Linear Logical Relations for Session-Based Concurrency

Jorge A. Pérez[1], Luís Caires[1], Frank Pfenning[2], and Bernardo Toninho[1,2]

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

Abstract. In prior work we proposed an interpretation of intuitionistic linear logic propositions as session types for concurrent processes. The type system obtained from the interpretation ensures fundamental properties of session-based typed disciplines—most notably, type preservation, session fidelity, and global progress. In this paper, we complement and strengthen these results by developing a theory of logical relations. Our development is based on, and is remarkably similar to, that for functional languages, extended to an (intuitionistic) linear type structure. A main result is that well-typed processes always terminate (strong normalization). We also introduce a notion of observational equivalence for session-typed processes. As applications, we prove that all proof conversions induced by the logic interpretation actually express observational equivalences, and explain how type isomorphisms resulting from linear logic equivalences are realized by coercions between interface types of session-based concurrent systems.

1 Introduction

Modern computing systems rely heavily on the concurrent communication of distributed software artifacts. Hence, to a large extent, guaranteeing their correctness amounts to ensuring consistent dialogues between these artifacts—an extremely challenging task given the complex interaction patterns they usually feature. *Session-based concurrency* has consolidated as a foundational approach to communication correctness: dialogues between participants are structured into *sessions*, the basic units of communication; descriptions of the interaction patterns are then abstracted as *session types* [11], which are statically checked against specifications. These specifications are usually given in the π-calculus [16], so we obtain *processes* communicating through so-called session channels connecting exactly two subsystems. The discipline of session types ensures session protocols in which actions always occur in dual pairs: when one partner sends, the 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. While *concurrency* arises in the simultaneous execution of sessions, *mobility* is present in the exchange of session and server names.

In session-based concurrency, typing disciplines usually guarantee communication correctness via (forms of) *subject reduction* and *progress* properties. The former states that well-typed processes always evolve to well-typed processes (a *safety* property); the latter says that well-typed processes will never run into a stuck state (a *liveness* property). In addition to ensure that sets of interactions adhere to their prescribed behavior, it is sensible to require such interactions to be *finite*: while from a global perspective systems are meant to run forever, at a local level we would like participants which always

respond within a finite amount of time, and never engage into infinite internal computations. *Termination* (more commonly known as *strong normalization* in the functional setting) is indeed a most desirable liveness property; in session-based concurrency, it may substantially improve the correctness guarantees provided by subject reduction and progress. Ensuring termination in concurrent calculi, however, is known to be hard: in (variants of) the π-calculus, proofs require heavy constraints on the language and/or its types, often relying on ad-hoc machineries (see [8] for a survey).

In the first part of this paper, we study termination in session-based concurrency. The starting point is our interpretation of (intutionistic) linear logic propositions as session types [4], which has provided the first purely logical account of session types. In the interpretation, types are assigned to names (denoting communication channels) and describe their session protocol. This way, an object of type $A \multimap B$ denotes a session that first inputs a session channel of type A, and then behaves as B—another interactive behavior. An object of type $A \otimes B$ denotes a session that first sends a session channel of type A and then behaves as B. The $!A$ type is interpreted as a type of a shared server for sessions of type A. The additive product and sum are interpreted as branch and choice session type operators, respectively. The type system distinguishes two kinds of type environments: a *linear* part Δ and an *unrestricted* part Γ, where weakening and contraction principles hold for Γ but not for Δ. A type judgment is then of the form $\Gamma; \Delta \vdash P :: z{:}C$, with Γ, Δ, and $z{:}C$ having pairwise disjoint domains. We refer to $\Gamma; \Delta$ and $z{:}C$ as the left- and right-hand side typings, respectively. Such a judgment asserts: process P implements session C along channel z, provided it is placed in an environment offering the sessions declared in Γ and Δ. The classic duality of session types is retained via the multiplicative/additive nature of linear logic propositions. This way, e.g., $\otimes$ and $\multimap$ are dual in that *using* a session of one type (in the left-hand side typing) is equivalent to *implementing* a type of the other (in the right-hand side typing).

The interpretation establishes a tight correspondence between session types for the π-calculus and intuitionistic linear logic: typing rules correspond to linear sequent calculus proof rules and, moreover, process reduction may be simulated by proof conversions and reductions, and vice versa. As a result, we obtain subject reduction from which session fidelity follows. The type system ensures global progress, beyond the restricted progress on a single session property obtained in pure session type systems. Examples illustrating the expressiveness of the type system can be found in [5, 4].

Our main contribution is a simple theory of logical relations for session types. The method of logical relations has proved to be extremely productive in the functional setting; in fact, properties such as termination, various forms of equivalence, confluence, parametricity can be established via logical relations. In this presentation, we use logical relations to prove termination for session-typed processes. Although our interpretation assigns types to names (and not to terms, as in the typed λ-calculus), quite remarkably, we are able to define *linear* logical relations which are truly defined on the structure of types—as in logical relations for the typed λ-calculus [23, 24]. A salient aspect of our proof is that it closely follows the principles of the (linear) type system. As hinted at above, this is in sharp contrast with known proofs of termination in the π-calculus. To our knowledge, ours is the first proof of termination of its kind in the context of session-based concurrency.

Certifying termination of session-typed interacting programs is very important in practice. In server-client interactions, for instance, it is critical for clients to be sure that running some piece of code provided by a server (say, code embedded in web pages of a cloud application) will not cause it to get stuck indefinitely (as in a denial-of-service attack, or just due to some bug). Furthermore, strengthening session-based type disciplines with termination guarantees should be highly beneficial for the increasingly growing number of implementations (libraries, programming language extensions) based on session types foundations—see, e.g., [12, 17, 20].

In the second part of the paper, we present two applications of the basic theory, which bear witness to its complementarity with the other properties derived from the interpretation. The applications rely on a notion of *typed observational equivalence*, which we define following the intuitive meaning of type judgements. The first application concerns the *proof conversions* induced by the logic interpretation. In [4] a set of such conversions was shown to correspond to either structural congruence or reduction in the π-calculus. The conversions we study here (not considered in [4]) cannot be explained similarly: they induce forms of "prefix commutation" on typed processes which appear rather counterintuitive. We prove *soundness* of the proof conversions with respect to the observational equivalence, i.e., processes induced by proof conversions are shown to be observationally equivalent. This result thus elegantly explains subtle forms of causality that arise in the (interleaved) execution of concurrent sessions. In our second application, we explain how *type isomorphisms* resulting from linear logic equivalences are realized by coercions between interface types of session-based concurrent systems. We provide a simple behavioral characterization of these isomorphisms, by relying on typed observational equivalence. Type isomorphisms can be seen as a validation of our interpretation with respect to basic linear logic principles. For instance, the apparent asymmetry in the interpretation of $A \otimes B$ is clarified here via an appropriate isomorphism. The two applications thus shed further light on the relationship between linear logic propositions and structured communications. Termination is central to both of them, intuitively because in the bisimulation game strong transitions can be matched by weak transitions which are always finite.

The rest of the paper is structured as follows. Section 2 presents our process model, a synchronous π-calculus with guarded choice. Section 3 recalls the type system derived from the logical interpretation and main results from [4]. Section 4 presents linear logical relations and the termination result. Section 5 introduces a typed observational equivalence for processes. Section 6 discusses soundness of proof conversions and type isomorphisms. Section 7 discusses related work and Section 8 collects final remarks.

2 Process Model: Syntax and Semantics

We introduce the syntax and operational semantics of the synchronous π-calculus [22] 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*

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

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 name y on x and proceed as P), $x(y).P$ (receive a name z on x and proceed as P with parameter y replaced by z), and $!x(y).P$ which denotes replicated (persistent) input. The forwarding construct $[x \leftrightarrow y]$ equates names x and y; it is a primitive representation of a copycat process, akin to the link processes used in internal mobility encodings of name passing [3]. Also, this construct allows for a simple identity axiom in the type system [25]. The remaining three operators define a minimal labeled choice mechanism, comparable to the n-ary branching constructs found in standard session π-calculi (see, e.g., [11]). 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 name y is binding, with scope P. The set of *free names* of a process P is denoted $\mathit{fn}(P)$. 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 capture-avoiding substitution of x for y in P. While *structural congruence* expresses basic identities on the structure of processes, *reduction* expresses the behavior of processes.

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

$$
\begin{array}{ll}
P \mid \mathbf{0} \equiv P & P \equiv_\alpha Q \Rightarrow P \equiv Q \\
P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\
(\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0} & x \notin \mathit{fn}(P) \Rightarrow P \mid (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \mid Q) \\
(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P & [x \leftrightarrow y] \equiv [y \leftrightarrow x]
\end{array}
$$

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

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

By definition, reduction is closed under $\equiv$. It specifies the computations a process performs on its own. To characterize the interactions of a process with its environment, we extend the early transition system for the π-calculus [22] with labels and transition rules for the choice and forwarding constructs. A transition $P \xrightarrow{\alpha} Q$ denotes that P may evolve to Q by performing the action represented by label α. Labels are given by

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

Actions are input $x(y)$, the left/right offers $x.\mathsf{inl}$ and $x.\mathsf{inr}$, 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 $\overline{x.\mathsf{inl}}$ and $\overline{x.\mathsf{inr}}$. 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 τ. 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 $\mathit{fn}(\alpha)$ and $\mathit{bn}(\alpha)$ of free and bound names, respectively, as usual. We denote by $s(\alpha)$ the subject of α (e.g., x in $x\langle y\rangle$).

