) or game semantics (*e.g.*, [19]) in systems of linear logic. In the field of concurrency, many research directions have also drawn inspiration from linear logic for developing type-theoretic analyses of mobile processes, motivated by the works of Kobayashi, Pierce, and Turner [21]; a similar influence is already noticeable in the first publications by Honda on session types [16]. Many expressive type disciplines for π-calculi in which linearity frequently plays a key role have been proposed since then (*e.g.*, [20,18,26,15]). However, linearity has been usually employed in such systems in indirect ways, exploiting the fine grained type context management techniques it provides, or the assignment of usage multiplicities to channels [21], rather than the deeper type-theoretic significance of linear logical operators.

In this paper we present a type system for the π-calculus that exactly corresponds to the standard sequent calculus proof system for dual intuitionistic linear logic. The key to our correspondence is a new, perhaps surprising, interpretation of intuitionistic linear logic formulas as a form of session types [16,18], in which the programming language is a session-typed π-calculus, and the type structure consists precisely of the connectives of intuitionistic linear logic, retaining their standard proof-theoretic interpretation.

In session-based concurrency, processes communicate through so-called session channels, connecting exactly two subsystems, and communication is disciplined by session protocols so that actions always occur in dual pairs: when one partner sends, the

P. Gastin and F. Laroussinie (Eds.): CONCUR 2010, LNCS 6269, pp. 222–236, 2010.

other receives; when one partner offers a selection, the other chooses; when a session terminates, no further interaction may occur. New sessions may be dynamically created by invocation of shared servers. Such a model exhibits concurrency in the sense that several sessions, not necessarily causally related, may be executing simultaneously, although races in unshared resources are forbidden; in fact this is the common situation in disciplined concurrent programming idioms. Mobility is also present, since both session and server names may be passed around (delegated) in communications. Session types have been introduced to discipline interactions in session-based concurrency, an important paradigm in communication-centric programming (see [11]).

It turns out that the connectives of intuitionistic linear logic suffice to express all the essential features of finite session disciplines. While in the linear λ-calculus types are assigned to terms (denoting functions and values), in our interpretation types are assigned to names (denoting communication channels) and describe their session protocol. The essence of our interpretation may already be found in the interpretation of the linear logic multiplicatives as behavioral prefix operators. Traditionally, an object of type $A \multimap B$ denotes a linear function that given an object of type A returns an object of type B [14]. In our interpretation, an object of type $A \multimap B$ denotes a session x that first inputs a session channel of type A, and then behaves as B, where B specifies again an interactive behavior, rather than a closed value. Linearity of $\multimap$ is essential, otherwise the behavior of the input session after communication could not be ensured. An object of type $A \otimes B$ denotes a session that first sends a session channel of type A and afterwards behaves as B. But notice that objects of type $A \otimes B$ really consist of two objects: the sent session of type A and the continuation session, of type B. These two sessions are separate and non-interfering, as enforced by the canonical semantics of the linear multiplicative conjunction ($\otimes$). Our interpretation of $A \otimes B$ appears asymmetric, in the sense that, of course, a channel of type $A \otimes B$ is in general not typable by $B \otimes A$. In fact, the symmetry captured by the proof of $A \otimes B \vdash B \otimes A$ is realized by an appropriately typed process that coerces any session of type $A \otimes B$ to a session of type $B \otimes A$. The other linear constructors are also given compatible interpretations, in particular, the $!A$ type is naturally interpreted as a type of a shared server for sessions of type A, and additive product and sum, to branch and choice session type operators. We thus obtain the first purely logical account of both shared and linear features of session types.

We briefly summarize the contributions of the paper. We describe a system of session types for the π-calculus (Section 3) that corresponds to the sequent calculus for dual intuitionistic linear logic DILL (Section 4). The correspondence is bidirectional and tight, in the sense that (a) any π-calculus computation can be simulated by proof reductions on typing derivations (Theorem 5.3), thus establishing a strong form of subject reduction (Theorem 5.6), and (b) that any proof reduction or conversion corresponds either to a computation step or to a process equivalence on the π-calculus side (Theorems 5.4 and 5.5). An intrinsic consequence of the logical typing is a global progress property, that ensures the absence of deadlock for systems with an arbitrary number of open sessions (Theorem 5.8). Finally, we illustrate the expressiveness of our system (Section 6) with some examples and discussion.

2 Process Model

We briefly introduce the syntax and operational semantics of the process model: the synchronous π-calculus (see [24]) extended with (binary) guarded choice.

Definition 2.1 (Processes). *Given an infinite set Λ of* names (x, y, z, u, v)*, the set of* processes (P, Q, R) *is defined by*

$$\begin{array}{llllll} P ::= \mathbf{0} \mid P \mid Q \mid (\boldsymbol{\nu} y)P & \mid x\langle y\rangle.P & \mid x(y).P & \mid\ !x(y).P \\ & \mid x.\mathtt{inl}; P & \mid x.\mathtt{inr}; P & \mid x.\mathtt{case}(P, Q) \end{array}$$

The operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition), and $(\boldsymbol{\nu} y)P$ (name restriction) comprise the static fragment of any π-calculus. We then have $x\langle y\rangle.P$ (send y on x and proceeds as P), $x(y).P$ (receive a name z on x and proceed as P with the input parameter y replaced by z), and $!x(y).P$ which denotes replicated (or persistent) input. The remaining three operators define a minimal labeled choice mechanism, comparable to the n-ary branching constructs found in standard session π-calculi (see eg., [18]). For the sake of minimality and without loss of generality we restrict our model to binary choice. In restriction $(\boldsymbol{\nu} y)P$ and input $x(y).P$ the distinguished occurrence of the name y is binding, with scope the process P. For any process P, we denote the set of *free names* of P by *fn*(P). A process is *closed* if it does not contain free occurrences of names. We identify process up to consistent renaming of bound names, writing $\equiv_\alpha$ for this congruence. We write $P\{x/y\}$ for the process obtained from P by capture avoiding substitution of x for y in P. Structural congruence expresses basic identities on the structure of processes, while reduction expresses the behavior of processes.

Definition 2.2. Structural congruence *$(P \equiv Q)$, is the least congruence relation on processes such that*

$$\begin{array}{llll} P \mid \mathbf{0} \equiv P & (S\mathbf{0}) & P \equiv_\alpha Q \Rightarrow P \equiv Q & (S\alpha) \\ P \mid Q \equiv Q \mid P & (S|C) & P \mid (Q \mid R) \equiv (P \mid Q) \mid R & (S|A) \\ (\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0} & (S\boldsymbol{\nu}\mathbf{0}) & x \notin fn(P) \Rightarrow P \mid (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \mid Q) & (S\boldsymbol{\nu}|) \\ (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P & (S\boldsymbol{\nu}\boldsymbol{\nu}) & & \end{array}$$

Definition 2.3. Reduction *$(P \to Q)$, is the binary relation on processes defined by:*

$$\begin{array}{ll} x\langle y\rangle.Q \mid x(z).P \to Q \mid P\{y/z\} & (RC) \\ x\langle y\rangle.Q \mid !x(z).P \to Q \mid P\{y/z\} \mid !x(z).P & (R!) \\ x.\mathtt{inl}; P \mid x.\mathtt{case}(Q, R) \to P \mid Q & (RL) \\ x.\mathtt{inr}; P \mid x.\mathtt{case}(Q, R) \to P \mid R & (RR) \\ Q \to Q' \Rightarrow P \mid Q \to P \mid Q' & (R|) \\ P \to Q \Rightarrow (\boldsymbol{\nu} y)P \to (\boldsymbol{\nu} y)Q & (R\boldsymbol{\nu}) \\ P \equiv P', P' \to Q', Q' \equiv Q \Rightarrow P \to Q & (R\equiv) \end{array}$$

Notice that reduction is closed (by definition) under structural congruence. Reduction specifies the computations a process performs on its own. To characterize the interactions a process may perform with its environment, we introduce a labeled transition system; the standard early transition system for the π-calculus [24] extended with appropriate labels and transition rules for the choice constructs. A transition $P \xrightarrow{\alpha} Q$

