Relational Parametricity for Polymorphic Session Types

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

April 2012
CMU-CS-12-108

School of Computer Science[1]
Carnegie Mellon University
Pittsburgh, PA 15213

CITI and Departamento de Informática[2]
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

Abstract

We introduce a theory of polymorphic concurrent processes, which arises from an interpretation of second-order intuitionistic linear logic propositions as polymorphic session types, in the style of the Girard-Reynolds polymorphic λ-calculus. The interpretation naturally generalizes recent discoveries on the correspondence between linear logic propositions and session types. In our proposed theory, polymorphism accounts for the exchange of abstract communication protocols, and dynamic instantiation of heterogeneous interfaces. Well-typed processes enjoy a strong form of subject reduction (type preservation) and global progress, but also termination (strong normalization) and relational parametricity (representation independence). The latter two properties are obtained by adapting proof techniques in the functional setting to linear session types. Furthermore, we discuss a faithful encoding of System F into our polymorphic session calculus.

Support for this research was provided by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program, under grants SFRH / BD / 33763 / 2009 and INTERFACES NGN-44 / 2009, and CITI.

Keywords: Session Types, Polymorphic Concurrent Processes

1 Introduction

This paper develops a logically motivated theory of *impredicative* parametric polymorphism for session types. Our theory enables us to exploit *behavioral genericity* which, to the best of our knowledge, is novel in the realm of session-typed concurrent processes.

In an ongoing research program [8,29,24,9,23,10], we have been investigating logical foundations for session-based concurrency, based on a fresh interpretation of linear logical propositions as session types [19,21], sequent proofs as π-calculus processes, and cut elimination as process communication. The result is a tight correspondence, in the Curry-Howard style; well-typed processes enjoy strong forms of type preservation and global progress [8,9], and are strongly normalizing [23]. The key idea of our interpretation is to endow channel names with types (actually, linear logic propositions) that describe their session protocol. This way, e.g., an assignment $x{:}A \multimap B$ denotes a session x that first *inputs* a name of type A, and then behaves as B on x; dually, $x{:}A \otimes B$ denotes a session that first *outputs* a name of type A and then behaves as B on x. Other linear constructors are also given compatible interpretations; in particular, $!A$ is the type of a shared server offering sessions of type A.

The key novelty in the present work is the uniform extension of the system of [8] with two new kinds of session types, $\forall X.A$ and $\exists X.A$, corresponding to *impredicative* universal and existential quantification over *sessions*. We thus focus on an another ambitious stepping stone of the correspondence: obtaining, by purely logical means, a natural extension of session types with *parametric polymorphism* in the sense of the Girard-Reynolds polymorphic λ-calculus [26,15]. In the extended interpretation, universal and existential quantification correspond to an input and an output of a *session type*, respectively. This is rather remarkable: since session types denote communication protocol behavior, parametric polymorphism in our setting accounts for a rather expressive form of *abstract protocol communication*. As an example, consider the type:

$$\mathsf{AppStore} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !X$$

In our interpretation, this is the session type of a system which first inputs an arbitrary type (say $FlightBroker$); then inputs a shared service of type $\mathsf{api} \multimap FlightBroker$. Each instance of this service yields a session that when provided with the implementation of an API will provide a behavior of type $FlightBroker$; finally becoming a persistent (shared) server of type $FlightBroker$. The above type can be seen as an abstract specification of an application store service, which first receives both a behavioral description and a concrete implementation of such a description, and then offers that behavior persistently to any interested clients. The crucial combination of polymorphism and linearity allows types to specify process behavior in a very fine-grained way: impredicative quantification enforces that the store implementation must be agnostic to the specific behavior of the actual applications it will provide, while linearity allows us to reason precisely about behavior and session usage (e.g., the only way the store can provide the behavior X is by making use of session $\mathsf{api} \multimap X$).

A main research question standing is whether the techniques developed in [8,9,23] scale up to the case of parametric polymorphism, preserving the strong correctness properties well-typed processes enjoy (type preservation, global progress, strong normalization). This paper develops the technical machinery to answer positively to this question. By doing so, we further strengthen the logical foundations for structured communications, and contribute to the study of the connections between logic and models of concurrency (e.g., [1,4,20,3]).

We summarize the structure and technical achievements of this paper:

1. In Section 2, building upon the developments of [8,9], we define a system of polymorphic session-typed processes, and establish its fundamental properties: subject reduction (type preservation, Theorem 2.11) and global progress (Theorem 2.13). We thus extend the results in [8,9] to the much more expressive setting of polymorphic sessions. Examples of polymorphic processes are detailed in Section 3.
2. In Section 4, we prove strong normalization for well-typed polymorphic processes via linear logical relations/reducibility candidates (Theorem 4.16). The proof relies on a natural generalization of the linear logical relations introduced in [23].
3. Section 5 studies the behavioral theory of polymorphic session-typed processes. We prove a result of *relational parametricity* (Theorem 5.16) which allows for precise characterizations of behavior through typing, such as the given above for the application store example. Moreover, we provide a sound and complete characterization of contextual equivalence via logical relations (Theorems 5.19 and 5.20).