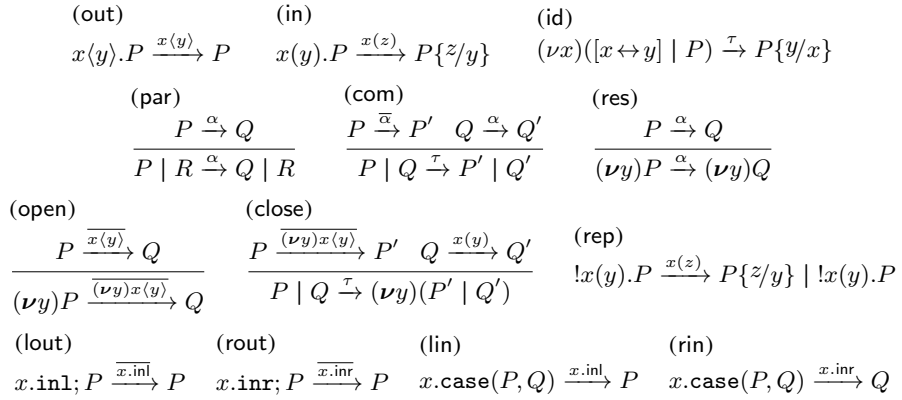

Fig. 1. π-calculus Labeled Transition System.

Definition 2.4 (Labeled Transition System). *The relation* labeled transition *($P \xrightarrow{\alpha} Q$) is defined by the rules in Fig. 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 write $\rho_1\rho_2$ for the composition of relations ρ_1, ρ_2. Weak transitions are defined as usual: we write $\Longrightarrow$ for the reflexive, transitive closure of $\xrightarrow{\tau}$. Given $\alpha \neq \tau$, notation $\stackrel{\alpha}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$ and $\stackrel{\tau}{\Longrightarrow}$ stands for $\Longrightarrow$. We recall some basic facts about reduction, structural congruence, and labeled transition: closure of labeled transitions under structural congruence, and coincidence of τ-labeled transition and reduction [22]: (1) if $P \equiv \xrightarrow{\alpha} Q$ then $P \xrightarrow{\alpha} \equiv Q$, and (2) $P \to Q$ if and only if $P \xrightarrow{\tau} \equiv Q$.

3 Session Types as Dual Intutionistic Linear Logic Propositions

As anticipated in the introduction, the type structure coincides with intuitionistic linear logic [10, 2], omitting atomic formulas and the additive constants $\top$ and $\mathbf{0}$.

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

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

Types are assigned to (channel) names, and are interpreted as a form of session types; an assignment $x{:}A$ enforces the use of name x according to discipline A. $A \otimes B$ types a session channel that first performs an output to its partner (sending a session channel of type A) before proceeding as specified by B. Similarly, $A \multimap B$ types a session channel that first performs an input from its partner (receiving a session channel of type A) before proceeding as specified by B. Type $\mathbf{1}$ means that the session terminated, no further interaction will take place on it; names of type $\mathbf{1}$ may still be passed around in sessions, as opaque values. $A \& B$ types a session channel that offers its partner a choice between an A behavior ("left" choice) and a B behavior ("right" choice). Dually, $A \oplus B$ types a session that either selects "left" and then proceeds as specified by A, or

$$\frac{}{\Gamma; x{:}A \vdash [x \leftrightarrow z] :: z{:}A}\ (\mathsf{Tid}) \quad \frac{\Gamma; \Delta \vdash P :: T}{\Gamma; \Delta, x{:}\mathbf{1} \vdash P :: T}\ (\mathbf{T1L}) \quad \frac{}{\Gamma; \cdot \vdash \mathbf{0} :: x{:}\mathbf{1}}\ (\mathbf{T1R})$$

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

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

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

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

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

$$\frac{\Gamma; \Delta, x{:}A \vdash P :: T \quad \Gamma; \Delta, x{:}B \vdash Q :: T}{\Gamma; \Delta, x{:}A \oplus B \vdash x.\mathtt{case}(P, Q) :: T}\ (\mathrm{T}\oplus\mathrm{L}) \quad \frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta \vdash Q :: x{:}B}{\Gamma; \Delta \vdash x.\mathtt{case}(P, Q) :: x{:}A \,\&\, B}\ (\mathrm{T}\&\mathrm{R})$$

$$\frac{\Gamma; \Delta, x{:}A \vdash P :: T}{\Gamma; \Delta, x{:}A \,\&\, B \vdash x.\mathtt{inl}; P :: T}\ (\mathrm{T}\&\mathrm{L}_1) \quad \frac{\Gamma; \Delta \vdash P :: x{:}A}{\Gamma; \Delta \vdash x.\mathtt{inl}; P :: x{:}A \oplus B}\ (\mathrm{T}\oplus\mathrm{R}_1)$$

$$\frac{\Gamma; \Delta, x{:}B \vdash P :: T}{\Gamma; \Delta, x{:}A \,\&\, B \vdash x.\mathtt{inr}; P :: T}\ (\mathrm{T}\&\mathrm{L}_2) \quad \frac{\Gamma; \Delta \vdash P :: x{:}B}{\Gamma; \Delta \vdash x.\mathtt{inr}; P :: x{:}A \oplus B}\ (\mathrm{T}\oplus\mathrm{R}_2)$$

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

else selects "right", and then proceeds as specified by B. Type $!A$ types a shared (non-linearized) channel, 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. Two kinds of type environments are subject to different structural properties: a *linear* part Δ and an *unrestricted* part Γ, where weakening and contraction principles hold for Γ but not for Δ. A type judgment is of 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. Such a judgment asserts: P is ensured to safely provide a usage of name z according to the behavior 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 [4], in our case safety ensures that behavior is free of communication errors and deadlock. A client Q that relies on external services and does not provide any is 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 $\Gamma; \Delta$. A system typed as $\Gamma; \Delta \vdash R :: z{:}!A$ represents a shared server. 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.

The rules of our type system πDILL are given in Fig. 2. We use T, S for right-hand side singleton environments (e.g., $z{:}C$). Rule (Tid) defines identity in terms of the forwarding construct. Since in rule (T$\otimes$R) the sent name is always fresh, our typed calculus conforms to an internal mobility discipline [3], without loss of expressiveness. The composition rules (Tcut/Tcut$^!$) follow the "composition plus hiding" principle [1], extended to a name passing setting. Other linear typing rules for parallel composition (as in, e.g., [13]) are derivable—see [4]. As we consider π-calculus terms up to structural congruence, typability is closed under $\equiv$ by definition. πDILL enjoys the usual properties of equivariance, weakening, 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 [22], type-checking is decidable.

Session type constructors thus correspond directly to intuitionistic linear logic connectives. By erasing processes, typing judgments in πDILL correspond to DILL, a sequent formulation of Barber's dual intuitionistic linear logic [2, 6]. Below we only provide some intuitions of this correspondence; see [4] for details.

DILL is 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 $\Gamma; \Delta \vdash C$. 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. The converse also holds: if $\Gamma; \Delta \vdash P :: z{:}A$ is derivable in πDILL there is a derivation D that proves $\Gamma; \Delta \vdash D : A$. This correspondence is made explicit by a translation from faithful proof terms to processes: given $\Gamma; \Delta \vdash D : C$, we write $\hat{D}^z$ for the translation of D such that $\Gamma; \Delta \vdash \hat{D}^z :: z{:}C$. More precisely, we have *typed extraction*: we write $\Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$, 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. As processes are related by structural and computational rules, namely those involved in the definition of $\equiv$ and $\rightarrow$, derivations in DILL are related by structural and computational rules, that express certain sound proof transformations that arise in cut-elimination. Reductions 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. Similarly, structural conversions in DILL correspond to structural equivalences in the π-calculus, since they just change the order of cuts.

We now recall some main results from [4]: *subject reduction* and *progress*.
For any P, define $live(P)$ iff $P \equiv (\nu \widetilde{n})(\pi.Q \mid R)$, for some sequence of names $\widetilde{n}$, a process R, and a *non-replicated* guarded process $\pi.Q$.

Theorem 3.2 (Subject Reduction). *If $\Gamma; \Delta \vdash P{::}z{:}A$ and $P \rightarrow Q$ then $\Gamma; \Delta \vdash Q{::}z{:}A$.*

Theorem 3.3 (Progress). *If $\cdot; \cdot \vdash P{::}z{:}\mathbf{1}$ and $live(P)$ then exists a Q such that $P \rightarrow Q$.*

4 Linear Logical Relations and Termination of Typed Processes

A process P *terminates* (written $P\Downarrow$) if there is no infinite reduction path from P. Here we introduce a theory of *linear* logical relations for session types, and use it to prove that well-typed processes always terminate. The proof can be summarized into two steps:

(i) Definition of a logical predicate on processes, by induction on the structure of types. Processes in the predicate are terminating by definition. (ii) Proof that every well-typed process is in the logical predicate.