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

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

$$x(y).P \xrightarrow{x(z)} P\{z/y\}\ \text{(in)} \qquad !x(y).P \xrightarrow{x(z)} P\{z/y\} \mid !x(y).P\ \text{(rep)} \qquad x.\texttt{inl};P \xrightarrow{\overline{x.\texttt{inl}}} P\ \text{(lout)}$$

$$x.\texttt{inr};P \xrightarrow{\overline{x.\texttt{inr}}} P\ \text{(rout)} \qquad x.\mathsf{case}(P,Q) \xrightarrow{x.\texttt{inl}} P\ \text{(lin)} \qquad x.\mathsf{case}(P,Q) \xrightarrow{x.\texttt{inr}} Q\ \text{(rin)}$$

Fig. 1. π-calculus Labeled Transition System

denotes that process P may evolve to process Q by performing the action represented by the label α. Transition labels are given by

$$\alpha ::= \overline{x\langle y\rangle} \mid x(y) \mid \overline{(\nu y)x\langle y\rangle} \mid x.\mathsf{inl} \mid x.\mathsf{inr} \mid \overline{x.\mathsf{inl}} \mid \overline{x.\mathsf{inr}}$$

Actions are input $x(y)$, the left/right offers $x.\texttt{inl}$ and $x.\texttt{inr}$, and their matching co-actions, respectively the output $\overline{x\langle y\rangle}$ and bound output $\overline{(\boldsymbol{\nu} y)x\langle y\rangle}$ actions, and the left/right selections $\overline{x.\texttt{inl}}$ and $\overline{x.\texttt{inr}}$. The bound output $\overline{(\boldsymbol{\nu} y)x\langle y\rangle}$ denotes extrusion of a fresh name y along (channel) x. Internal action is denoted by $\boldsymbol{\tau}$, in general an action α ($\overline{\alpha}$) requires a matching $\overline{\alpha}$ (α) in the environment to enable progress, as specified by the transition rules. For a label α, we define the sets *fn*(α) and *bn*(α) of free and bound names, respectively, as usual. We denote by $s(\alpha)$ the subject of α (*e.g.*, x in $x\langle y\rangle$).

Definition 2.4 (Labeled Transition System). *The relation* labeled transition *($P \xrightarrow{\alpha} Q$) is defined by the rules in Figure 1, subject to the side conditions: in rule* (res)*, we require $y \notin fn(\alpha)$; in rule* (par)*, we require $bn(\alpha) \cap fn(R) = \emptyset$; in rule* (close)*, we require $y \notin fn(Q)$. We omit the symmetric versions of rules* (par)*,* (com)*, and* (close)*.*

We recall some basic facts about reduction, structural congruence, and labeled transition, namely: closure of labeled transitions under structural congruence, and coincidence of $\boldsymbol{\tau}$-labeled transition and reduction [24]: (1) if $P \equiv\xrightarrow{\alpha} Q$, then $P \xrightarrow{\alpha}\equiv Q$, and (2) $P \rightarrow Q$ if and only if $P \xrightarrow{\boldsymbol{\tau}}\equiv Q$. We write $\rho_1\rho_2$ for relation composition (e.g, $\xrightarrow{\boldsymbol{\tau}}\equiv$).

3 Type System

We first describe our type structure, which coincides with intuitionistic linear logic [14,3], omitting atomic formulas and the additive constants $\top$ and $\mathbf{0}$.

Definition 3.1 (Types). *Types (A, B, C) are given by*

$$A, B ::= \mathbf{1} \mid !A \mid A \otimes B \mid A \multimap B \mid A \oplus B \mid A \mathbin{\&} B$$

Types are assigned to (channel) names, and may be conveniently interpreted as a form of session types; an assignment $x{:}A$ enforces that the process will use x according to the discipline A. $A \otimes B$ is the type of a session channel that first performs an output

$$\frac{\Gamma; \Delta \vdash P :: T}{\Gamma; \Delta, x{:}\mathbf{1} \vdash P :: T}\;(\text{T1L}) \qquad \frac{}{\Gamma; \cdot \vdash 0 :: x{:}\mathbf{1}}\;(\text{T1R})$$

$$\frac{\Gamma; \Delta, y{:}A, x{:}B \vdash P :: T}{\Gamma; \Delta, x{:}A \otimes B \vdash x(y).P :: T}\;(\text{T}\otimes\text{L}) \qquad \frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: x{:}B}{\Gamma; \Delta, \Delta' \vdash (\nu y)x\langle y\rangle.(P \mid Q) :: x{:}A \otimes B}\;(\text{T}\otimes\text{R})$$

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q :: T}{\Gamma; \Delta, \Delta', x{:}A \multimap B \vdash (\nu y)x\langle y\rangle.(P \mid Q) :: T}\;(\text{T}\multimap\text{L}) \qquad \frac{\Gamma; \Delta, y{:}A \vdash P :: x{:}B}{\Gamma; \Delta \vdash x(y).P :: x{:}A \multimap B}\;(\text{T}\multimap\text{R})$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: T}\;(\mathsf{Tcut}) \qquad \frac{\Gamma; \cdot \vdash P :: y{:}A \quad \Gamma, u{:}A; \Delta \vdash Q :: T}{\Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: T}\;(\mathsf{Tcut}^{!})$$

$$\frac{\Gamma, u{:}A; \Delta, y{:}A \vdash P :: T}{\Gamma, u{:}A; \Delta \vdash (\nu y)u\langle y\rangle.P :: T}\;(\mathsf{Tcopy})$$

$$\frac{\Gamma, u{:}A; \Delta \vdash P\{u/x\} :: T}{\Gamma; \Delta, x{:}!A \vdash P :: T}\;(\text{T!L}) \qquad \frac{\Gamma; \cdot \vdash Q :: y{:}A}{\Gamma; \cdot \vdash !x(y).Q :: x{:}!A}\;(\text{T!R})$$

$$\frac{\Gamma; \Delta, x{:}A \vdash P :: T \quad \Gamma; \Delta, x{:}B \vdash Q :: T}{\Gamma; \Delta, x{:}A \oplus B \vdash x.\mathtt{case}(P, Q) :: T}\;(\text{T}\oplus\text{L}) \qquad \frac{\Gamma; \Delta, x{:}B \vdash P :: T}{\Gamma; \Delta, x{:}A \mathbin{\&} B \vdash x.\mathtt{inr}; P :: T}\;(\text{T\&L}_2)$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta \vdash Q :: x{:}B}{\Gamma; \Delta \vdash x.\mathtt{case}(P, Q) :: x{:}A \mathbin{\&} B}\;(\text{T\&R}) \qquad \frac{\Gamma; \Delta, x{:}A \vdash P :: T}{\Gamma; \Delta, x{:}A \mathbin{\&} B \vdash x.\mathtt{inl}; P :: T}\;(\text{T\&L}_1)$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A}{\Gamma; \Delta \vdash x.\mathtt{inl}; P :: x{:}A \oplus B}\;(\text{T}\oplus\text{R}_1) \qquad \frac{\Gamma; \Delta \vdash P :: x{:}B}{\Gamma; \Delta \vdash x.\mathtt{inr}; P :: x{:}A \oplus B}\;(\text{T}\oplus\text{R}_2)$$

Fig. 2. The Type System $\pi\mathtt{DILL}$

(sending a session channel of type A) to its partner before proceeding as specified by B. In a similar way, $A \multimap B$ types a session channel that first performs an input (receiving a session channel of type A) from its partner, before proceeding as specified by B. The type $\mathbf{1}$ means that the session terminated, no further interaction will take place on it. Notice that names of type $\mathbf{1}$ may still be passed around in sessions, as opaque values. $A \oplus B$ types a session that either selects "left" and then proceed as specified by A, or else selects "right", and then proceeds as specified by B. Dually, $A \mathbin{\&} B$ types a session channel that offers its partner a choice between an A typed behavior ("left" choice) and a B typed behavior ("right" choice). The type $!A$ types a non-session (non-linearized, shared) channel (called *standard channel* in [13]), to be used by a server for spawning an arbitrary number of new sessions (possibly none), each one conforming to type A.