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

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

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

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

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

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

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

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

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

$$\frac{\Omega, X; \Gamma; \Delta \vdash D : A}{\Omega; \Gamma; \Delta \vdash \forall\mathsf{R}\ (X.\ D) : \forall X.A}\ \forall\mathsf{R} \qquad \frac{\Omega \vdash B\ \mathsf{prop} \quad \Omega; \Gamma; \Delta, x : A\{B/X\} \vdash D : C}{\Omega; \Gamma; \Delta, x : \forall X.A \vdash \forall\mathsf{L}\ x\ B\ (x.\ D) : C}\ \forall\mathsf{L}$$

$$\frac{\Omega \vdash B\ \mathsf{prop} \quad \Omega; \Gamma; \Delta \vdash D : A\{B/X\}}{\Omega; \Gamma; \Delta \vdash \exists\mathsf{R}\ B\ D : \exists X.A}\ \exists\mathsf{R} \qquad \frac{\Omega, X; \Gamma; \Delta, x{:}A \vdash D : C}{\Omega; \Gamma; \Delta, x : \exists X.A \vdash \exists\mathsf{L}\ x\ \ (X.x.\ D) : C}\ \exists\mathsf{L}$$

Fig. 1. Second-order Intuitionistic Linear Logic - Proofs.

4. Section 6 connects our work with impredicative polymorphism in the functional setting via an encoding of System F which is proved to be faithful (Theorem 6.6).

Section 7 discusses related work, while Section 8 offers some concluding remarks.

2 Polymorphic session types

2.1 Second-order Intuitionistic Linear Logic

This presentation of second-order linear logic consists of a natural extension of a sequent calculus formulation of linear logic known as DILL [2,11]. To handle the second-order quantifiers, we add an additional context Ω to our sequent judgment that keeps track of predicate variables. Thus, our main judgment is of the form $\Omega; \Gamma; \Delta \vdash D : A$, meaning that D is a proof of A, using the assumptions in Δ linearly (i.e., not subject to weakening or contraction), the assumptions in Γ in an unrestricted fashion and where Ω all the higher-order variables in the given proof. We range over variables in Δ with x, y, z and over variables in Γ with u, v, w. Since propositions have binding structure, we define an additional judgment $\Omega \vdash A\ \mathsf{prop}$, meaning that A is a well-formed proposition under the assumptions recorded in Ω. The rules for proofs and propositions are given in Fig. 1 and Fig. 2, respectively.

Theorem 2.1 (Regularity). *If $\Omega; \Gamma; \Delta \vdash D : A$ and $\Omega \vdash A_i\ \mathsf{prop}$ for each assumption $x : A_i \in \Delta$ and $u : A_i \in \Gamma$, then $\Omega \vdash A\ \mathsf{prop}$.*

$$\frac{}{\Omega, X \vdash X \text{ prop}} \qquad \frac{\Omega \vdash A \text{ prop} \quad \Omega \vdash B \text{ prop}}{\Omega \vdash A \multimap B \text{ prop}} \qquad \frac{\Omega \vdash A \text{ prop} \quad \Omega \vdash B \text{ prop}}{\Omega \vdash A \otimes B \text{ prop}} \qquad \frac{\Omega \vdash A \text{ prop} \quad \Omega \vdash B \text{ prop}}{\Omega \vdash A \,\&\, B \text{ prop}}$$

$$\frac{\Omega \vdash A \text{ prop} \quad \Omega \vdash B \text{ prop}}{\Omega \vdash A \oplus B \text{ prop}} \qquad \frac{}{\Omega \vdash \mathbf{1} \text{ prop}} \qquad \frac{\Omega, X \vdash A \text{ prop}}{\Omega \vdash \forall X.A \text{ prop}} \qquad \frac{\Omega, X \vdash A \text{ prop}}{\Omega \vdash \exists X.A \text{ prop}} \qquad \frac{\Omega \vdash A \text{ prop}}{\Omega \vdash\, !A \text{ prop}}$$

Fig. 2. Second-order Intuitionistic Linear Logic - Propositions.

2.2 The Process Model

We consider a synchronous π-calculus extended with binary guarded choice, channel links, and type input and output prefixes. The syntax of processes/types is as follows:

Definition 2.2 (Processes, Session Types). *Given an infinite set Λ of* names (x, y, z, u, v)*, the set of* processes (P, Q, R) *and session types (A, B, C) is defined by*

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

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

The guarded choice mechanism and the channel link construct are as in [8,29,23]. Informally, channel links "re-implement" an ambient session on a different channel name, thus defining a renaming operation (see below). Moreover, channel links allow a simple interpretation of the identity rule. Polymorphism is represented by prefixes for input and output of types, denoting the exchange of abstract communication protocols.

We identify processes 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, and $fn(P)$ for the free names of P. Session types are directly generated from the language of linear propositions. Structural congruence expresses basic identities on the structure of processes, reduction expresses internal behavior of processes, and labelled transitions define interaction with the environment.