We begin by stating an extension to $\equiv$, which will be useful in our developments.

Definition 4.1. *We write $\equiv_!$ for the least congruence relation on processes which results from extending structural congruence $\equiv$ (Def. 2.2) with the following axioms:*

1. $(\nu u)(!u(z).P \mid (\nu y)(Q \mid R)) \equiv_! (\nu y)((\nu u)(!u(z).P \mid Q) \mid (\nu u)(!u(z).P \mid R))$
2. $(\nu u)(!u(y).P \mid (\nu v)(!v(z).Q \mid R)) \equiv_! (\nu v)((!v(z).(\nu u)(!u(y).P \mid Q)) \mid (\nu u)(!u(y).P \mid R))$
3. $(\nu u)(!u(y).Q \mid P) \equiv_! P \quad$ *if* $u \notin fn(P)$

These axioms are called the *sharpened replication axioms* [22] and are known to express sound behavioral equivalences up to strong bisimilarity in our typed setting. Intuitively, (1) and (2) represent principles for the distribution of shared servers among processes, while (3) formalizes the garbage collection of shared servers which cannot be invoked by any process. Notice that $\equiv_!$ was defined in [4] (Def 4.3), and noted $\simeq_s$.

Proposition 4.2. *Let P and Q be well-typed processes.*

1. *If $P \longrightarrow P'$ and $P \equiv_! Q$ then there is Q' such that $Q \longrightarrow Q'$ and $P' \equiv_! Q'$.*
2. *If $P \xrightarrow{\alpha} P'$ and $P \equiv_! Q$ then there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv_! Q'$.*

Proposition 4.3. *If $P\Downarrow$ and $P \equiv_! Q$ then $Q\Downarrow$.*

First Step: The Logical Predicate and its Closure Properties. We define a logical predicate on well-typed processes and establish a few associated closure properties. More precisely, we define a sequent-indexed family of sets of processes (process predicates) so that a set of processes $\mathcal{L}[\Gamma; \Delta \vdash T]$ enjoying certain closure properties is assigned to any sequent $\Gamma; \Delta \vdash T$. The logical predicate is defined by induction on the structure of sequents. The base case, given below, considers sequents with empty left-hand side typing, where we abbreviate $\mathcal{L}[\Gamma; \Delta \vdash T]$ by $\mathcal{L}[T]$. We write $P \not\longrightarrow$ to mean that P cannot reduce; it can perform visible actions, though.

Definition 4.4 (Logical Predicate - Base case). *For any type $T = z{:}A$ we inductively define $\mathcal{L}[T]$ as the set of all processes P such that $P\Downarrow$ and $\cdot; \cdot \vdash P :: T$ and*

$$P \in \mathcal{L}[z{:}\mathbf{1}] \text{ if } \forall P'.(P \Longrightarrow P' \wedge P' \not\longrightarrow) \Rightarrow P' \equiv_! \mathbf{0}$$

$$P \in \mathcal{L}[z{:}A \multimap B] \text{ if } \forall P' y.(P \overset{z(y)}{\Longrightarrow} P') \Rightarrow \forall Q \in \mathcal{L}[y{:}A].(\nu y)(P' \mid Q) \in \mathcal{L}[z{:}B]$$

$$P \in \mathcal{L}[z{:}A \otimes B] \text{ if } \forall P' y.(P \overset{\overline{(\nu y) z\langle y\rangle}}{\Longrightarrow} P') \Rightarrow \exists P_1, P_2.(P' \equiv_! P_1 \mid P_2 \wedge P_1 \in \mathcal{L}[y{:}A] \wedge P_2 \in \mathcal{L}[z{:}B])$$

$$P \in \mathcal{L}[z{:}!A] \text{ if } \forall P'.(P \Longrightarrow P') \Rightarrow \exists P_1.(P' \equiv_! !z(y).P_1 \wedge P_1 \in \mathcal{L}[y{:}A])$$

$$P \in \mathcal{L}[z{:}A \,\&\, B] \text{ if } (\forall P'.(P \overset{z.\mathsf{inl}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{L}[z{:}A]) \wedge (\forall P'.(P \overset{z.\mathsf{inr}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{L}[z{:}B])$$

$$P \in \mathcal{L}[z{:}A \oplus B] \text{ if } (\forall P'.(P \overset{\overline{z.\mathsf{inl}}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{L}[z{:}A]) \wedge (\forall P'.(P \overset{\overline{z.\mathsf{inr}}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{L}[z{:}B])$$

Some comments are in order. First, observe how the definition of $\mathcal{L}[T]$ relies on both reductions and labeled transitions, and the fact that processes in the logical predicate are terminating by definition. Also, notice that the use of $\equiv_!$ in $\mathcal{L}[z{:}\mathbf{1}]$ is justified by the fact that a terminated process may be well the composition of a number of shared servers with no potential clients. Using suitable processes that "close" the derivative of the transition, in $\mathcal{L}[z{:}A{\multimap}B]$ and $\mathcal{L}[z{:}A \otimes B]$ we adhere to the linear logic interpretations for input and output types, respectively. In particular, in $\mathcal{L}[z{:}A \otimes B]$ it is worth observing how $\equiv_!$ is used to "split" the derivative of the transition, thus preserving consistency with the separate, non-interfering nature of the multiplicative conjunction. The definition of $\mathcal{L}[z{:}!A]$ is also rather structural, relying again on the distribution principles embodied in $\equiv_!$. The definition of $\mathcal{L}[z{:}A \,\&\, B]$ and $\mathcal{L}[z{:}A \oplus B]$ are self-explanatory.

Below, we extend the logical predicate to arbitrary typing environments. Observe how we adhere to the principles of rules (Tcut) and (Tcut!) for this purpose.

Definition 4.5 (Logical Predicate - Inductive case). *For any sequent $\Gamma; \Delta \vdash T$ with a non-empty left hand side environment, we define $\mathcal{L}[\Gamma; \Delta \vdash T]$ to be the set of processes inductively defined as follows:*

$$P \in \mathcal{L}[\Gamma; y{:}A, \Delta \vdash T] \text{ if } \forall R \in \mathcal{L}[y{:}A].(\nu y)(R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T]$$
$$P \in \mathcal{L}[u{:}A, \Gamma; \Delta \vdash T] \text{ if } \forall R \in \mathcal{L}[y{:}A].(\nu u)(!u(y).R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T]$$

We often rely on the following alternative characterization of the sets $\mathcal{L}[\Gamma; \Delta \vdash T]$.

Definition 4.6. *Let $\Gamma = u_1{:}B_1, \ldots, u_k{:}B_k$, and $\Delta = x_1{:}A_1, \ldots, x_n{:}A_n$ be a non-linear and a linear typing environment, resp. Letting $I = \{1, \ldots, k\}$ and $J = \{1, \ldots, n\}$, we define the sets of processes $\mathcal{C}_\Gamma$ and $\mathcal{C}_\Delta$ as:*

$$\mathcal{C}_\Gamma \stackrel{def}{=} \Big\{ \prod_{i \in I} !u_i(y_i).R_i \mid R_i \in \mathcal{L}[y_i{:}B_i] \Big\} \qquad \mathcal{C}_\Delta \stackrel{def}{=} \Big\{ \prod_{j \in J} Q_j \mid Q_j \in \mathcal{L}[x_j{:}A_j] \Big\}$$

Because of the rôle of left-hand side typing environments, processes in $\mathcal{C}_\Gamma$ and $\mathcal{C}_\Delta$ are then logical representatives of the behavior specified by Γ and Δ, respectively.

Proposition 4.7. *Let Γ and Δ be a non-linear and a linear typing environment, resp. Then, for all $Q \in \mathcal{C}_\Gamma$ and for all $R \in \mathcal{C}_\Delta$, we have $Q\Downarrow$ and $R\Downarrow$. Moreover, $Q \not\longrightarrow$.*

The proof of the following lemma is immediate from Definitions 4.5 and 4.6.

Lemma 4.8. *Let $\Gamma; \Delta \vdash P{::}T$, with $\Gamma{=}u_1{:}B_1, \ldots, u_k{:}B_k$ and $\Delta{=}x_1{:}A_1, \ldots, x_n{:}A_n$. We have: $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$ iff $\forall Q \in \mathcal{C}_\Gamma, \forall R \in \mathcal{C}_\Delta, (\nu \widetilde{u}, \widetilde{x})(P \mid Q \mid R) \in \mathcal{L}[T]$.*

The following closure properties will be of the essence in the second step of the proof, when we will show that well-typed processes are in the logical predicate. We first state closure of $\mathcal{L}[T]$ with respect to substitution and structural congruence:

Proposition 4.9. *Let A be a type. If $P \in \mathcal{L}[z{:}A]$ then $P\{x/z\} \in \mathcal{L}[x{:}A]$.*

Proposition 4.10. *Let P, Q be well-typed. If $P \in \mathcal{L}[T]$ and $P \equiv Q$ then $Q \in \mathcal{L}[T]$.*

The next proposition provides a basic liveness guarantee for certain typed processes.