A type environment is a collection of type assignments, of the form $x : A$ where x is a name and A a type, the names being pairwise disjoint. Following the insights behind dual intuitionistic linear logic, which goes back to Andreoli's *dyadic* system for classical linear logic [2], we distinguish two kinds of type environments subject to different structural properties: a *linear* part Δ and an *unrestricted* part Γ, where weakening and contraction principles hold for Γ but not for Δ. A judgment of our system has then the form $\Gamma; \Delta \vdash P :: z{:}C$ where name declarations in Γ are always propagated unchanged to all premises in the typing rules, while name declarations in Δ are handled multiplicatively or additively, depending on the nature of the type being defined. The domains of Γ, Δ and $z{:}C$ are required to be pairwise disjoint.

Intuitively, such a judgment asserts: P is ensured to safely provide a usage of name z according to the behavior (session) specified by type C, whenever composed with any process environment providing usages of names according to the behaviors specified by names in $\Gamma; \Delta$. As shown in Section 5, in our case safety ensures that the behavior is free of communication errors and deadlock. A pure client Q that just relies on external services, and does not provide any, will be typed as $\Gamma; \Delta \vdash Q :: -{:}\mathbf{1}$. In general, a process P such that $\Gamma; \Delta \vdash P :: z{:}C$ represents a system providing behavior C at channel z, building on "services" declared in $\Gamma; \Delta$. Of particular interest is a system typed as $\Gamma; \Delta \vdash R :: z{:}!A$, representing a shared server. Quite interestingly, the asymmetry induced by the intuitionistic interpretation of $!A$ enforces locality of shared names but not of linear (session names), which exactly corresponds to the intended model of sessions.

We present the rules of our type system $\pi\mathtt{DILL}$ in Fig. 2. We use T, S for right hand side singleton environments (e.g., $z{:}C$). The interpretation of the various rules should be clear, given the explanation of types given above. Notice that since in $\otimes\mathsf{R}$ the sent name is always fresh, our typed calculus conforms to a session-based internal mobility discipline [23,7], without loss of expressiveness. The composition rules (cut and cut$^!$) follow the "composition plus hiding" principle [1], extended to a name passing setting. More familiar linear typing rules for parallel composition (*e.g.*, as in [21]) are derivable (see Section 6). Since we are considering π-calculus terms up to structural congruence, typability is closed under $\equiv$ by definition. $\pi\mathtt{DILL}$ enjoys the usual properties of equivariance, weakening in Γ and contraction in Γ. The coverage property also holds: if $\Gamma; \Delta \vdash P :: z{:}A$ then $\mathit{fn}(P) \subseteq \Gamma \cup \Delta \cup \{z\}$. In the presence of type-annotated restrictions $(\nu x{:}A)P$, as usual in typed π-calculi [24], type-checking is decidable.

We illustrate the type system with a simple example, frequently used to motivate session based interactions (see *e.g.*, [13]). A client may choose between a "buy" operation, in which it indicates a product name and a credit card number to receive a receipt, and a "quote" operation, in which it indicates a product name, to obtain the product price. From the client perspective, the session protocol exposed by the server may be specified by the type

$$\mathit{ServerProto} \triangleq (N \multimap I \multimap (N \otimes \mathbf{1}))\ \&\ (N \multimap (I \otimes \mathbf{1}))$$

We assume that N and I are types representing shareable values (*e.g.*, strings N and integers I). To simplify, we set $N = I = \mathbf{1}$. Assuming s to be the name of the session channel connecting the client and server, consider the code

$$\mathit{QClntBody}_s \triangleq s.\mathtt{inr}; (\boldsymbol{\nu}\mathit{tea})s\langle \mathit{tea}\rangle.s(\mathit{pr}).\mathbf{0}$$

$\mathit{QClntBody}_s$ specifies a client that asks for the price of tea (we simply abstract away from what the client might do with the price after reading it). It first selects the quoting operation on the server ($s.\mathtt{inr}$), then sends the id of the product to the server ($s\langle \mathit{tea}\rangle$), then receives the price $s(\mathit{pr})$ from the server and finally terminates the session ($\mathbf{0}$). Then

$$\cdot; s : \mathit{ServerProto} \vdash \mathit{QClntBody}_s :: -{:}\mathbf{1}$$

is derivable (by $T\mathbf{1}\mathsf{R}$, $T\otimes\mathsf{L}$, $T\multimap\mathsf{L}$ and $T\&\mathsf{L}_2$). Here we wrote $-$ for an anonymous variable that does not appear in *QClntBody*. This is possible even in a linear type discipline since the inactive process $\mathbf{0}$ is typed by $x{:}\mathbf{1}$ and does not use x. Concerning the server code, let $\mathit{SrvBody}_s \triangleq s.\mathtt{case}(\ s(\mathit{pn}).s(\mathit{cn}).(\nu \mathit{rc})s\langle \mathit{rc}\rangle.\mathbf{0},\ \ s(\mathit{pn}).(\nu \mathit{pr})s\langle \mathit{pr}\rangle.\mathbf{0}\ )$.

$$\frac{\Gamma;\Delta\vdash D:C}{\Gamma;\Delta,x:\mathbf{1}\vdash\mathsf{1L}\;x\;D:C}\;(\mathsf{1L})\qquad\frac{}{\Gamma;\cdot\vdash\mathsf{1R}:\mathbf{1}}\;(\mathsf{1R})$$

$$\frac{\Gamma;\Delta,y:A,x:B\vdash D:C}{\Gamma;\Delta,x:A\otimes B\vdash\otimes\mathsf{L}\;x\;(y.x.\;D):C}\;(\otimes\mathsf{L})\qquad\frac{\Gamma;\Delta\vdash D:A\quad\Gamma;\Delta'\vdash E:B}{\Gamma;\Delta,\Delta'\vdash\otimes\mathsf{R}\;D\;E:A\otimes B}\;(\otimes\mathsf{R})$$

$$\frac{\Gamma;\Delta\vdash D:A\qquad\Gamma;\Delta',x:B\vdash E:C}{\Gamma;\Delta,\Delta',x:A\multimap B\vdash\multimap\mathsf{L}\;x\;D\;(x.\;E):C}\;(\multimap\mathsf{L})\qquad\frac{\Gamma;\Delta,y:A\vdash D:B}{\Gamma;\Delta\vdash\multimap\mathsf{R}\;(y.\;D):A\multimap B}\;(\multimap\mathsf{R})$$

$$\frac{\Gamma;\Delta\vdash D:A\quad\Gamma;\Delta',x:A\vdash E:C}{\Gamma;\Delta,\Delta'\vdash\mathsf{cut}\;D\;(x.E):C}\;(\mathsf{cut})\qquad\frac{\Gamma;\cdot\vdash D:A\quad\Gamma,u:A;\Delta\vdash E:C}{\Gamma;\Delta\vdash\mathsf{cut}^!\;D\;(u.\;E):C}\;(\mathsf{cut}^!)$$

$$\frac{\Gamma,u:A;\Delta,y:A\vdash D:C}{\Gamma,u:A;\Delta\vdash\mathsf{copy}\;u\;(y.\;D):C}\;(\mathsf{copy})$$

$$\frac{\Gamma;\cdot\vdash D:A}{\Gamma;\cdot\vdash\mathsf{!R}\;D:!A}\;(\mathsf{!R})\qquad\frac{\Gamma,u:A;\Delta\vdash D:C}{\Gamma;\Delta,x:!A\vdash\mathsf{!L}\;x\;(u.D):C}\;(\mathsf{!L})$$

$$\frac{\Gamma;\Delta,x:A\vdash D:C}{\Gamma;\Delta,x:A\mathbin{\&}B\vdash\mathsf{\&L}_1\;x\;(x.\;D):C}\;(\mathsf{\&L}_1)\qquad\frac{\Gamma;\Delta,x:B\vdash D:C}{\Gamma;\Delta,x:A\mathbin{\&}B\vdash\mathsf{\&L}_2\;x\;(x.\;D):C}\;(\mathsf{\&L}_2)$$

$$\frac{\Gamma;\Delta\vdash D:A\quad\Gamma;\Delta\vdash E:B}{\Gamma;\Delta\vdash\mathsf{\&R}\;D\;E:A\mathbin{\&}B}\;(\mathsf{\&R})\qquad\frac{\Gamma;\Delta\,x:A\vdash D:C\quad\Gamma;\Delta,x:B\vdash E:C}{\Gamma;\Delta,x:A\oplus B\vdash\oplus\mathsf{L}\;x\;(x.\;D)\;(x.\;E):C}\;(\oplus\mathsf{L})$$

$$\frac{\Gamma;\Delta\vdash D:A}{\Gamma;\Delta\vdash\oplus\mathsf{R}_1\;D:A\oplus B}\;(\oplus\mathsf{R}_1)\qquad\frac{\Gamma;\Delta\vdash D:B}{\Gamma;\Delta\vdash\oplus\mathsf{R}_2\;D:A\oplus B}\;(\oplus\mathsf{R}_2)$$