Definition 2.3. Structural congruence *($P \equiv Q$), is the least congruence relation on processes such that*

$P \mid \mathbf{0} \equiv P$	*(S**0**)*	$P \equiv_\alpha Q \Rightarrow P \equiv Q$	*(Sα)*
$P \mid Q \equiv Q \mid P$	*(S|C)*	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	*(S|A)*
$(\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0}$	*(S$\boldsymbol{\nu}$**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$\boldsymbol{\nu}\boldsymbol{\nu}$)*	$[x \leftrightarrow y] \equiv [y \leftrightarrow x]$	*(S$\leftrightarrow$)*

Definition 2.4. Reduction *($P \to Q$), is the binary relation on processes defined by:*

$\overline{x}\langle y\rangle.Q \mid x(z).P \to Q \mid P\{y/z\}$	*(RC)*	$x\langle A\rangle.Q \mid x(Y).P \to Q \mid P\{A/Y\}$	*(RTC)*
$x\langle y\rangle.Q \mid\, !x(z).P \to Q \mid P\{y/z\} \mid\, !x(z).P$	*(R!)*	$(\nu x)([x \leftrightarrow y] \mid P) \to P\{y/x\}$	*(R$\leftrightarrow$)*
$x.\mathtt{inl}; P \mid x.\mathtt{case}(Q, R) \to P \mid Q$	*(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$\equiv$)*		

A transition $P \xrightarrow{\alpha} Q$ denotes that P may evolve to Q by performing the action represented by label α. In general, an action α ($\overline{\alpha}$) requires a matching $\overline{\alpha}$ (α) in the environment to enable progress. Labels include: the silent internal action τ, output and bound output actions $\overline{x\langle y\rangle}$ and $\overline{(\nu z)x\langle z\rangle}$, respectively, and input action $x(y)$. Also, they include labels pertaining to the binary choice construct ($x.\mathsf{inl}$, $\overline{x.\mathsf{inl}}$, $x.\mathsf{inr}$, and $\overline{x.\mathsf{inr}}$), and labels describing output and input of types (noted $\overline{x\langle A\rangle}$ and $x(A)$, respectively).

Definition 2.5 (Labeled Transition System). *The relation* labeled transition *($P \xrightarrow{\alpha} Q$) is defined by the rules in Fig. 3, 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 $\overset{\alpha}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$ and $\overset{\tau}{\Longrightarrow}$ stands for $\Longrightarrow$.

$$\text{(out)}\quad \overline{x}\langle y\rangle.P \xrightarrow{\overline{x\langle y\rangle}} P \qquad \text{(in)}\quad x(y).P \xrightarrow{x(z)} P\{z/y\} \qquad \text{(outT)}\quad \overline{x}\langle A\rangle.P \xrightarrow{\overline{x\langle A\rangle}} P \qquad \text{(inT)}\quad x(Y).P \xrightarrow{x(B)} P\{B/Y\}$$

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

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

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

Fig. 3. π-calculus Labeled Transition System.

2.3 Type System

Our type system assigns session types to process communication channels. Our session type language (cf. Definition 2.2) corresponds exactly to second-order linear logic, and our typing rules capture this correspondence in a precise way. We define two judgments: $\Omega; \Gamma; \Delta \vdash P :: x{:}A$ and $\Omega \vdash A$ type. Context Ω keeps track of type variables that can be introduced by the polymorphic type constructors; Γ records persistent sessions $u{:}B$, which can be invoked arbitrarily often along channel u; Δ maintains the sessions $x{:}B$ that can be used exactly once, on channel x. Thus, our main typing judgment states that process P implements a session of type A along channel x, provided it is composed with processes providing sessions linearly in Δ and persistently in Γ, such that the types occurring in the judgment are well-formed according to Ω. Our second judgment, $\Omega \vdash A$ type, defines well-formedness for types: it denotes that A is a well-formed type with free variables registered in Ω. The rules for type well-formedness are straightforward and thus omitted.

The typing rules for our polymorphic session calculus are given in Fig. 4. We use T, S for right-hand-side singleton environments (e.g., $z{:}C$). All the rules pertaining to the propositional fragment are those introduced in [8], extended with context Ω. The new rules explain how to *provide* and *use* sessions of a polymorphic type. We provide a session of universal type $\forall X.A$ by inputing an arbitrary type, bound to X, and proceeding as A, which may bind the type variable X, regardless of what the actual received type is. Using a session of type $\forall X.A$ consists of the output of a type B—well-formed under the current type context Ω—which then warrants the use of the session as $A\{B/X\}$. The existential type is dual: providing an existentially typed session $\exists X.A$ is accomplished by outputting a well-formed type B and then providing a session of type $A\{B/X\}$. Using an existential session $\exists X.A$ denotes inputing a type and then using the session as A, agnostic to what the actual received type can be. Note that in the presence of polymorphism, the identity rule (not present in [8,9], but used in [29,24,23]) is necessary, since it is the only way of typing a session with a type variable. As usual, in the presence of type annotations in binders, type-checking is decidable in our system (we omit these for readability). We are considering π-calculus terms up to structural congruence, and so typability is closed under $\equiv$ by definition. The system enjoys the usual properties of equivariance, weakening and contraction in Γ.

Theorem 2.6 (Coverage). *If $\Omega; \Gamma; \Delta \vdash P :: z{:}A$ then $fn(P) \subseteq \Gamma \cup \Delta \cup \{z\}$*

We write $\Omega \triangleright \Gamma$ (resp. $\Omega \triangleright \Delta$) to denote that the types occurring in Γ (resp. Δ) are well-formed wrt the type variables declared in Ω (resp. Δ).