Proposition 4.11. *Let $P \in \mathcal{L}[z{:}T]$ with $T \in \{A \otimes B, A \multimap B, A \oplus B, A \,\&\, B\}$. Then, there exist α, P' such that (i) $P \stackrel{\alpha}{\Longrightarrow} P'$, and (ii) if $T{=}A \otimes B$ then $\alpha = \overline{(\nu y)z\langle y\rangle}$; if $T{=}A \multimap B$ then $\alpha = z(y)$; if $T{=}A \oplus B$ then $\alpha = \overline{z.\mathsf{inr}}$ or $\alpha = \overline{z.\mathsf{inl}}$; if $T{=}A \,\&\, B$ then $\alpha = z.\mathsf{inr}$ or $\alpha = z.\mathsf{inl}$.*

We now extend Proposition 4.10 so as to state closure of $\mathcal{L}[T]$ under $\equiv_!$.

Proposition 4.12. *Let P, Q be well-typed. If $P \in \mathcal{L}[T]$ and $P \equiv_! Q$ then $Q \in \mathcal{L}[T]$.*

We now state *forward* and *backward* closure of the logical predicate with respect to reduction; these are typical ingredients in the method of logical relations.

Proposition 4.13 (Forward Closure). *If $P \in \mathcal{L}[T]$ and $P \longrightarrow P'$ then $P' \in \mathcal{L}[T]$.*

Proposition 4.14 (Backward Closure). *If for all P_i such that $P \longrightarrow P_i$ we have $P_i \in \mathcal{L}[T]$ then $P \in \mathcal{L}[T]$.*

The final closure property concerns parallel composition of processes:

Proposition 4.15 (Weakening). *Let P, Q be processes such that $P \in \mathcal{L}[T]$ and $Q \in \mathcal{L}[-{:}\mathbf{1}]$. Then, $P \mid Q \in \mathcal{L}[T]$.*

Second Step: Well-typed Processes are in the Logical Predicate. We now prove that well-typed processes are in the logical predicate. Because of the definition of the predicate, termination of well-typed processes will follow as a consequence.

Lemma 4.16. *Let P be a process. If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$.*

Proof. By induction on the derivation of $\Gamma; \Delta \vdash P :: T$, with a case analysis on the last typing rule used. We have 18 cases to check; in all cases, we use Lemma 4.8 to show that every $M = (\nu \widetilde{u}, \widetilde{x})(P \mid G \mid D)$ with $G \in \mathcal{C}_\Gamma$ and $D \in \mathcal{C}_\Delta$, is in $\mathcal{L}[T]$. In case (Tid), we use Proposition 4.9 (closure wrt substitution) and Proposition 4.14 (backward closure). In cases (T$\otimes$L), (T$\multimap$L), (Tcopy), (T$\oplus$L), (T$\&$L$_1$), and (T$\&$L$_2$), we proceed in two steps: first, using Proposition 4.13 (forward closure) we show that every M'' such that $M \Longrightarrow M''$ is in $\mathcal{L}[T]$; then, we use this result in combination with Proposition 4.14 (backward closure) to conclude that $M \in \mathcal{L}[T]$. In cases (T$\mathbf{1}$R), (T$\otimes$R), (T$\multimap$R), (T!R), (T$\oplus$R$_1$), and (T$\oplus$R$_2$), we show that M conforms to a specific case of Definition 4.4. Case (T$\mathbf{1}$L) uses Proposition 4.15 (weakening). Cases (T$\otimes$L), (T$\multimap$L), (T$\oplus$L), and (T$\&$L$_1$) use the liveness guarantee given by Proposition 4.11. Cases (Tcopy), (T!L), and (Tcut$^!$) use Proposition 4.10 (closure under $\equiv$). Cases (Tcut), (T$\multimap$R), and (T!R) use Proposition 4.12 (closure under $\equiv_!$). See [18] for details. □

We now state the main result of this section: well-typed processes terminate.

Theorem 4.17 (Termination). *If $\Gamma; \Delta \vdash P :: T$ then $P\Downarrow$.*

Proof. Follows from previously proven facts. By assumption, we have $\Gamma; \Delta \vdash P :: T$. Using this and Lemma 4.16 we obtain $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$. Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$: combining $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$ and Lemma 4.8 gives us $(\nu \widetilde{u}, \widetilde{x})(P \mid G \mid D) \in \mathcal{L}[T]$. By using this, together with Definition 4.4, we infer $(\nu \widetilde{u}, \widetilde{x})(P \mid G \mid D)\Downarrow$. Since Proposition 4.7 ensures that $G\Downarrow$ and $D\Downarrow$, this latter result allows us to conclude $P\Downarrow$. □

5 An Observational Equivalence for Typed Processes

Here we introduce *typed context bisimilarity*, an observational equivalence over typed processes. It is defined contextually, as a binary relation indexed over sequents. Roughly, typed context bisimilarity equates two processes if, once coupled with all of their requirements (as described by the left-hand side typing), they perform the same actions (as described by the right-hand side typing). To formalize this intuition, we rely on a combination of inductive and coinductive arguments. The base case of the definition covers the cases in which the left-hand side typing environment is empty (i.e., the process requires nothing from its context to execute): the bisimulation game is then defined by induction on the structure of the (right-hand side) typing, following the expected behavior in each case. The inductive case covers the cases in which the left-hand side typing environment is not empty: the tested processes are put in parallel with processes implementing the behaviors described in the left-hand side typing.

Below, we use $\mathcal{S}$ to range over sequents of the form $\Gamma; \Delta \vdash T$. In the following, we write $\vdash T$ to stand for $\cdot\,;\cdot \vdash T$. The definition of typed context bisimilarity relies on *type-respecting* relations, which are indexed by sequents $\mathcal{S}$.

Definition 5.1 (Type-respecting relations). *A* type-respecting binary relation *over processes, written $\{\mathcal{R}_{\mathcal{S}}\}_{\mathcal{S}}$, is defined as a family of relations over processes indexed by $\mathcal{S}$. We often write $\mathcal{R}$ to refer to the whole family. We write $\Gamma; \Delta \vdash P\,\mathcal{R}\,Q :: T$ to mean that (i) $\Gamma; \Delta \vdash P :: T$ and $\Gamma; \Delta \vdash Q :: T$, and (ii) $(P, Q) \in \mathcal{R}_{\Gamma;\Delta\vdash T}$.*

Definition 5.2 (Typed Context Bisimilarity). *A symmetric type-respecting binary relation over processes $\mathcal{R}$ is a* typed context bisimulation *if*

Base Cases

Tau $\vdash P\,\mathcal{R}\,Q :: T$ *implies that for all* P' *such that* $P \xrightarrow{\tau} P'$*, there exists a* Q' *such that* $Q \Longrightarrow Q'$ *and* $\vdash P'\,\mathcal{R}\,Q' :: T$

Input $\vdash P\,\mathcal{R}\,Q :: x{:}A \multimap B$ *implies that for all* P' *such that* $P \xrightarrow{x(y)} P'$*, there exists a* Q' *such that* $Q \overset{x(y)}{\Longrightarrow} Q'$ *and for all* R *such that* $\vdash R :: y{:}A$*,* $\vdash (\nu y)(P' \mid R)\,\mathcal{R}\,(\nu y)(Q' \mid R) :: x{:}B$.

Output $\vdash P\,\mathcal{R}\,Q :: x{:}A \otimes B$ *implies that for all* P' *such that* $P \xrightarrow{\overline{(\nu y)x\langle y\rangle}} P'$*, there exists a* Q' *such that* $Q \overset{\overline{(\nu y)x\langle y\rangle}}{\Longrightarrow} Q'$ *and for all* R *such that* $\cdot\,; y{:}A \vdash R :: -{:}\mathbf{1}$*,* $\vdash (\nu y)(P' \mid R)\,\mathcal{R}\,(\nu y)(Q' \mid R) :: x{:}B$.

Replication $\vdash P\,\mathcal{R}\,Q :: x{:}!A$ *implies that for all* P' *such that* $P \xrightarrow{x(z)} P'$*, there exists a* Q' *such that* $Q \overset{x(z)}{\Longrightarrow} Q'$ *and, for all* R *such that* $\cdot\,; y{:}A \vdash R :: -{:}\mathbf{1}$*,* $\vdash (\nu z)(P' \mid R)\,\mathcal{R}\,(\nu z)(Q' \mid R) :: x{:}!A$.

Choice $\vdash P\,\mathcal{R}\,Q :: x{:}A \,\&\, B$ *implies both:*

- *If* $P \xrightarrow{x.\mathsf{inl}} P'$ *then* $\vdash P'\,\mathcal{R}\,Q' :: x{:}A$*, for some* Q' *such that* $Q \overset{x.\mathsf{inl}}{\Longrightarrow} Q'$*; and*
- *If* $P \xrightarrow{x.\mathsf{inr}} P'$ *then* $\vdash P'\,\mathcal{R}\,Q' :: x{:}B$*, for some* Q' *such that* $Q \overset{x.\mathsf{inr}}{\Longrightarrow} Q'$.