Fig. 3. Dual Intuitionistic Linear Logic DILL

Then $\cdot;\cdot\vdash$ *SrvBody*$_s$ :: s:*ServerProto* is derivable, by $T\&\mathsf{R}$. By $T\mathsf{cut}$ we obtain for the system *QSimple* $\triangleq(\boldsymbol{\nu}s)(SrvBody_s \mid QClntBody_s)$ the typing $\cdot;\cdot\vdash QSimple :: -{:}\mathbf{1}$. In this example we have only introduced processes interacting in a single session, but clearly the system accomodates all the generality of session types, e.g., a simple process interacting in different sessions is $x{:}A\multimap\mathbf{1}, y{:}A\otimes\mathbf{1}\vdash y(w).(\boldsymbol{\nu}k)x\langle k\rangle.\mathbf{0} :: -{:}\mathbf{1}$.

4 Dual Intuitionistic Linear Logic

As presented, session type constructors correspond directly to intuitionistic linear logic connectives. Typing judgments directly correspond to sequents in dual intuitionistic linear logic, by erasing processes [3,10]. In Figure 3 we present the DILL sequent calculus. In our presentation, DILL is conveniently equipped with a faithful proof term assignment, so sequents have the form $\Gamma;\Delta\vdash D : C$ where Γ is the unrestricted context, Δ the linear context, C a formula (= type) and D the proof term that faithfully represents the derivation of $\Gamma;\Delta\vdash C$. Our use of names in the proof system will be consistent with the proof discipline, u, v, w for variables in Γ and x, y, z for variables in Δ. This is consistent with standard usage of names in π-calculi. Given the parallel structure of the two systems, if $\Gamma;\Delta\vdash D : A$ is derivable in DILL then there is a process P and a name z such that $\Gamma;\Delta\vdash P :: z{:}A$ is derivable in πDILL, and the converse result also holds: if $\Gamma;\Delta\vdash P :: z{:}A$ is derivable in πDILL there is a

D	$\leadsto \hat{D}^z$	D	$\leadsto \hat{D}^z$
$\mathsf{1R}$	$\leadsto 0$	$\oplus\mathsf{R}_1\ D$	$\leadsto z.\mathsf{inl}; \hat{D}^z$
$\mathsf{1L}\ x\ D$	$\leadsto \hat{D}^z$	$\oplus\mathsf{R}_2\ E$	$\leadsto z.\mathsf{inr}; \hat{E}^z$
$\otimes\mathsf{R}\ D\ E$	$\leadsto (\nu y)\, z\langle y\rangle.\,(\hat{D}^y \mid \hat{E}^z)$	$\oplus\mathsf{L}\ x\ (x.\,D)\ (x.\,E)$	$\leadsto x.\mathsf{case}(\hat{D}^z, \hat{E}^z)$
$\otimes\mathsf{L}\ x\ (y.x.\,D)$	$\leadsto x(y).\,\hat{D}^z$	$\mathsf{cut}\ D\ (x.\,E)$	$\leadsto (\nu x)(\hat{D}^x \mid \hat{E}^z)$
$\multimap\mathsf{R}\ (y.\,D)$	$\leadsto z(y).\,\hat{D}^z$		
$\multimap\mathsf{L}\ x\ D\ (x.\,E)$	$\leadsto (\nu y)\, x\langle y\rangle.\,(\hat{D}^y \mid \hat{E}^z)$	$\mathsf{!R}\ D$	$\leadsto \mathsf{!}z(y).\,\hat{D}^y$
$\&\mathsf{R}\ D\ E$	$\leadsto z.\,\mathsf{case}(\hat{D}^z, \hat{E}^z)$	$\mathsf{!L}\ x\ (u.\,D)$	$\leadsto \hat{D}^z\{x/u\}$
$\&\mathsf{L}_1\ x\ (x.\,D)$	$\leadsto x.\,\mathsf{inl}; \hat{D}^z$	$\mathsf{copy}\ u\ (y.\,D)$	$\leadsto (\nu y)\, u\langle y\rangle.\,\hat{D}^z$
$\&\mathsf{L}_2\ x\ (x.\,E)$	$\leadsto x.\,\mathsf{inr}; \hat{E}^z$	$\mathsf{cut}^!\ D\ (u.\,E)$	$\leadsto (\nu u)((\mathsf{!}u(y).\,\hat{D}^y) \mid \hat{E}^z)$

Fig. 4. Proof D extracts to process $\hat{D}^z$

derivation D that proves $\Gamma; \Delta \vdash D : A$. This correspondence is made explicit by a translation from faithful proof terms to processes, defined in Fig. 4: for $\Gamma; \Delta \vdash D : C$ we write $\hat{D}^z$ for the translation of D such that $\Gamma; \Delta \vdash \hat{D}^z :: z{:}C$.

Definition 4.1 (Typed Extraction). *We write $\Gamma; \Delta \vdash D \leadsto P :: z{:}A$, meaning "proof D extracts to P", whenever $\Gamma; \Delta \vdash D : A$ and $\Gamma; \Delta \vdash P :: z{:}A$ and $P \equiv \hat{D}^z$.*

Typed extraction is unique up to structural congruence, in the sense that if $\Gamma; \Delta \vdash D \leadsto P :: z{:}A$ and $\Gamma; \Delta \vdash D \leadsto Q :: z{:}A$ then $P \equiv Q$, as a consequence of closure of typing under structural congruence. The system DILL as presented does not admit atomic formulas, and hence has no true initial sequents. However, the correspondence mentioned above yields an explicit identity theorem:

Proposition 4.2. *For any type A and distinct names x, y, there is a process $id_A(x, y)$ and a cut-free derivation D such that $\cdot; x{:}A \vdash D \leadsto id_A(x, y) :: y{:}A$.*

The $id_A(x, y)$ process, with exactly the free names x, y, implements a synchronous mediator that bidirectionally plays the protocol specified by A between channels x and y. For example, we analyze the interpretation of the sequent $A \otimes B \vdash B \otimes A$. We have

$$x{:}A \otimes B \vdash F \leadsto x(z).(\nu n)y\langle n\rangle.(P \mid Q) :: y{:}B \otimes A$$

where $F = \otimes\mathsf{L}\ x\ (z.x.\ \otimes\mathsf{R}\ D\ E)$, $P = id_B(x, n)$ and $Q = id_A(z, y)$. This process is an interactive proxy that coerces a session of type $A \otimes B$ at x to a session of type $B \otimes A$ at y. It first receives a session of type A (bound to z) and after sending on y a session of type B (played by copying the continuation of x to n), it progresses with a session of type A on y (copying the continuation of z to y).

As processes are related by structural and computational rules, namely those involved in the definition of $\equiv$ and $\to$, derivations in DILL are related by structural and computational rules, that express certain sound proof transformations that arise in cut-elimination. The reductions (Figure 5) generally take place when a right rule meets a left rule for the same connective, and correspond to reduction steps in the process term assignment. On the left, we show the usual reductions for cuts; on the right, we show the corresponding reductions (if any) of the process terms, modulo structural congruence. Since equivalences depend on variable occurrences, we write D_x if x may occur in D.