Theorem 2.7 (Regularity). *If $\Omega; \Gamma; \Delta \vdash P :: z{:}A$, $\Omega \triangleright \Gamma$ and $\Omega \triangleright \Delta$, then $\Omega \vdash A$ type.*

2.4 Correspondence

It is easy to informally see the correspondence between our type system and the presentation of second-order linear logic from the previous section. We make this correspondence precise by defining a mapping from linear logic proof terms to processes. Given $\Omega; \Gamma; \Delta \vdash D : C$ we write $\hat{D}^z$ for the translation of D such that $\Omega; \Gamma; \Delta \vdash \hat{D}^z :: z{:}C$. This

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

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

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

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

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

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

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

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

$$\text{(T}\forall\text{L)}\ \frac{\Omega \vdash B\ \mathsf{type} \quad \Omega;\Gamma;\Delta, x : A\{B/X\} \vdash P :: T}{\Omega;\Gamma;\Delta, x : \forall X.A \vdash x\langle B\rangle.P :: T} \qquad \text{(T}\forall\text{R)}\ \frac{\Omega, X;\Gamma;\Delta \vdash P :: z{:}A}{\Omega;\Gamma;\Delta \vdash z(X).P :: z{:}\forall X.A}$$

$$\text{(T}\exists\text{L)}\ \frac{\Omega, X;\Gamma;\Delta, x{:}A \vdash P :: T}{\Omega;\Gamma;\Delta, x : \exists X.A \vdash x(X).P :: T} \qquad \text{(T}\exists\text{R)}\ \frac{\Omega \vdash B\ \mathsf{type} \quad \Omega;\Gamma;\Delta \vdash P :: x{:}A\{B/X\}}{\Omega;\Gamma;\Delta \vdash x\langle B\rangle.P :: x{:}\exists X.A}$$

Fig. 4. The Type System.

mapping is given in Fig. 5 (an inverse mapping can also be straightforwardly defined, but we omit it for the sake of brevity).

Definition 2.8 (Typed Extraction). *We write $\Omega;\Gamma;\Delta \vdash D \rightsquigarrow P :: z{:}A$, meaning "proof D extracts to P", whenever $\Omega;\Gamma;\Delta \vdash D : A$ and $\Gamma;\Delta \vdash P :: z{:}A$ and $P \equiv \hat{D}^z$.*

2.5 Subject Reduction and Progress

We can show a strong form of subject reduction by showing a simulation between reductions in the typed π-calculus and proof conversions / reductions that arise naturally in proof theory (Theorem 2.11). The proof follows closely that of [8,9], extending it with lemmas that relate process reductions with proof reductions at universal and existential types.

Lemma 2.9. *Assume*

(a) $\Omega;\Gamma;\Delta_1 \Rightarrow P :: x : \forall X.B$ *with* $P \overset{x(A)}{\rightarrow} P'$ *and* $\Omega \vdash A\ \mathsf{type}$

(b) $\Omega;\Gamma;\Delta_2, x : \forall X.B \Rightarrow Q :: z : C$ *with* $Q \overset{\overline{x\langle A\rangle}}{\rightarrow} Q'$ *and* $\Omega \vdash A\ \mathsf{type}$

D	$\rightsquigarrow \hat{D}^z$	D	$\rightsquigarrow \hat{D}^z$
$\mathbf{1}\mathsf{R}$	$\rightsquigarrow 0$	$\oplus\mathsf{R}_2\ E$	$\rightsquigarrow z.\mathsf{inr}; \hat{E}^z$
$\mathbf{1}\mathsf{L}\ x\ D$	$\rightsquigarrow \hat{D}^z$	$\oplus\mathsf{L}\ x\ (x.D)\ (x.E)$	$\rightsquigarrow x.\mathsf{case}(\hat{D}^z, \hat{E}^z)$
$\mathsf{id}\ x$	$\rightsquigarrow [x \leftrightarrow z]$	$\mathsf{cut}\ D\ (x.E)$	$\rightsquigarrow (\nu x)(\hat{D}^x \mid \hat{E}^z)$
$\otimes\mathsf{R}\ D\ E$	$\rightsquigarrow (\nu y)\, z\langle y\rangle.\, (\hat{D}^y \mid \hat{E}^z)$	$!\mathsf{R}\ D$	$\rightsquigarrow !z(y).\, \hat{D}^y$
$\otimes\mathsf{L}\ x\ (y.x.D)$	$\rightsquigarrow x(y).\, \hat{D}^z$	$!\mathsf{L}\ x\ (u.D)$	$\rightsquigarrow \hat{D}^z\{x/u\}$
$\multimap\mathsf{R}\ (y.D)$	$\rightsquigarrow z(y).\, \hat{D}^z$	$\mathsf{copy}\ u\ (y.D)$	$\rightsquigarrow (\nu y)\, u\langle y\rangle.\, \hat{D}^z$
$\multimap\mathsf{L}\ x\ D\ (x.E)$	$\rightsquigarrow (\nu y)\, x\langle y\rangle.\, (\hat{D}^y \mid \hat{E}^z)$	$\mathsf{cut}^!\ D\ (u.E)$	$\rightsquigarrow (\nu u)((!u(y).\, \hat{D}^y) \mid \hat{E}^z)$
$\&\mathsf{R}\ D\ E$	$\rightsquigarrow z.\, \mathsf{case}(\hat{D}^z, \hat{E}^z)$	$\forall\mathsf{R}\ (X.D)$	$\rightsquigarrow z(X).\hat{D}^z$
$\&\mathsf{L}_1\ x\ (x.D)$	$\rightsquigarrow x.\, \mathsf{inl}; \hat{D}^z$	$\forall\mathsf{L}\ x\ A\ (x.D)$	$\rightsquigarrow x\langle A\rangle.\hat{D}^z$
$\&\mathsf{L}_2\ x\ (x.E)$	$\rightsquigarrow x.\, \mathsf{inr}; \hat{E}^z$	$\exists\mathsf{R}\ A\ D$	$\rightsquigarrow z\langle A\rangle.\hat{D}^z$
$\oplus\mathsf{R}_1\ D$	$\rightsquigarrow z.\mathsf{inl}; \hat{D}^z$	$\exists\mathsf{L}\ x\ (X.x.D)$	$\rightsquigarrow x(X).\hat{D}^z$