Selection $\vdash P\,\mathcal{R}\,Q :: x{:}A \oplus B$ *implies both:*

- *If* $P \xrightarrow{\overline{x.\mathsf{inl}}} P'$ *then* $\vdash P'\,\mathcal{R}\,Q' :: x{:}A$ *for some* Q' *such that* $Q \overset{\overline{x.\mathsf{inl}}}{\Longrightarrow} Q'$*; and*
- *If* $P \xrightarrow{\overline{x.\mathsf{inr}}} P'$ *then* $\vdash P'\,\mathcal{R}\,Q' :: x{:}B$ *for some* Q' *such that* $Q \overset{\overline{x.\mathsf{inr}}}{\Longrightarrow} Q'$.

Inductive Cases

Linear Names $\Gamma; \Delta, y{:}A \vdash P \mathcal{R} Q :: T$ *implies that*
for all R *such that* $\vdash R :: y{:}A$, *then* $\Gamma; \Delta \vdash (\nu y)(P \mid R) \mathcal{R} (\nu y)(Q \mid R) :: T$.

Shared Names $\Gamma, u{:}A; \Delta \vdash P \mathcal{R} Q :: T$ *implies that for all* R *such that* $\vdash R :: z{:}A$, *then* $\Gamma; \Delta \vdash (\nu u)(!u(z).R \mid P) \mathcal{R} (\nu u)(!u(z).R \mid Q) :: T$.

We write $\approx$ *for the union of all typed context bisimulations, and call it* typed context bisimilarity.

In all cases, a strong action is matched with a weak transition. In proofs, we shall exploit the fact that by virtue of Theorem 4.17 such a weak transition is always finite. In the base case, the clauses for input, output, and replication decree the closure of the tested processes with a process R that "complements" the continuation of the tested behavior; observe the very similar treatment for output and replication (where R *depends* on some behavior), and contrast it with that for input (where R *provides* the behavior). Also, notice how all clauses but that for replication are defined coinductively for the tested processes (in the sense that closed evolutions should be in the relation), but inductively on the type indexing the relation—the clause for replication may be thus considered as the only fully coinductive one. Also worth noticing is how the closures defined in such clauses (and those defined by the clauses in the inductive case) follow closely the spirit of (Tcut/Tcut$^!$) rules in the type system.

Definition 5.2 immediately suggests a proof technique for typed context bisimilarity. First, close the processes with representatives of their context, applying repeatedly the inductive case until the left-hand side typing is empty. Then, following the usual co-inductive proof technique, show a type-respecting relation containing the processes obtained in the first step. The following results are useful to realize these intuitions.

We use K, K' to range over *(process) contexts*, i.e., processes with a hole $[\cdot]$. In particular, we use parallel contexts: contexts in which the hole can only occur in parallel.

Definition 5.3. *Let* Γ *and* Δ *be non-empty typing environments. The set of parallel contexts* $\mathcal{K}_{\Gamma;\Delta}$ *is defined by induction on the typing environments as follows:*

$$
\begin{aligned}
K \in \mathcal{K}_{\emptyset;\emptyset} \text{ if } & K = [\cdot] \\
K \in \mathcal{K}_{\Gamma, u:B;\, \Delta} \text{ if } & K \equiv (\nu u)(K' \mid !u(y).R) \quad \text{for some } K' \in \mathcal{K}_{\Gamma;\Delta} \text{ and } \vdash R :: y{:}B \\
K \in \mathcal{K}_{\Gamma;\, \Delta, x:A} \text{ if } & K \equiv (\nu x)(K' \mid S) \quad \text{for some } K' \in \mathcal{K}_{\Gamma;\Delta} \text{ and } \vdash S :: x{:}A
\end{aligned}
$$

Proposition 5.4. *Let* $\Gamma = u_1{:}B_1, \ldots, u_k{:}B_k$ *and* $\Delta = x_1{:}A_1, \ldots, x_n{:}A_n$ *be typing environments. Letting* $I = \{1, \ldots, k\}$ *and* $J = \{1, \ldots, n\}$, *we say that* $K \in \mathcal{K}_{\Gamma;\Delta}$ *if*

$$
K \equiv (\nu \widetilde{u}, \widetilde{x})([\cdot] \mid \prod_{i \in I} !u_i(y_i).R_i \mid \prod_{j \in J} S_j) \quad \text{with } \vdash R_i :: y_i{:}B_i \text{ and } \vdash S_j :: x_j{:}A_j
$$

The following proposition allows us to move from an (inductive) proof under non-empty typing environments Γ, Δ to a (coinductive) proof under empty environments, with pairs of processes within parallel contexts in $\mathcal{K}_{\Gamma;\Delta}$.

Proposition 5.5. $\Gamma; \Delta \vdash P \approx Q :: T$ *implies* $\vdash K[P] \approx K[Q] :: T$, *for every parallel context* $K \in \mathcal{K}_{\Gamma;\Delta}$.

$$
\begin{array}{rl}
(\nu x)(\hat{D} \mid (\nu y)z\langle y\rangle.(\hat{E} \mid \hat{F})) & \simeq_c (\nu y)z\langle y\rangle.((\nu x)(\hat{D} \mid \hat{E}) \mid \hat{F}) \\
(\nu x)(\hat{D} \mid y(z).\hat{E}) & \simeq_c y(z).(\nu x)(\hat{D} \mid \hat{E}) \\
(\nu x)(\hat{D} \mid y.\mathtt{inl}; \hat{E}) & \simeq_c y.\mathtt{inl}; (\nu x)(\hat{D} \mid \hat{E}) \\
(\nu x)(\hat{D} \mid (\nu y)u\langle y\rangle.\hat{E}) & \simeq_c (\nu y)u\langle y\rangle.(\nu x)(\hat{D} \mid \hat{E}) \\
(\nu x)(\hat{D} \mid y.\mathtt{case}(\hat{E}, \hat{F})) & \simeq_c y.\mathtt{case}((\nu x)(\hat{D} \mid \hat{E}), (\nu x)(\hat{D} \mid \hat{F})) \\
(\nu u)((!u(y).\hat{D}) \mid \mathbf{0}) & \simeq_c \mathbf{0} \\
(\nu u)((!u(y).\hat{D}) \mid (\nu z)x\langle z\rangle.(\hat{E} \mid \hat{F})) & \simeq_c (\nu z)x\langle z\rangle.((\nu u)((!u(y).\hat{D}) \mid \hat{E}) \mid (\nu u)((!u(y).\hat{D}) \mid \hat{F})) \\
(\nu u)((!u(y).\hat{D}) \mid y(z).\hat{E}) & \simeq_c y(z).(\nu u)((!u(y).\hat{D}) \mid \hat{E}) \\
(\nu u)((!u(z).\hat{D}) \mid y.\mathtt{inl}; \hat{E}) & \simeq_c y.\mathtt{inl}; (\nu u)((!u(z).\hat{D}) \mid \hat{E}) \\
(\nu u)((!u(z).\hat{D}) \mid y.\mathtt{case}(\hat{E}, \hat{F})) & \simeq_c y.\mathtt{case}((\nu u)((!u(z).\hat{D}) \mid \hat{E}), (\nu u)((!u(z).\hat{D}) \mid \hat{F})) \\
(\nu u)((!u(y).\hat{D}) \mid !x(z).\hat{E}) & \simeq_c !x(z).(\nu u)((!u(y).\hat{D}) \mid \hat{E}) \\
(\nu u)((!u(y).\hat{D}) \mid (\nu y)v\langle y\rangle.\hat{E}) & \simeq_c (\nu y)v\langle y\rangle.(\nu u)((!u(y).\hat{D}) \mid \hat{E})) \\
(\nu w)z\langle w\rangle.(R \mid (\nu y)x\langle y\rangle.(P \mid Q)) & \simeq_c (\nu y)x\langle y\rangle.(P \mid (\nu w)z\langle w\rangle.(R \mid Q)) \\
x(y).z(w).P & \simeq_c z(w).x(y).P
\end{array}
$$

Fig. 3. A sample of process equalities induced by proof conversions

Definition 5.6. *A type-respecting relation $\mathcal{R}$ is an* equivalence *if it enjoys the following three properties:*

- Reflexivity: $\Gamma; \Delta \vdash P :: T$ *implies* $\Gamma; \Delta \vdash P \mathcal{R} P :: T$*;*
- Symmetry: $\Gamma; \Delta \vdash P \mathcal{R} Q :: T$ *implies* $\Gamma; \Delta \vdash Q \mathcal{R} P :: T$*;*
- Transitivity: $\Gamma; \Delta \vdash P \mathcal{R} P' :: T$ *and* $\Gamma; \Delta \vdash P' \mathcal{R} Q :: T$ *imply* $\Gamma; \Delta \vdash P \mathcal{R} Q :: T$.

Proposition 5.7. *$\approx$ is an equivalence relation.*

In our setting, a notion of congruence for type-respecting relations turns out to be quite type-directed: both right- and left-hand side typings are quite explicit on the compositionality properties of processes. Defining such a notion is relatively straightforward: unsurprisingly, it mirrors the structure of the typing rules. For space reasons, we elide the details; see [18] for the definition and proof that $\approx$ is indeed a congruence.