The structural conversions in Figure 6 correspond to structural equivalences in the π-calculus, since they just change the order of cuts, *e.g.*, $(\mathsf{cut}/-/\mathsf{cut}_1)$ translates to

$$(\nu x)(\hat{D}^x \mid (\nu y)(\hat{E}^y \mid \hat{F}^z)) \equiv (\nu y)((\nu x)(\hat{D}^x \mid \hat{E}^y) \mid \hat{F}^z)$$

In addition, we have two special conversions. Among those, $(\mathsf{cut}/\mathbf{1}\mathsf{R}/\mathbf{1}\mathsf{L})$ is not needed in order to simulate the π-calculus reduction, while $(\mathsf{cut}/!\mathsf{R}/!\mathsf{L})$ is. In cut-elimination procedures, these are always used from left to right. Here, they are listed as equivalences because the corresponding π-calculus terms are structurally congruent. The root cause for this is that the rules $\mathbf{1}\mathsf{L}$ and $!\mathsf{L}$ are *silent*: the extracted terms in the premise and conclusion are the same, modulo renaming. For $\mathbf{1}\mathsf{L}$, this is the case because a terminated process, represented by $0 :: - : \mathbf{1}$ silently disappears from a parallel composition by structural congruence. For $!\mathsf{L}$, this is the case because the actual replication of a server process is captured in the copy rule which clones $u{:}A$ to $y{:}A$, rule rather than $!\mathsf{L}$. It is precisely for this reason that the rule commuting a persistent cut $(\mathsf{cut}^!)$ over a copy rule (copy) is among the computational conversions.

The structural conversions in Figure 8 propagate $\mathsf{cut}^!$. From the proof theoretic perspective, because $\mathsf{cut}^!$ cuts a persistent variable u, $\mathsf{cut}^!$ may be duplicated or erased. On the π-calculus side, these no longer correspond to structural congruences, but, quite remarkably, to behavioral equivalences, derivable from known properties of typed processes, the (sharpened) Replication Theorems [24]. These hold in our language, due to our interpretation of ! types. Our operational correspondence results also depend on six commuting conversions, four in Figure 7 plus two symmetric versions. The commuting conversions push a cut up (or inside) the $\mathbf{1}\mathsf{L}$ and $!\mathsf{L}$ rules. During the usual cut elimination procedures, these are used from left to right. In the correspondence with the sequent calculus, the situation is more complex. Because the $\mathbf{1}\mathsf{L}$ and $!\mathsf{L}$ rules do not affect the extracted term, cuts have to be permuted with these two rules in order to simulate π-calculus reduction. From the process calculus perspective, such conversions correspond to identity. There is a second group of commuting conversions (not shown), not necessary for our current development. Those do not correspond to structural congruence nor to strong bisimilarities on π-calculus, as they may not preserve process behavior in the general untyped setting, since they promote an action prefix from a subexpression to the top level. We conjecture that such equations denote behavioral identities under a natural definition of typed observational congruence for our calculus.

Definition 4.3 (Relations on derivations induced by conversions). *(1) $\equiv$: the least congruence on derivations generated by the structural conversions (I) and the commuting conversions (II): (2) $\simeq_s$: the least congruence on derivations generated by all structural conversions (I-III). We extend $\simeq_s$ to processes as the congruence generated by the process equations on the right. (3) $\Mapsto$: the reduction on derivations obtained by orienting all conversions in the direction shown, from left to right or top to bottom.*

As discussed above, $\simeq_s$ is a typed behavioral equivalence on processes.

5 Computational Correspondence, Preservation, and Progress

We now present the results stating the key properties of our type system and logical interpretation. Theorem 5.3 states the existence of a simulation between reductions in

$$\begin{array}{ll}
\mathsf{cut}\ (\otimes\mathsf{R}\ D_1\ D_2)\ (x.\,\otimes\mathsf{L}\ x\ (y.x.\,E_{xy})) & \rightsquigarrow (\nu x)(((\nu y)\,x\langle y\rangle.\,(\hat{D}_1^y \mid \hat{D}_2^x)) \mid x(y).\,\hat{E}^z) \\
\Rightarrow & \ \rightarrow \\
\mathsf{cut}\ D_1\ (y.\,\mathsf{cut}\ D_2\ (x.\,E_{xy})) & \rightsquigarrow (\nu x)(\nu y)(\hat{D}_1^y \mid \hat{D}_2^x \mid \hat{E}^z) \\
\mathsf{cut}\ (\multimap\mathsf{R}\ (y.\,D_y))\ (x.\,\multimap\mathsf{L}\ x\ E_1\ (x.\,E_{2x})) & \rightsquigarrow (\nu x)((x(y).\,\hat{D}^x) \mid (\nu y)\,x\langle y\rangle.\,(\hat{E}_1^y \mid \hat{E}_2^z)) \\
\Rightarrow & \ \rightarrow \\
\mathsf{cut}\ (\mathsf{cut}\ E_1\ (y.\,D_y))\ (x.\,E_{2x}) & \rightsquigarrow (\nu x)(\nu y)(\hat{D}^x \mid \hat{E}_1^y \mid \hat{E}_2^z) \\
\mathsf{cut}\ (\&\mathsf{R}\ D_1\ D_2)\ (x.\&\mathsf{L}_i\ x\ (x.\,E_x)) & \rightsquigarrow (\nu x)(x.\mathsf{case}(\hat{D}_1^x, \hat{D}_2^x) \mid x.\mathsf{inl};\hat{E}^z) \\
\Rightarrow & \ \rightarrow \\
\mathsf{cut}\ D_i\ (x.\,E_x) & \rightsquigarrow (\nu x)(\hat{D}_i^x \mid \hat{E}^z) \\
\mathsf{cut}\ (\oplus\mathsf{R}_i\ D)\ (x.\oplus\mathsf{L}\ x\ (x.\,E_{1x})\ (x.\,E_{2x})) & \rightsquigarrow (\nu x)(x.\mathsf{inl};\hat{D}^x \mid x.\mathsf{case}(\hat{E}_1^z, \hat{E}_2^z)) \\
\Rightarrow & \ \rightarrow \\
\mathsf{cut}\ D\ (x.\,E_{ix}) & \rightsquigarrow (\nu x)(\hat{D}^x \mid \hat{E}_i^z) \\
\mathsf{cut}^!\ D\ (u.\,\mathsf{copy}\ u\ (y.\ E_{uy})) & \rightsquigarrow (\nu u)((!u(y).\,\hat{D}^y) \mid (\nu y)\,u\langle y\rangle.\,\hat{E}^z) \\
\Rightarrow & \ \rightarrow \\
\mathsf{cut}\ D\ (y.\,\mathsf{cut}^!\ D\ (u.\ E_{uy})) & \rightsquigarrow (\nu y)(\hat{D}^y \mid (\nu u)((!u(y).\,\hat{D}^y) \mid \hat{E}^z))
\end{array}$$

Fig. 5. Computational Conversions

$$\begin{array}{lll}
(\mathsf{cut}/{-}/\mathsf{cut}_1) & \mathsf{cut}\ D\ (x.\,\mathsf{cut}\ E_x\ (y.\ F_y)) & \equiv \mathsf{cut}\ (\mathsf{cut}\ D\ (x.\,E_x))\ (y.\ F_y) \\
(\mathsf{cut}/{-}/\mathsf{cut}_2) & \mathsf{cut}\ D\ (x.\,\mathsf{cut}\ E\ (y.\ F_{xy})) & \equiv \mathsf{cut}\ E\ (y.\,\mathsf{cut}\ D\ (x.\ F_{xy})) \\
(\mathsf{cut}/\mathsf{cut}^!/{-}) & \mathsf{cut}\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (x.\,F_x) & \equiv \mathsf{cut}^!\ D\ (u.\,\mathsf{cut}\ E_u\ (x.\,F_x)) \\
(\mathsf{cut}/{-}/\mathsf{cut}^!) & \mathsf{cut}\ D\ (x.\,\mathsf{cut}^!\ E\ (u.\,F_{xu})) & \equiv (\mathsf{cut}^!\ E\ (u.\,\mathsf{cut}\ D\ (x.\,F_{xu})) \\
(\mathsf{cut}/\mathbf{1}\mathsf{R}/\mathbf{1}\mathsf{L}) & \mathsf{cut}\ \mathbf{1}\mathsf{R}\ (x.\,\mathbf{1}\mathsf{L}\ x\ D) & \equiv D \\
(\mathsf{cut}/!\mathsf{R}/!\mathsf{L}) & \mathsf{cut}\ (!\mathsf{R}\ D)\ (x.\,!\mathsf{L}\ x\ (u.\,E)) & \equiv \mathsf{cut}^!\ D\ (u.\,E)
\end{array}$$