Fig. 5. Proof D extracts to process $\hat{D}^z$.

Then:

(c) $\Omega; \Gamma; \Delta_1, \Delta_2 \Rightarrow (\boldsymbol{\nu} x)(P' \mid Q') :: z : C$

Proof. By simultaneous induction on D and E. See Appendix A.1 for details. □

Lemma 2.10. *Assume*

(a) $\Omega; \Gamma; \Delta_1 \Rightarrow P :: x : \exists y : \tau.B$ *with* $P \overset{x\langle A\rangle}{\rightarrow} P'$ *and* $\Omega \vdash A$ type

(b) $\Omega; \Gamma; \Delta_2, x : \exists y : \tau.B \Rightarrow Q :: z : C$ *with* $Q \overset{x(A)}{\rightarrow} Q'$ *and* $\Omega \vdash A$ type

Then:

(c) $\Omega; \Gamma; \Delta_1, \Delta_2 \Rightarrow (\boldsymbol{\nu} x)(P' \mid Q') :: z : C$

Proof. By simultaneous induction on D and E. See Appendix A.2 for details. □

Making use of lemmas of the form above, for each session type constructor, we can then show the following strong subject reduction property.

Theorem 2.11 (Subject Reduction). *Let* $\Omega; \Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}A$ *and* $P \rightarrow Q$. *Then there is* E *such that* $D \equiv \Rightarrow \equiv E$ *and* $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z{:}A$

We also establish a global progress property for well-typed processes. Also in this case, the proof is an orthogonal extension from that of [8,9], requiring a series of simple inversion lemmas and a notion of a *live* process. For any P, define

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

where $\pi.Q$ is a *non-replicated* guarded process. We first establish the following contextual progress property, from which Theorem 2.13 follows as a corollary.

Lemma 2.12. *Let* $\Omega; \Gamma; \Delta \vdash D \rightsquigarrow P :: z{:}C$. *If* $\mathit{live}(P)$ *then there is* Q *such that either*

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

Theorem 2.13 (Progress). *If* $\cdot; \cdot \vdash D \rightsquigarrow P :: x{:}\mathbf{1}$ *and* $\mathit{live}(P)$ *then exists* Q *st.* $P \rightarrow Q$.

3 Examples of Polymorphic Processes

We present two simple examples that illustrate some of the expressiveness we obtain via polymorphic sessions. First, and in the line of standard presentations of polymorphic types, we describe a polymorphic implementation of booleans as session-typed processes. Then, we develop the AppStore scenario sketched in the Introduction. Below, for the sake of clarity, we abbreviate bound outputs along x as $\overline{x}\langle y\rangle$ and $\overline{x}\langle y\rangle.([y \leftrightarrow w] \mid P)$ as $x\langle w\rangle.P$. Also, we will consider a straightforward extension of our system with labelled n-ary choice.

Booleans. We develop a server that implements booleans and their primitive operations of conjunction and negation, while keeping the actual representation of booleans abstract using an existential type. As a particular case, we will consider a representation of booleans using the following types and processes:

$$
\begin{array}{lll}
\mathsf{Bool} & \triangleq & \forall X.(!X \multimap (!X \multimap X)) \\
\mathsf{Ptrue}_z & \triangleq & z(X).z(t).z(f).t\langle z\rangle :: z{:}\mathsf{Bool} \\
\mathsf{Pfalse}_z & \triangleq & z(X).z(t).z(f).f\langle z\rangle :: z{:}\mathsf{Bool} \\
\mathsf{Pand}_z & \triangleq & z(u_1).z(u_2).\overline{u_1}\langle x\rangle.x\langle \mathsf{Bool}\rangle.\overline{x}\langle y_1\rangle. \\
 & & \quad (!y_1(v).u_2\langle v\rangle \mid \\
 & & \quad\quad \overline{x}\langle y_2\rangle.(!y_2(v).u_1\langle v\rangle \\
 & & \quad\quad\quad \mid [z \leftrightarrow x])) :: z{:}!\mathsf{Bool} \multimap !\mathsf{Bool} \multimap \mathsf{Bool} \\
\mathsf{Pnot}_z & \triangleq & z(x).z(X).z(t).z(f).x\langle X\rangle.x\langle f\rangle.x\langle t\rangle.[x \leftrightarrow z] :: z{:}\mathsf{Bool} \multimap \mathsf{Bool}
\end{array}
$$