6 Soundness of Proof Conversions and Type Isomorphisms

We use typed context bisimilarity—together with termination, subject reduction, and progress results—to clarify two issues derived from the logical interpretation: soundness of proof conversions and observational characterizations of type isomorphisms.

Soundness of Proof Conversions. Derivations in DILL are related by structural and computational rules that express sound proof transformations that arise in cut-elimination. As mentioned in Section 3 (and fully detailed in [4]), in our interpretation reductions and structural conversions in DILL correspond to reductions and structural congruence in the π-calculus. There is, however, a group of conversions in DILL not considered in [4] and which do not correspond to neither reduction or structural congruence in the process side. We call them *proof conversions*: they induce a congruence on

typed processes, denoted $\simeq_c$. In this section, we show *soundness* of $\simeq_c$ with respect to $\approx$, that is, processes extracted from proof conversions are typed contextually bisimilar.

We illustrate the proof conversions and their associated π-calculus processes; Fig. 3 presents a sample of process equalities extracted from them. Each equality $M \simeq_c N$ is associated to appropriate right- and left-hand side typings; this way, e.g., the last equality in Fig. 3—associated to two applications of rule (T$\otimes$L)—could be stated as

$$\cdot\,; x{:}A \otimes B, z{:}C \otimes D \vdash x(y).z(w).P \simeq_c z(w).x(y).P :: T$$

where A, B, C, D are types and T is a right-hand side typing. For the sake of illustration, however, in Fig. 3 these typings are elided, as we would like to stress on the consequences of conversions on the process side. Proof conversions describe the interplay of two rules in a type-preserving way: regardless of the order in which the rules are applied, they lead to typing derivations with the same right- and left-hand side typings, but with syntactically different processes. We consider two kinds of proof conversions. The first kind captures the interplay of left/right rules with Tcut/Tcut$^!$ rules; the first twelve rows in Fig. 3 are examples (the first five involve (Tcut), the other seven involve (Tcut$^!$)). The second kind captures the interplay of left and right rules with each other; typically they describe type-preserving transformations which commute actions from non-interfering sessions inside a process (the last two rows in Fig. 3 are examples).

Let us comment on the fifth process equality in Fig. 3. It corresponds to the interplay of rules (Tcut) and (T$\oplus$L), under typing assumptions $\Gamma; \Delta_1 \vdash \hat{D} :: x{:}C$, $\Gamma; \Delta_2, y{:}A, x{:}C \vdash \hat{E}{::}T$, and $\Gamma; \Delta_2, y{:}A, x{:}C \vdash \hat{F}{::}T$. Letting $\Delta = \Delta_1, \Delta_2$, we have:

$$\Gamma; \Delta, y{:}A \oplus B \vdash \underbrace{(\nu x)(\hat{D} \,|\, y.\mathtt{case}(\hat{E}, \hat{F}))}_{(1)} \simeq_c \underbrace{y.\mathtt{case}((\nu x)(\hat{D} \,|\, \hat{E}), (\nu x)(\hat{D} \,|\, \hat{F}))}_{(2)} :: T$$

with types T, A, B, and C, linear environments Δ_1, Δ_2, and non-linear environment Γ.

Read from (1) to (2), this conversion can be interpreted as the "promotion" of the choice at y, which causes $\hat{D}$ to get "delayed" as a result. However, such a delay is seen to be only apparent once we examine the individual typing of $\hat{D}$ and the whole typing derivation. The first typing assumption says that $\hat{D}$ is able to offer behavior C at x (a free name in $\hat{D}$), as long as it is placed in a context in which the behaviors described by names in Γ, Δ_1 are available. The left-hand side typing for both processes says that they can offer some behavior T, as long as the behaviors declared in Γ, Δ and behavior $A \oplus B$ at y are provided. Crucially, since x is private to (1), type T cannot correspond to $x{:}C$. That is, even if $\hat{D}$ is at the top-level in (1) its behavior is not immediately available. Also because of the left-hand side typing, we know that (1) and (2) are only able to interact with some selection at y; only then, $\hat{D}$ will be able to interact with either $\hat{E}$ or $\hat{F}$, whose behavior depends on the presence of behavior C at x. A conversion of (1) into (2) could be seen as a "behavioral optimization" if one considers that (2) has only one available prefix, while (1) has two parallel components.

For all proof conversions, the apparent phenomenon of "prefix promotion" induced by proof conversions can be explained along the above lines. In our soundness result (Theorem 6.2 below), the crucial point is capturing the fact that some top-level processes may not be able to *immediately* exercise their behavior (cf. $\hat{D}$ in (1) above). We

use the following notations on type-respecting relations. $\mathcal{I}_{\Gamma;\Delta\vdash T}$ stands for the relation $\{(P,Q)\ :\ \Gamma;\Delta\vdash P :: T,\ \Gamma;\Delta\vdash Q :: T\}$ which collects pairs of processes with identical left- and right-hand side typings. Based on the logical interpretation of types, we introduce a notion of "continuation relation" for pairs of typed processes:

Definition 6.1. *Using $\boxtimes$ to range over $\otimes, \multimap$ and $\boxplus$ to range over $\oplus, \&$, we define the type-respecting relation $\mathcal{W}_{\vdash x:A}$ by induction on the right-hand side typing, as follows:*

$$\mathcal{W}_{\vdash x:\mathbf{1}} = \mathcal{I}_{\vdash x:\mathbf{1}} \qquad \mathcal{W}_{\vdash x:A\boxtimes B} = \mathcal{I}_{\vdash x:B} \cup \mathcal{W}_{\vdash x:B}$$
$$\mathcal{W}_{\vdash x:!A} = \mathcal{I}_{\vdash x:!A} \qquad \mathcal{W}_{\vdash x:A\boxplus B} = \mathcal{I}_{\vdash x:A} \cup \mathcal{W}_{\vdash x:A} \cup \mathcal{I}_{\vdash x:B} \cup \mathcal{W}_{\vdash x:B}$$

This way, e.g., the continuation relation for $\vdash x{:}A \otimes B$ is $\mathcal{I}_{\vdash x:B} \cup \mathcal{W}_{\vdash x:B}$: it contains all pairs typed by $\vdash x{:}B$ (as processes of type $x{:}A \otimes B$ are to be typed by $x{:}B$ after the output action) as well as those pairs in the continuation relation for $x{:}B$.

Theorem 6.2 (Soundness of Proof Conversions). *Let P, Q be processes such that (i) $\Gamma;\Delta \vdash D \rightsquigarrow P :: T$; (ii) $\Gamma;\Delta \vdash E \rightsquigarrow Q :: T$; (iii) $P \simeq_c Q$. Then, $\Gamma;\Delta \vdash P \approx Q :: T$.*

Proof. By coinduction, exhibiting appropriate typed context bisimulations for each proof conversion. In the bisimulation game, we exploit termination of well-typed processes (Theorem 4.17) to ensure that actions can be matched with finite weak transitions, and subject reduction (Theorem 3.2) to ensure type preservation under reductions.

We detail the case for the first proof conversion in Fig. 3—see [18] for other cases. This proof conversion corresponds to the interplay of rules (T⊗R) and (Tcut). We have to show that $\Gamma;\Delta \vdash M \approx N :: z{:}A \otimes B$ where

$$\Delta = \Delta_1, \Delta_2, \Delta_3 \quad \Gamma;\Delta_1 \vdash \hat{D} :: x{:}C \quad \Gamma;\Delta_2, x{:}C \vdash \hat{E} :: y{:}A \quad \Gamma;\Delta_3 \vdash \hat{F} :: z{:}B \ (1)$$
$$M = (\nu x)(\hat{D} \mid (\nu y)z\langle y\rangle.(\hat{E} \mid \hat{F})) \quad N = (\nu y)z\langle y\rangle.((\nu x)(\hat{D} \mid \hat{E}) \mid \hat{F})$$

Using Proposition 5.5, we have to show that for every $K \in \mathcal{K}_{\Gamma;\Delta}$, we have $\vdash K[M] \approx K[N] :: z{:}A \otimes B$. In turn, this implies exhibiting a typed context bisimulation $\mathcal{R}$ containing the pair $(K[M], K[N])$. We define $\mathcal{R} = \mathcal{W}_{\vdash z:A\otimes B} \cup \mathcal{S} \cup \mathcal{S}^{-1}$, with

$$\mathcal{S} = \{(K_1[M'], K_2[N])\ :\ M \Longrightarrow M',\ \ K_1, K_2 \in \mathcal{K}_{\Gamma;\Delta}\}$$

and $\mathcal{W}_{\vdash z:A\otimes B}$ is as in Definition 6.1. Notice that $\mathcal{S}$ is a type-respecting relation indexed by $\vdash z{:}A \otimes B$. In fact, using the typings in (1)—with $\Gamma = \Delta = \emptyset$—and exploiting subject reduction (Theorem 3.2), it can be checked that for all $(P,Q) \in \mathcal{S}$ both $\vdash P :: z{:}A \otimes B$ and $\vdash Q :: z{:}A \otimes B$ can be derived.