Fig. 6. Structural Conversions (I): Cut Conversions

$$\begin{array}{lll}
(\mathsf{cut}/\mathbf{1}\mathsf{L}/{-}) & \mathsf{cut}\ (\mathbf{1}\mathsf{L}\ y\ D)\ (x.\,F_x) & \equiv \mathbf{1}\mathsf{L}\ y\ (\mathsf{cut}\ D\ (x.\,F_x)) \\
(\mathsf{cut}/!\mathsf{L}/{-}) & \mathsf{cut}\ (!\mathsf{L}\ y\ (u.\,D_u))\ (x.\,F_x) & \equiv\ !\mathsf{L}\ y\ (u.\,\mathsf{cut}\ D_u\ (x.\,F_x)) \\
(\mathsf{cut}^!/{-}/\mathbf{1}\mathsf{L}) & \mathsf{cut}^!\ D\ (u.\,\mathbf{1}\mathsf{L}\ y\ E_u) & \equiv \mathbf{1}\mathsf{L}\ y\ (\mathsf{cut}^!\ D\ (u.\,E_u)) \\
(\mathsf{cut}^!/{-}/!\mathsf{L}) & \mathsf{cut}^!\ D\ (u.\,!\mathsf{L}\ y\ (v.\,E_{uv})) & \equiv\ !\mathsf{L}\ y\ (v.\ \mathsf{cut}^!\ D\ (u.\,E_{uv}))
\end{array}$$

Fig. 7. Structural Conversions (II): Commuting Conversions

$$\begin{array}{ll}
\mathsf{cut}^!\ D\ (u.\,\mathsf{cut}\ E_u\ (y.\ F_{uy})) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid (\nu y)(\hat{E}^y \mid \hat{F}^z)) \\
\simeq & \ \simeq \\
\mathsf{cut}\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (y.\ \mathsf{cut}^!\ D\ (u.\,F_{uy})) & \rightsquigarrow (\nu y)((\nu u)(!u(y).\hat{D}^y \mid \hat{E}^y) \mid \\
 & \qquad\qquad (\nu u)(!u(y).\hat{D}^y \mid \hat{F}^z)\ ) \\
\mathsf{cut}^!\ D\ (u.\,\mathsf{cut}^!\ E_u\ (v.\ F_{uv})) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid (\nu v)(!v(y).\hat{E}^y \mid \hat{F}^z)) \\
\simeq & \ \simeq \\
\mathsf{cut}^!\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (v.\ \mathsf{cut}^!\ D\ (u.\,F_{uv})) & \rightsquigarrow (\nu v)((!v(y).(\nu u)(!u(y).\hat{D}^y \mid \hat{E}^y)) \mid \\
 & \qquad\qquad (\nu u)(!u(y).\hat{D}^y \mid \hat{F}^z)\ ) \\
\mathsf{cut}^!\ (\mathsf{cut}^!\ D\ (u.\,E_u))\ (v.\ F_v) & \rightsquigarrow (\nu v)(!v(y).(\nu u)(!u(y).\hat{D}^y \mid \hat{E}^y)) \mid F^z) \\
\simeq & \ \simeq \\
\mathsf{cut}^!\ D\ (u.\,\mathsf{cut}^!\ E_u\ (v.\ F_v)) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid (\nu v)(!v(y).\hat{E}^y \mid \hat{F}^z)) \\
\mathsf{cut}^!\ D\ (u.\,E) & \rightsquigarrow (\nu u)(!u(y).\hat{D}^y \mid \hat{E}^z) \\
\simeq & \ \simeq \\
E & \rightsquigarrow \hat{E}^z \quad (\text{for } u \notin FN(\hat{E}^z))
\end{array}$$

Fig. 8. Structural Conversions (III): Cut! Conversions

the typed π-calculus and proof conversions / reductions, expressing a strong form of subject reduction for our type system. The proof relies on several auxiliary lemmas, which we mostly omit, among them a sequence of lemmas relating process reduction with derivation reduction, from which we select two typical examples.

Lemma 5.1. *Assume (a) $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x{:}A_1 \otimes A_2$ with $P \overset{\overline{(\nu y)x\langle y\rangle}}{\rightarrow} P'$; and (b) $\Gamma; \Delta_2, x{:}A_1 \otimes A_2 \vdash E \rightsquigarrow Q :: z{:}C$ with $Q \overset{x(y)}{\rightarrow} Q'$. Then (c)* $\mathsf{cut}\ D\ (x.\, E) \equiv\Rightarrow\equiv F$ *for some F; (d) $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$ for $R \equiv (\nu y)(\nu x)(P' \mid Q')$.*

Lemma 5.2. *Assume (a) $\Gamma; \cdot \vdash D \rightsquigarrow P :: x{:}A$; (b) $\Gamma, u{:}A; \Delta_2 \vdash E \rightsquigarrow Q :: z{:}C$ with $Q \overset{\overline{(\nu y)u\langle y\rangle}}{\rightarrow} Q'$. Then (c)* $\mathsf{cut}^!\ D\ (u.\, E) \equiv\Rightarrow\equiv F$ *for some F; (d) $\Gamma; \Delta \vdash F \rightsquigarrow R :: z{:}C$ for some $R \equiv (\nu u)(!u(x).P \mid (\nu y)(P\{y/x\} \mid Q'))$.*

Theorem 5.3. *Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ and $P \rightarrow Q$. Then there is E such that $D \equiv\Rightarrow\equiv E$ and $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$*

Proof. By induction on the structure of derivation D. The possible cases for D are $D = \mathsf{1L}\ y\ D'$, $D = \mathsf{!L}\ x\ (u.\, D')$, $D = \mathsf{cut}\ D_1\ (x.\, D_2)$, and $D = \mathsf{cut}^!\ D_1\ (x.\, D_2)$, in all other cases $P \not\rightarrow$. Key cases are the cuts, where we rely on a series of reduction lemmas, one for each type C of cut formula, which assign certain proof conversions to process labeled transitions. For example, for $C = C_1 \otimes C_2$, we rely on Lemma 5.1. The case of $\mathsf{cut}^!$, similar to the case $C = !C'$, relies on Lemma 5.2. We show such case in detail. Let $D = \mathsf{cut}^!\ D_1\ (u.\, D_2)$. We have $P \equiv (\nu u)(!u(w).P_1 \mid P_2)$, $\Gamma; \vdash D_1 \rightsquigarrow P_1 :: x{:}C$, and $\Gamma, u : C; \Delta \vdash D_2 \rightsquigarrow P_2 :: z{:}A$ by inversion. Since $P \rightarrow Q$, there two cases: (1) $P_2 \rightarrow Q_2$ and $Q = (\nu u)(!u(w).P_1 \mid Q_2)$, or (2) $P_2 \overset{\alpha}{\rightarrow} Q_2$ where $\alpha = \overline{(\nu y)x\langle y\rangle}$ and $Q = (\nu u)(!u(w).P_1 \mid (\nu y)(P_1\{y/x\} \mid Q_2))$. Case (1): We have $\Gamma, u : C; \Delta \vdash D_2 \rightsquigarrow Q_2 :: z{:}A$ for E' with $D_2 \equiv\Rightarrow\equiv E'$ by i.h. Then $\mathsf{cut}^!\ D_1\ (u.\, D_2) \equiv\Rightarrow\equiv \mathsf{cut}^!\ D_1\ (u.\, E')$ by congruence. Let $E = \mathsf{cut}^!\ D_1\ (u.\, E')$. So $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$ by $\mathsf{cut}^!$. Case (2): By Lemma 5.2, $\mathsf{cut}^!\ D_1\ (u.\, D_2) \equiv\Rightarrow\equiv E$ for some E, and $\Gamma; \Delta \vdash E \rightsquigarrow R :: z{:}A$ with $R \equiv Q$. □

Theorems 5.4 and 5.5 state that any proof reduction or conversion also corresponds to either a process equivalence or to a reduction step on the π-calculus.