Above, Ptrue_z is a process that encodes truth: along channel z, it will input a type X, two sessions t and f of type $!X$ and will output on the first one (signaling truth). Similarly, process Pfalse_z encodes falsehood: it performs the same three inputs, and then outputs along the session f to signal falsehood. Process Pand_z represents the encoding of boolean conjunction: it inputs the two sessions encoding the booleans that are to be anded, and modulo some bureaucracy due to the replicated processes, sends to the first boolean session the second one followed by the first. It is easy to see this encodes conjunction correctly. Finally, process Pnot_z, is the encoding of boolean negation: it receives the session encoding the boolean that is to be negated and then produces the encoding of a boolean where the t and f sessions mentioned above are flipped with respect to the originally received one, encoding its negation.

We can then produce a process with the following type:

$$\mathsf{TBoolS} \triangleq \exists X.!(\mathsf{true} : X, \mathsf{false} : X, \mathsf{and} : !X \multimap (!X \multimap X), \mathsf{not} : X \multimap X)$$

Intuitively, such a session type describes a persistent abstract service that offers as a labelled choice the boolean values and the conjunction and negation operations. One possible process of this type is the following (using an extended case construct):

$$
\begin{array}{l}
\mathsf{PBoolS} \triangleq x\langle \mathsf{Bool}\rangle.!x(z).z.\mathsf{case}(\mathsf{true} \Rightarrow \mathsf{Ptrue}_z; \mathsf{false} \Rightarrow \mathsf{Pfalse}_z; \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{and} \Rightarrow \mathsf{Pand}_z; \mathsf{not} \Rightarrow \mathsf{Pnot}_z) :: x{:}\mathsf{TBoolS}
\end{array}
$$

A client can then use this service to provide a different service on abstract booleans:

$$
\begin{array}{l}
\mathsf{TBoolOr} \triangleq \exists X.!(\mathsf{true} : X, \mathsf{false} : X, \mathsf{or} : !X \multimap !X \multimap X) \\
\mathsf{PBoolOr} \triangleq x(X).w\langle X\rangle.!w(y).y.\mathsf{case}(\mathsf{true} \Rightarrow \overline{x}\langle v\rangle.v.\mathsf{true}; [v \leftrightarrow y]; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{false} \Rightarrow \overline{x}\langle v\rangle.v.\mathsf{false}; [v \leftrightarrow y]; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{or} \Rightarrow y(b_1).y(b_2).P_{\mathsf{or}}) :: w{:}\mathsf{TBoolOr}
\end{array}
$$

where P_{or} implements boolean disjunction using the DeMorgan duality of conjunction and negation, which are available along channel x. Using cut, we can compose PBoolS and $\mathsf{PBoolOr}$ to produce a closed process providing a session of type $\mathsf{TBoolOr}$.

This illustrates how existential types, allow for representation hiding and provide a natural form of abstract service composition, where a client can use abstract types to implement other services without ever knowing/exposing the internal representation.

AppStore. We now revisit the example of the App Store and produce a process with the appropriate polymorphic type:

$$\mathsf{AppStore} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !X$$

Such a process is (providing the App Store service along channel x):

$$x(X).x(y).!x(w).\overline{y}\langle v\rangle.\overline{v}\langle a\rangle.(P_{api} \mid [w \leftrightarrow v])$$

where P_{api} is a process implementing the App Store API along channel a. The AppStore expects a protocol description X (a session type) and a session y, which is a persistent implementation of X that requires the API provided by the Store. The Store process will then create a replicated service that can provide the behavior X after delivering to y the API implementation that is represented by channel P_{api}.

A generic client for the App Store service (an App developer) using the store's resources to provide some service T (we assume the store is present along channel x):

$$\mathsf{AppDev} \triangleq x\langle T\rangle.\overline{x}\langle y\rangle.(!y(w).w(a).S_w \mid [x \leftrightarrow z])$$

where S_a is a process implementing the service T along channel w, making use of the API provided by the App Store along a. The client begins by sending the protocol T to the App Store, followed by a session y that consists of a persistent service, that when given the API will produce a session of type T. After this communication step, the App Store session now provides the full behavior of T along x, and so the client forwards x along the endpoint channel z, thus providing T along z by making use of the functionality provided by the App Store. We can then produce a typing derivation for the following:

$$x{:}\mathsf{AppStore} \vdash \mathsf{AppDev} :: z{:}T$$

We can make the example more concrete, by providing specific instance of T, a pdf translation service trans. This service takes a file and generates a pdf version of the original file (e.g. performing OCR on images and generating the pdf of the text):

$$\mathsf{trans} \triangleq \mathsf{file} \multimap (\mathsf{pdf} \otimes \mathbf{1})$$

Our system can also capture a richer App Store service, that provides its API but forces the resulting service to support the App Store by adding ads to the service. We can encode such a revised App Store with the following type:

$$\mathsf{AppStoreAds} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !(X \mathbin{\&} \mathsf{AdListings})$$

where the type $\mathsf{AdListings}$ encodes a listing of advertisements that the App Store injects into the service (we encode this injection as a choice – & – to model the ability of a client to choose to view an advertisement). Its straightforward to see how this mechanism can be extended with further functionality (e.g. providing the App Developer with a special administrator service that is then not exposed to the external clients).