We now show that $\mathcal{R}$ is a typed context bisimulation. Pick any $K \in \mathcal{K}_{\Gamma;\Delta}$. Using Proposition 5.4, we can assume $K = (\nu\widetilde{u},\widetilde{x})(K_\Gamma \mid K_\Delta \mid [\cdot])$ where

- $K_\Gamma \equiv \prod_{i\in I} !u_i(y_i).R_i$, with $\vdash R_i :: y_i{:}D_i$, for every $u_i{:}D_i \in \Gamma$;
- $K_\Delta \equiv \prod_{j\in J} S_j$, with $\vdash S_j :: x_j{:}C_j$, for every $x_j{:}C_j \in \Delta$.

Clearly, $(K[M], K[N]) \in \mathcal{S}$, and so it is in $\mathcal{R}$. Now, suppose $K[M]$ moves first: $K[M] \xrightarrow{\alpha} M_1^\star$. We have to find a matching action α from $K[N]$, i.e., $K[N] \stackrel{\alpha}{\Longrightarrow} N_1^\star$. Since $\vdash K[M] :: z{:}A \otimes B$, we have two possible cases for α:

1. Case $\alpha = \tau$. We consider the possibilities for the origin of the reduction:
   (a) $K_\Gamma \xrightarrow{\tau} K'_\Gamma$ and $K[M] \xrightarrow{\tau} K'[M]$. However, this cannot be the case, as by construction K_Γ corresponds to the parallel composition of input-guarded replicated processes which cannot evolve on their own.
   (b) $K_\Delta \xrightarrow{\tau} K'_\Delta$ and $K[M] \xrightarrow{\tau} K'[M]$. Then, for some $l \in J$, $S_l \xrightarrow{\tau} S'_l$:

$$K[M] \xrightarrow{\tau} (\nu \widetilde{u}, \widetilde{x})(K_\Gamma \mid K'_\Delta \mid M) = K'[M] = M_1^\star$$

   Now, context K is the same in $K[N]$. Then K_Δ occurs identically in $K[N]$, and this reduction can be matched by a ***finite*** weak transition (Theorem 4.17):

$$K[N] \Longrightarrow (\nu \widetilde{u}, \widetilde{x})(K_\Gamma \mid K''_\Delta \mid N) = K''[N] = N_1^\star$$

   By subject reduction (Theorem 3.2), $\vdash S'_l :: x_l{:}C_l$; hence, K', K'' are in $\mathcal{K}_{\Gamma;\Delta}$. Hence, the pair $(K'[M], K''[N])$ is in $\mathcal{S}$ (as $M \Longrightarrow M$) and so it is in $\mathcal{R}$.
   (c) $M \xrightarrow{\tau} M'$ and $K[M] \xrightarrow{\tau} K[M']$. Since $M = (\nu x)(\hat{D} \mid (\nu y) z\langle y\rangle.(\hat{E} \mid \hat{F}))$, the only possibility is that there is a $\hat{D}_1$ such that $\hat{D} \xrightarrow{\tau} \hat{D}_1$ and $M' = (\nu x)(\hat{D}_1 \mid (\nu y) z\langle y\rangle.(\hat{E} \mid \hat{F}))$. This way,

$$K[M] \xrightarrow{\tau} (\nu \widetilde{u}, \widetilde{x})(K_\Gamma \mid K_\Delta \mid M') = K[M'] = M_1^\star$$

   We observe that $K[N]$ cannot match this action, but $K[N] \Longrightarrow K[N]$ is a valid weak transition. Hence, $N_1^\star = K[N]$. By subject reduction (Theorem 3.2), we infer that $\vdash K[M'] :: z{:}A \otimes B$. We use this fact to observe that the pair $(K[M'], K[N])$ is included in $\mathcal{S}$. Hence, it is in $\mathcal{R}$.
   (d) There is an interaction between M and K_Γ or between M and K_Δ: this is only possible by the interaction of $\hat{D}$ with K_Γ or K_Δ on names in $\widetilde{u}, \widetilde{x}$. Again, the only possible weak transition from $K[N]$ matching this reduction is $K[N] \Longrightarrow K[N]$, and the analysis proceeds as in the previous case.
2. Case $\alpha \neq \tau$. Then the only possibility, starting from $K[M]$, is an output action of the form $\alpha = \overline{(\nu y) z\langle y\rangle}$. This action can only originate in M:

$$K[M] \xrightarrow{\overline{(\nu y) z\langle y\rangle}} (\nu \widetilde{x}, \widetilde{u})(K_\Gamma \mid K_\Delta \mid (\nu x)(\hat{D} \mid (\nu y)(\hat{E} \mid \hat{F}))) = M_1^\star$$

   Process $K[N]$ can match this action via the following finite weak transition:

$$K[N] \stackrel{\overline{(\nu y) z\langle y\rangle}}{\Longrightarrow} (\nu \widetilde{x}, \widetilde{u})(K'_\Gamma \mid K'_\Delta \mid (\nu y)((\nu x)(\hat{D}' \mid \hat{E}') \mid \hat{F}')) = N_1^\star$$

   Observe how $N_1^\star$ reflects the changes in $K[N]$ due to the possible reductions before and after the output action. By definition of $\approx$ (output case), we consider the composition of $M_1^\star$ and $N_1^\star$ with any V such that $y{:}A \vdash V :: -{:}\mathbf{1}$. Using the typings in (1) and subject reduction (Theorem 3.2), we infer both

$$\vdash M_2^\star = (\nu \widetilde{x}, \widetilde{u})(K_\Gamma \mid K_\Delta \mid (\nu x)(\hat{D} \mid (\nu y)(\hat{E} \mid V \mid \hat{F}))) :: z{:}B$$
$$\vdash N_2^\star = (\nu \widetilde{x}, \widetilde{u})(K'_\Gamma \mid K'_\Delta \mid (\nu y)((\nu x)(\hat{D}' \mid \hat{E}' \mid V) \mid \hat{F}')) :: z{:}B$$

   Hence, the pair $(M_2^\star, N_2^\star)$ is in $\mathcal{W}_{\vdash z:A\otimes B}$ and so it is in $\mathcal{R}$.

Now suppose that $K[N]$ moves first: $K[N] \xrightarrow{\alpha} N_1^\star$. We have to find a matching action α from $K[M]$: $K[M] \stackrel{\alpha}{\Longrightarrow} M_1^\star$. Similarly as before, there are two cases: either $\alpha = \tau$ or $\alpha = (\nu y)z\langle y\rangle$. The former is as detailed before; the only difference is that reductions from $K[N]$ can only be originated in K_Δ; these are matched by $K[M]$ with finite weak transitions originating in both K and in M. We thus obtain pairs of processes in $\mathcal{S}^{-1}$. The analysis for the case for output mirrors the given above and is omitted. □

Type Isomorphisms. In type theory, types A and B are called *isomorphic* if there are morphisms (proofs in our case) π_A of $B \vdash A$ and π_B of $A \vdash B$ which compose to the identity in both ways—see, e.g., [9]. We adapt this notion to our setting, using typed context bisimilarity to account for *isomorphisms* in linear logic. (Below, we write $P^{\langle \widetilde{x}\rangle}$ for a process parametric on a sequence of names $x_1, \ldots, x_n$.)

Definition 6.3 (Isomorphism). *Two types A and B are called* isomorphic*, noted $A \simeq B$, if, for any names x, y, z, there exist processes $P^{\langle x,y\rangle}$ and $Q^{\langle y,x\rangle}$ such that:*
(i) $\cdot ; x{:}A \vdash P^{\langle x,y\rangle} :: y{:}B$; *(ii)* $\cdot ; y{:}B \vdash Q^{\langle y,x\rangle} :: x{:}A$;
(iii) $\cdot ; x{:}A \vdash (\nu y)(P^{\langle x,y\rangle} \mid Q^{\langle y,z\rangle}) \approx [x \leftrightarrow z] :: z{:}A$; *and*
(iv) $\cdot ; y{:}B \vdash (\nu x)(Q^{\langle y,x\rangle} \mid P^{\langle x,z\rangle}) \approx [y \leftrightarrow z] :: z{:}B$.

Thus, intuitively, if A, B are service specifications then by establishing $A \simeq B$ one can claim that having A is as good as having B, because we can build one from the other using an isomorphism. Isomorphisms in linear logic can then be used to simplify/transform service interfaces in the π-calculus. They can also help validating our interpretation with respect to basic linear logic principles. As an example, let us consider multiplicative conjunction $\otimes$. A basic linear logic principle is $A \otimes B \vdash B \otimes A$. Our interpretation of $A \otimes B$ may appear asymmetric as, in general, a channel of type $A \otimes B$ is not typable by $B \otimes A$. Theorem 6.4 below states the symmetric nature of $\otimes$ as a type isomorphism: symmetry is realized by a process which *coerces* any session of type $A \otimes B$ to a session of type $B \otimes A$. Other sensible isomorphisms, such as, e.g., $(A \oplus B) \multimap C \simeq (A \multimap C) \,\&\, (B \multimap C)$, can be handled similarly.