Theorem 5.4. *Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ and $D \simeq_s E$. Then there is Q where $P \simeq_s Q$ and $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$.*

Proof. Following the commuting squares relating $\equiv$, $\rightsquigarrow$ and $\simeq$ in Figures 6, 7 and 8. □

Theorem 5.5. *Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ and $D \Rightarrow E$. Then there is Q such that $P \rightarrow Q$ and $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$.*

Proof. Following the commuting squares relating $\Rightarrow$, $\rightsquigarrow$ and $\rightarrow$ in Figure 5. □

Notice that the simulation of π-calculus reductions by proof term conversions provided by Theorem 5.3, and from which subject reduction follows, is very tight indeed, as reduction is simulated up to structural congruence, which is a very fine equivalence on processes. To that end, structural conversions need to be applied symmetrically (as equations), unlike in a standard proof of cut-elimination, where they are usually considered as directed computational steps. Under the assumptions of Theorem 5.3, we can also prove that there is an E such that $D \mapsto\Rightarrow E$ and $\Gamma; \Delta \vdash E \rightsquigarrow R :: z{:}A$, for

$Q \simeq_s R$. Thus, even if one considers the proof conversions as directed reduction rules ($\Rrightarrow$), we still obtain a sound simulation up to typed strong behavioral congruence.

We now state type preservation and progress results for our type system. The subject reduction property (Theorem 5.6) directly follows from Theorem 5.3.

Theorem 5.6 (Subject Reduction). *If $\Gamma;\Delta \vdash P :: z{:}A$ and $P \rightarrow Q$ then $\Gamma;\Delta \vdash Q :: z{:}A$.*

Together with direct consequences of linear typing, Theorem 5.6 ensures session fidelity. Our type discipline also enforces a global progress property. For any P, define

$$\mathit{live}(P) \;\; \mathit{iff} \;\; P \equiv (\nu\overline{n})(\pi.Q \mid R) \quad \mathit{for\ some}\ \pi.Q, R, \overline{n}$$

where $\pi.Q$ is a *non-replicated* guarded process. We first establish the following contextual progress property, from which Theorem 5.8 follows as a corollary.

Lemma 5.7. *Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}C$. If $\mathit{live}(P)$ then there is Q such that either*

1. *$P \rightarrow Q$, or*
2. *$P \xrightarrow{\alpha} Q$ for α where $s(\alpha) \in (z, \Gamma, \Delta)$. More: if $C = {!A}$ for some A, then $s(\alpha) \neq z$.*

Proof. Induction on derivation D. The key cases are $D = \mathsf{cut}\, D_1\, (y.\, D_2)$ and $D = \mathsf{cut}^!\, D_1(u.\, D_2)$. In the case of cut, we rely on lemmas that characterize the possible actions of a process on name $y{:}A$, depending on type A. These lemmas show that a synchronization between dual actions must occur. For $\mathsf{cut}^!$, an inversion lemma is needed, stating that free names of a non-live process can only be typed by $\mathbf{1}$ or $!A$ types. □

Theorem 5.8 (Progress). *If $\cdot;\cdot \vdash D \rightsquigarrow P :: x{:}\mathbf{1}$ and $\mathit{live}(P)$ then exists Q st. $P \rightarrow Q$.*

Proof. By Lemma 5.7 and the fact that P cannot perform any action α with subject $s(\alpha) = x$ since $x{:}\mathbf{1}$ (by the action shape characterization lemmas). □

6 Discussion and Further Examples

We further compare our linear type system for (finite) session types with more familiar session type systems [21,18,13]. An immediate observation is that in our case types are freely generated, while traditionally there is a stratification of types in "session" and "standard types" (the later corresponding to our $!A$ types, typing session initiation channels). In our interpretation, a session may either terminate ($\mathbf{1}$), or become a replicated server ($!A$), which is more general and uniform, and a natural consequence of the logical interpretation. Concerning parallel composition, usually two rules can be found, one corresponding to the cancellation of two dual session endpoints (a name restriction rule), and another corresponding to independent parallel composition, also present in most linear type systems for mobile processes. In our case, cut combines both principles, and the following rule is derivable:

$$\frac{\Gamma; \Delta \vdash P :: -{:}\mathbf{1} \quad \Gamma; \Delta' \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash P \mid Q :: T}\ (\mathsf{comp})$$

A consequence of the logical composition rules cut and $\mathsf{cut}^!$ is that typing intrinsically enforces global progress, unlike with traditional session type systems [18,13], which do not ensure progress in the presence of multiple open sessions, as we do here. Techniques to ensure progress in sessions, but building on extraneous devices such as well-founded orderings on events, have been proposed [20,12]. It would be interesting to further compare the various approaches, as far as process typability is concerned.

Channel "polarities" are captured in our system by the left-right distinction of sequents, rather than by annotations on channels (cf. x^+, x^-). Session and linear type systems [21,18,13] also include a typing rule for output of the form

$$\frac{\Gamma; \Delta \vdash P :: x{:}C}{\Gamma; \Delta, y{:}A \vdash x\langle y\rangle.P :: x{:}A \otimes C}$$

In our case, an analogous rule may be derived by $\otimes$R and the copycat construction, where a "proxy" for the free name y, bidirectionally copying behavior A, is linked to z.

$$\frac{\Gamma; \Delta \vdash P :: x{:}C}{\Gamma; \Delta, y{:}A \vdash (\nu z)x\langle z\rangle.(id_A(y,z) \mid P) :: x{:}A \otimes C}$$

The copycat $id_A(y,z)$ plays the role of the "link" processes of [23,7]. Notice that in our case the definition of the "link" is obtained for free by the interpretation of identity axioms (Proposition 4.2). The two processes can be shown to be behaviorally equivalent, under an adequate notion of observational equivalence, as in [7].

We now elaborate on the example of Section 3, in order to illustrate sharing and session initiation. Consider now a different client, that picks the "buy" rather than the "quote" operation, and the corresponding composed system.

$$\begin{array}{ll} \mathit{BClntBody}_s & \triangleq s.\mathtt{inl}; (\boldsymbol{\nu}\mathit{cof})s\langle \mathit{cof}\rangle.(\boldsymbol{\nu}\mathit{pin})s\langle \mathit{pin}\rangle.s(\mathit{rc})\mathbf{0} \\ \mathit{BSimple} & \triangleq (\boldsymbol{\nu}s)(\mathit{SrvBody}_s \mid \mathit{BClntBody}_s) \end{array}$$

We have the typings $\cdot; s{:}\mathit{ServerProto} \vdash \mathit{BClntBody}_s :: -{:}\mathbf{1}$ and $\cdot; \cdot \vdash \mathit{BSimple} :: -{:}\mathbf{1}$.

In these examples, there is a single installed pair client-server, where the session is already initiated, and only known to the two partners. To illustrate sharing, we now consider a replicated server. Such a replicated server is able to spawn a fresh session instance for each initial invocation, each one conforming to the general behavior specified by *ServerProto*, and can be typed by !*ServerProto*. Correspondingly, clients must initially invoke the replicated server to instantiate a new session (cf. the Tcopy rule).

$$\begin{array}{llll} \mathit{QClient} & \triangleq (\boldsymbol{\nu}s)c\langle s\rangle.QClntBody_s & \mathit{BClient} & \triangleq (\boldsymbol{\nu}s)c\langle s\rangle.BClntBody_s \\ \mathit{Server} & \triangleq\, !c(s).SrvBody_s & \mathit{SharSys} & \triangleq (\boldsymbol{\nu}c)(\mathit{Server} \mid \mathit{BClient} \mid \mathit{QClient}) \end{array}$$

For the shared server, by T!R, we type $\cdot; \cdot \vdash \mathit{Server} :: c{:}!\mathit{ServerProto}$. We also have, for the clients, by Tcopy the typings $c{:}\mathit{ServerProto}\ ;\ \cdot \vdash \mathit{BClient} :: -{:}\mathbf{1}$ and $c{:}\mathit{ServerProto}\ ;\ \cdot \vdash \mathit{QClient} :: -{:}\mathbf{1}$. By (comp), T!L, and Tcut we obtain the intended typing for the whole system: $\cdot; \cdot \vdash \mathit{SharSys} :: -\ :\ \mathbf{1}$. Notice how the session instantiation protocol is naturally explained by the logical interpretation of the ! operator.