Theorem 6.4. *Let A, B be any type, as in Def 3.1. Then $A \otimes B \simeq B \otimes A$.*

Proof. We check conditions (i)-(iv) of Def. 6.3 for processes $P^{\langle x,y\rangle}, Q^{\langle y,x\rangle}$ defined as

$$P^{\langle x,y\rangle} = x(u).(\nu n)y\langle n\rangle.([x \leftrightarrow n] \mid [u \leftrightarrow y])$$
$$Q^{\langle y,x\rangle} = y(w).(\nu m)x\langle m\rangle.([y \leftrightarrow m] \mid [w \leftrightarrow x])$$

Checking (i)-(ii), i.e., $\cdot ; x{:}A \otimes B \vdash P^{\langle x,y\rangle} :: y{:}B \otimes A$ and $\cdot ; y{:}B \otimes A \vdash Q^{\langle y,x\rangle} :: x{:}A \otimes B$ is easy; rule (Tid) ensures that both typings hold for any A, B.
We then show (iii) and (iv). We sketch only the proof of (iii); the proof of (iv) is analogous. Let $M = (\nu y)(P^{\langle x,y\rangle} \mid Q^{\langle y,z\rangle})$ and $N = [x \leftrightarrow z]$; we need to show $\cdot ; x{:}A \otimes B \vdash M \approx N :: z{:}A \otimes B$. By Proposition 5.5, we have to show that for every $K \in \mathcal{K}_{\cdot ; x:A\otimes B}$, we have $\vdash K[M] \approx K[N] :: z{:}A \otimes B$. In turn, this implies exhibiting a typed context bisimulation $\mathcal{R}$ containing $(K[M], K[N])$. Letting $\mathcal{S} = \{(R_1, R_2) : K[M] \Longrightarrow R_1,\ K[N] \Longrightarrow R_2\}$, we set $\mathcal{R} = \mathcal{W}_{\vdash z:A\otimes B} \cup \mathcal{S} \cup \mathcal{S}^{-1}$. Following expected lines, $\mathcal{R}$ can be shown to be a typed context bisimulation. □

7 Related Work

Termination in the π-calculus using logical relations has been studied in [26, 21]. Neither of these works considers session types; hence, the technical details of the logical relations are very different, with semantic interpretations of types relying on constraints on the syntax and the types of processes. Here we started from a well-established type discipline for the π-calculus and showed termination of well-typed processes. In contrast, both [26, 21] follow a somewhat opposite path, and aim at type disciplines that guarantee termination. The interpretation of intuitionistic linear logic as session types allows for intuitive logical relations, truly defined on the structure of types. In this sense, our approach is more principled than in [26, 21], as it is not an adaptation of the method, but rather an instantiation of the method on our canonical linear type structure.

Another interpretation of session types as linear logic propositions is proposed in [7]. It is based on *soft* linear logic [15], and so the exponential "!" is treated following a non canonical discipline that uses two different typing environments. Hence, typing rules and judgements in [7] are rather different from ours. A bound on the length of reductions starting from well-typed-processes is obtained; the proof uses techniques from Implicit Computational Complexity. Notions of observational equivalence and their applications are not addressed in [7]. Although here we do not provide a similar bound, it is remarkable that our proof of termination follows *only* the principles and properties of [4]; in contrast to [7], our proof does not appeal to extraneous technical devices, and preserves a standard, intuitive treatment of "!". This is particularly desirable for extensions/generalizations of our framework, such as the proposed in [25, 19].

Loosely related to typed context bisimilarity is [27], where a form of *linear bisimilarity* is proposed; following a linear type structure, it treats some visible actions as internal actions, thus leading to an equivalence larger than standard bisimilarity which is a congruence. The only work on behavioral equivalences for session-based concurrency we are aware of is [14]. It studies the behavioral theory of a π-calculus with asynchronous session communication and an event inspection primitive for buffered messages. The aim is to capture the distinction between order-preserving communications (inside already established connections) and non-order-preserving communications (outside established connections). Such a behavioral theory accounts for principles for prefix commutation that appear similar to those induced by our proof conversions. However, the origin and the nature of these commutations are quite different. In fact, while in [14] prefix commutation arises from the distinction mentioned above, commutations in our (synchronous) framework are due to causality relations captured by types.

8 Concluding Remarks

By relying on the principles established by an interpretation of linear logic as session types [4], we have introduced a theory of logical relations for session-typed disciplines. Our development is remarkably similar to that for functional languages; although in our setting types are assigned to names (and not to terms), our linear logical relations are defined on the structure of types, relying both on process reductions and labeled transitions. A main application of this theory is a proof that well-typed processes always

terminate. This way, in addition to *safety* properties (nothing bad happens, cf. subject reduction), we have shown that session-typed processes also enjoy an important *liveness* property such as termination. Certifying termination of interacting concurrent systems is indeed important, from foundational and practical standpoints. We developed two applications of these results, which complement the results in [4]. Both of them rely on a novel observational equivalence for typed processes. First, we have shown soundness of *proof conversions* with respect to observational equivalence—an issue left open in [4]. Second, we studied *type isomorphisms* resulting from linear logic equivalences in our setting. The basic properties of the interpretation—especially, the combination of subject reduction and termination—were of the essence in both applications. Ongoing work concerns sound *and* complete axiomatizations of typed context bisilmilarity via proof conversions. Having introduced the method of logical relations for session types, we plan to explore it further for obtaining other results, such as parametricity.

Acknowledgments. This research was supported by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program, under grants INTERFACES NGN-44 / 2009 and SFRH / BD / 33763 / 2009, and CITI. We thank the anonymous reviewers for their useful comments.

References

1. S. Abramsky. Computational interpretations of linear logic. *Theor. Comput. Sci.*, 111:3–57, April 1993.
2. A. Barber. Dual intuitionistic linear logic. Technical report, LFCS-96-347, Univ. of Edinburgh, 1996.
3. M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theor. Comput. Sci.*, 195:205–226, March 1998.
4. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR'2010*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
5. L. Caires, F. Pfenning, and B. Toninho. Towards concurrent type theory. In *Proc. of 7th Workshop on Types in Language Design and Implementation – TLDI'12*, 2012.
6. B.-Y. E. Chang, K. Chaudhuri, and F. Pfenning. A judgmental analysis of linear logic. Technical report, CMU-CS-03-131R, Carnegie Mellon University, 2003.
7. U. Dal Lago and P. Di Giamberardino. Soft session types. In *Proc. of 18th Workshop on Expressiveness in Concurrency – EXPRESS'11*, volume 64 of *EPTCS*, pages 59–73, 2011.
8. R. Demangeon, D. Hirschkoff, and D. Sangiorgi. Mobile processes and termination. In *Semantics and Algebraic Specification*, volume 5700 of *LNCS*. Springer, 2009.
9. R. Di Cosmo. A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):825–838, 2005.
10. J.-Y. Girard and Y. Lafont. Linear logic and lazy computation. In *TAPSOFT'87, Vol.2*, volume 250 of *LNCS*, pages 52–66. Springer, 1987.
11. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.
12. R. Hu, N. Yoshida, and K. Honda. Session-based distributed programming in java. In *Proc. of ECOOP*, volume 5142 of *LNCS*, pages 516–541. Springer, 2008.
13. N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. In *POPL*, pages 358–371, 1996.

14. D. Kouzapas, N. Yoshida, R. Hu, and K. Honda. On asynchronous session semantics. In *Proc. of FMOODS-FORTE'2011*, volume 6722 of *LNCS*, pages 228–243. Springer, 2011.
15. Y. Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180, 2004.
16. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, part I/II. *Inf. Comput.*, 100(1):1–77, 1992.
17. N. Ng, N. Yoshida, O. Pernet, R. Hu, and Y. Kryftis. Safe parallel programming with session java. In *Proc. of COORDINATION*, volume 6721 of *LNCS*, pages 110–126. Springer, 2011.
18. J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear Logical Relations for Session-Based Concurrency (Extended Version), 2012. http://goo.gl/iQVZu.
19. F. Pfenning, L. Caires, and B. Toninho. Proof-carrying code in a session-typed process calculus. In *Proc. of CPP '11*, volume 7086 of *LNCS*, pages 21–36. Springer, 2011.
20. R. Pucella and J. A. Tov. Haskell session types with (almost) no class. In *Proc. of ACM SIGPLAN Symposium on Haskell*, pages 25–36. ACM, 2008.
21. D. Sangiorgi. Termination of processes. *Mathematical Structures in Computer Science*, 16(1):1–39, 2006.
22. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.
23. R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2/3):85–97, 1985.
24. W. W. Tait. Intensional Interpretations of Functionals of Finite Type I. *J. Symbolic Logic*, 32:198–212, 1967.
25. B. Toninho, L. Caires, and F. Pfenning. Dependent session types via intuitionistic linear type theory. In *Proc. of PPDP '11*, pages 161–172, New York, NY, USA, 2011. ACM.
26. N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the pi -calculus. *Inf. Comput.*, 191(2):145–202, 2004.
27. N. Yoshida, K. Honda, and M. Berger. Linearity and bisimulation. *J. Log. Algebr. Program.*, 72(2):207–238, 2007.