7 Related Work and Conclusions

We have established a tight correspondence between a session-based type discipline for the π-calculus and intuitionistic linear logic: typing rules correspond to dual intuitionistic linear sequent calculus proof rules, moreover process reduction may be simulated in a type preserving way by proof conversions and reductions, and *vice versa*. As a result, we obtain the subject reduction property, from which session fidelity follows. Our basic typing discipline intrinsically ensures global progress, beyond the restricted "progress on a single session" property obtained in pure session type systems.

Other works have investigated π-calculus models of linear logic proofs. Bellin and Scott [5] establish a mapping from linear logic proofs to a variant of the π-calculus and some connections between proof reduction and π-calculus reduction. However, this mapping results in complex encodings, so that their system could hardly be considered a type assignment system for processes, which has been achieved in this work. Moreover, no relation between behavioral descriptions and logical propositions was identified, as put by the authors: "[our encodings] have less to do with logic than one might think, they are essentially only about the abstract pluggings in proof structures". A realizability interpretation for a linear logic augmented with temporal modalities (cf. Hennessy-Milner) was proposed in [4], also based on a π-calculus variant. A recent related development is [17], where a correspondence between (independently formulated) proof nets and an IO-typed π-calculus is established. In our case, the type system and the logic proof system are exactly the same, and we reveal a direct connection between pure linear logic propositions and behavioral types on π-calculus, that covers all (both shared and linear) features of finite session types. A development of session types as linear process types (in the sense of [21]) is presented in [15], where linearity and sharing are expressed by special annotations, unrelated to a linear logic interpretation.

We have also analyzed the relation between our type discipline and (finite, deadlock-free) session types. It is important to notice that our interpretation does not require locality for session (linear) channels (under which only the output capability of names could be transmitted), which seems required in other works on linearity for π-calculi (*e.g.*, [26]). On the other hand, our intuitionistic discipline enforces locality of shared channels, which, quite interestingly, seems to be the sensible choice for distributed implementations of sessions. Interesting related topics would be the accommodation of recursive types, logical relations [8], and the characterization of observational equivalences under our typing discipline. In particular, we expect that all conversions (including commuting conversions) between DILL derivations correspond to observational equivalences on our typed π-calculus.

One important motivation for choosing a purely logical approach to typing is that it often suggests uniform and expressive generalizations. In ongoing work, we have also established an explicit relationship between session-based concurrency and functional computation where in both cases determinacy (no races) and progress (deadlock-freedom) are expected features. In particular, we have been investigating new encodings of λ-calculi into the π-calculus that arise from translations from DILL natural deduction into sequent calculus. We also believe that dependent generalizations of our system of simple linear types, perhaps along the lines of LLF [9] or CLF [25], may be able to capture many additional properties of communication behavior in a purely logical manner. Already, some systems of session types have dependent character, such as [6] that, among other properties, integrates correspondence assertions into session types.

Acknowledgments. To FCT/MCTES (INTERFACES NGN44), and the ICTI at Carnegie-Mellon. Thanks also to Bernardo Toninho, Nobuko Yoshida, and Andre Platzer.

References

1. Abramsky, S.: Computational Interpretations of Linear Logic. TCS 111(1&2) (1993)
2. Andreoli, J.-M.: Logic Programming with Focusing Proofs in Linear Logic. Journal of Logic and Computation 2(3), 197–347 (1992)

3. Barber, A., Plotkin, G.: Dual Intuitionistic Linear Logic. Technical Report LFCS-96-347, Univ. of Edinburgh (1997)
4. Beffara, E.: A Concurrent Model for Linear Logic. ENTCS 155, 147–168 (2006)
5. Bellin, G., Scott, P.: On the π-Calculus and Linear Logic. TCS 135, 11–65 (1994)
6. Bonelli, E., Compagnoni, A., Gunter, E.L.: Correspondence Assertions for Process Synchronization in Concurrent Communications. J. of Func. Prog. 15(2), 219–247 (2005)
7. Boreale, M.: On the Expressiveness of Internal Mobility in Name-Passing Calculi. Theoretical Computer Science 195(2), 205–226 (1998)
8. Caires, L.: Logical semantics of types for concurrency. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 16–35. Springer, Heidelberg (2007)
9. Cervesato, I., Pfenning, F.: A Linear Logical Framework. Inf. & Comput. 179(1) (2002)
10. Chang, B.-Y.E., Chaudhuri, K., Pfenning, F.: A Judgmental Analysis of Linear Logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University (2003)
11. Dezani-Ciancaglini, M., de' Liguoro, U.: Sessions and Session Types: an Overview. In: 6th Intl. Workshop on Web Services and Formal Methods WS-FM 2009. LNCS. Springer, Heidelberg (2010)
12. Dezani-Ciancaglini, M., de' Liguoro, U., Yoshida, N.: On Progress for Structured Communications. In: Barthe, G., Fournet, C. (eds.) TGC 2007 and FODO 2008. LNCS, vol. 4912, pp. 257–275. Springer, Heidelberg (2008)
13. Gay, S., Hole, M.: Subtyping for Session Types in the Pi Calculus. Acta Informatica 42(2-3), 191–225 (2005)
14. Girard, J.-Y., Lafont, Y.: Linear Logic and Lazy Computation. In: Ehrig, H., Kowalski, R.A., Levi, G., Montanari, U. (eds.) TAPSOFT 1987 and CFLP 1987. LNCS, vol. 250, pp. 52–66. Springer, Heidelberg (1987)
15. Giunti, M., Vasconcelos, V.T.: A Linear Account of Session Types in the Pi-Calculus. In: Gastin, P., Laroussinie, F. (eds.) 21st International Conference on Concurrency Theory, Concur 2010. Springer, Heidelberg (2010)
16. Honda, K.: Types for Dyadic Interaction. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 509–523. Springer, Heidelberg (1993)
17. Honda, K., Laurent, O.: An Exact Correspondence between a Typed pi-calculus and Polarised Proof-Nets. Theoretical Computer Science (to appear, 2010)
18. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
19. Hyland, J.M.E., Luke Ong, C.-H.: Pi-Calculus, Dialogue Games and PCF. In: WG2.8 Conference on Functional Programming Languages, pp. 96–107 (1995)
20. Kobayashi, N.: A Partially Deadlock-Free Typed Process Calculus. ACM Tr. Progr. Lang. Sys. 20(2), 436–482 (1998)
21. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the Pi-Calculus. In: 23rd Symp. on Principles of Programming Languages, POPL 1996, pp. 358–371. ACM, New York (1996)
22. Milner, R.: Functions as processes. Math. Struc. in Computer Sciences 2(2), 119–141 (1992)
23. Sangiorgi, D.: Pi-Calculus, Internal Mobility, and Agent Passing Calculi. Theoretical Computer Science 167(1&2), 235–274 (1996)
24. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. CUP, Cambridge (2001)
25. Watkins, K., Cervesato, I., Pfenning, F., Walker, D.: Specifying properties of concurrent computations in CLF. In: Schürmann, C. (ed.) 4th Intl. Workshop on Logical Frameworks and Meta-Languages (LFM 2004), Cork, Ireland, July 2004. ENTCS, vol. 199 (2004)
26. Yoshida, N., Honda, K., Berger, M.: Linearity and Bisimulation. J. Logic and Algebraic Programming 72(2), 207–238 (2007)