4 Strong Normalization of Polymorphic Processes

Here we develop a theory of linear logical relations for polymorphic, session-typed processes. It corresponds to the orthogonal extension of the linear logical relations introduced in [23] to a polymorphic setting. Our main result is a proof of strong normalization (termination) of well-typed processes (Theorem 4.16). As usual, the proof proceeds in two stages: we first define a logical predicate by induction on the linear type structure; then, we show that all well-typed processes are in the logical predicate.

4.1 Auxiliary Definitions

Before introducing the logical predicate, we present some auxiliary definitions. We say that a process P terminates (written $P\Downarrow$) if there is no infinite reduction sequence starting with P. We require an extension to structural congruence that includes the so-called sharpened replication axioms [28].

Definition 4.1 (Extended Structural Congruence). *We define extended structural congruence $\equiv_!$ as the least congruence on processes generated from the rules of Def. 2.3 and 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* $u \notin fn(P)$

Intuitively, axioms (1) and (2) represent the distribution of shared servers among processes, while (3) formalizes the garbage collection of shared servers which can no longer be invoked by any process. It is worth noticing that these axioms express sound behavioral equivalences in our typed setting.

We now state some auxiliary properties regarding extended structural congruence, reduction and labelled transitions which are useful for our development.

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

1. *If $P \to P'$ and $P \equiv_! Q$ then there is Q' such that $Q \to Q'$ 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 $Q \Downarrow$.*

Following Girard's proof technique for System F [17], we first define a notion of *reducibility candidate* at a given type: this is a predicate on well-typed processes which satisfies some crucial closure conditions. As in Girard's proof, the idea is that one of the particular candidates is the "true" logical predicate. Below and henceforth, $\cdot \vdash P :: z{:}A$ stands for a process P well-typed under the empty typing environment.

Definition 4.4 (Reducibility Candidate). *Given a type A and a name z, a* reducibility candidate at $z : A$, *written* $\mathsf{R}[z{:}A]$, *is a predicate on all processes P such that $\cdot \vdash P :: z{:}A$ and satisfy the following:*

1. *If $P \in \mathsf{R}[z{:}A]$ then $P\Downarrow$.*
2. *If $P \in \mathsf{R}[z{:}A]$ and $P \Longrightarrow P'$ then $P' \in \mathsf{R}[z{:}A]$.*
3. *If for all P_i such that $P \Longrightarrow P_i$ we have $P_i \in \mathsf{R}[z{:}A]$ then $P \in \mathsf{R}[z{:}A]$.*

4.2 The Logical Predicate

The logical predicate is parametrized by two mappings, ω and η. Given a type variables context Ω, we write $\omega : \Omega$ to denote that ω is an assignment of closed types to variables in Ω. We write $\eta : \omega$ to denote that η is an assignment of functions taking *names* to reducibility candidates, to type variables in Ω (at the types in ω). This is the key difference from known logical relation arguments for functional languages with impredicative polymorphism, where types are assigned to *terms* and so one maintains a mapping from type variables to reducibility candidates at the appropriate types. In our setting, since types are assigned to channel *names*, we need the ability to refer to reducibility candidates at a given type at channel names which are *yet to be determined*. As an example, consider a session z of type $\forall X.X \multimap X$, which is implemented by a process $z(X).z(x).[x \leftrightarrow z]$. When we expand the definition of the logical predicate at this type, we will need to consider the type variable X both as the type for x and as the type for z, after the two inputs take place, thus needing reducibility candidates at the appropriate type for both names. However, when we define the case for the logical predicate for polymorphic types (in this case, $\forall X.X \multimap X$), we have no *a priori* knowledge as to what will be the names on which we will need candidates—this can only be known in the case for the type variable. Therefore, when we quantify over all types and all reducibility candidates at that type, intuitively we need to delay the choice of the actual name along which the candidate must offer the session type. A "delayed" reducibility candidate at type A is denoted as $\mathsf{R}[-{:}A]$, where $-$ stands for the name which will be instantiated at a later time. In the previous example, it is important to observe that we are dealing with a single reducibility candidate for type X that is instantiated with two different names x and z (which can be thought of "ports" for the reducibility candidate).

For the reader concerned with some foundational issues that may lurk in the background of this slightly non-standard presentation, note that we are still in the same realm (outside of $\mathbf{PA_2}$) as in Girard's original proof. Quoting [17]:

$$
\begin{aligned}
P \in \mathcal{T}_\eta^\omega[z{:}X] &\text{ iff } P \in \eta(X)(z)\\
P \in \mathcal{T}_\eta^\omega[z{:}\mathbf{1}] &\text{ iff } \forall P'.(P \Longrightarrow P' \wedge P' \not\longrightarrow) \Rightarrow P' \equiv_! \mathbf{0}\\
P \in \mathcal{T}_\eta^\omega[z{:}A \multimap B] &\text{ iff } \forall P' y.(P \overset{z(y)}{\Longrightarrow} P') \Rightarrow \forall Q \in \mathcal{T}_\eta^\omega[y{:}A].(\nu y)(P' \mid Q) \in \mathcal{T}_\eta^\omega[z{:}B]\\
P \in \mathcal{T}_\eta^\omega[z{:}A \otimes B] &\text{ iff } \forall P' y.(P \overset{\overline{(\nu y) z\langle y\rangle}}{\Longrightarrow} P') \Rightarrow\\
&\qquad \exists P_1, P_2.(P' \equiv_! P_1 \mid P_2 \wedge P_1 \in \mathcal{T}_\eta^\omega[y{:}A] \wedge P_2 \in \mathcal{T}_\eta^\omega[z{:}B])\\
P \in \mathcal{T}_\eta^\omega[z{:}!A] &\text{ iff } \forall P'.(P \Longrightarrow P') \Rightarrow \exists P_1.(P' \equiv_! !z(y).P_1 \wedge P_1 \in \mathcal{T}_\eta^\omega[y{:}A])\\
P \in \mathcal{T}_\eta^\omega[z{:}A \,\&\, B] &\text{ iff } (\forall P'.(P \overset{z.\mathsf{inl}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_\eta^\omega[z{:}A]) \wedge\\
&\qquad (\forall P'.(P \overset{z.\mathsf{inr}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_\eta^\omega[z{:}B])\\
P \in \mathcal{T}_\eta^\omega[z{:}A \oplus B] &\text{ iff } (\forall P'.(P \overset{\overline{z.\mathsf{inl}}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_\eta^\omega[z{:}A]) \wedge\\
&\qquad (\forall P'.(P \overset{\overline{z.\mathsf{inr}}}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_\eta^\omega[z{:}B])\\
P \in \mathcal{T}_\eta^\omega[z{:}\forall X.A] &\text{ iff } (\forall B, P', \mathsf{R}[-{:}B].\ (B\ \mathsf{type} \wedge P \overset{z\langle B\rangle}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_{\eta[X \mapsto \mathsf{R}[-:B]]}^{\omega[X \mapsto B]}[z{:}A])\\
P \in \mathcal{T}_\eta^\omega[z{:}\exists X.A] &\text{ iff } (\exists B, \mathsf{R}[-{:}B].(B\ \mathsf{type} \wedge P \overset{z\langle B\rangle}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_{\eta[X \mapsto \mathsf{R}[-:B]]}^{\omega[X \mapsto B]}[z{:}A])
\end{aligned}
$$

Fig. 6. Logical Predicate - Base Case

> We seek to use "all possible" axioms of comprehension, or at least a large class of them. [...] to interpret the universal quantification scheme $t\ U$, we have to substitute in the definition of reducibility candidate for t, not an arbitrary candidate, but the one we get by induction on the construction of U. So we must be able to define a set of terms of type U by a *formula*, and this uses the comprehension scheme in an essential way.

In our setting, intuitively, we characterize a set of terms of type U by a slightly different, yet just as reasonable formula, making use of comprehension in a similar way.

We write $\omega[X \mapsto A]$ to denote the mapping ω extended with a new mapping of X to A. We use a similar notation for extensions of η. We write $\hat{\omega}(P)$ (resp. $\hat{\omega}(A)$) to denote the application of the mapping ω to free type-variables in P (resp. in A).

We define a sequent-indexed family of process predicates: a set of processes $\mathcal{T}_\eta^\omega[\Gamma; \Delta \vdash T]$ satisfying some conditions is assigned to any sequent of the form $\Omega; \Gamma; \Delta \vdash T$, provided both $\omega : \Omega$ and $\eta : \omega$. The predicate is defined inductively on the structure of the sequents: the base case considers sequents with an empty left-hand side typing (which we abbreviate to $\mathcal{T}_\eta^\omega[T]$ for readability), whereas the inductive case considers arbitrary typing contexts and exploits principles for typed process composition.

Definition 4.5 (Logical Predicate - Base Case). *For any type A and name z, the logical predicate $\mathcal{T}_\eta^\omega[z{:}A]$ is inductively defined by the set of all processes P such that $\cdot \vdash \hat{\omega}(P) :: z{:}\hat{\omega}(A)$ and satisfy the conditions in Figure 6.*

Definitions 4.5 and 4.6 are the natural extension of the predicate given in [23] to the case of impredicative polymorphic types. Notice how the interpretation of the variable type includes the instantiation at name z of the reducibility candidate given by $\eta(X)$. The clause for the universal $\forall X.A$ denotes that a terminating session of universal type must be able to input *any* type and then be terminating at the open type A, where the meaning of the type variable can be any possible candidate of appropriate type (which includes the actual logical predicate). The clause for the existential is dual.

Definition 4.6 (Logical Predicate - Inductive Case). *For any sequent $\Omega; \Gamma; \Delta \vdash T$ with a non-empty left hand side environment, we define $\mathcal{T}_\eta^\omega[\Gamma; \Delta \vdash T]$ (with $\omega : \Omega$ and $\eta : \omega$) as the set of processes inductively defined as follows:*

$$
\begin{aligned}
P \in \mathcal{T}_\eta^\omega[\Gamma; y{:}A, \Delta \vdash T] \textit{ iff } \forall R \in \mathcal{T}_\eta^\omega[y{:}A].(\nu y)(\hat{\omega}(R) \mid \hat{\omega}(P)) \in \mathcal{T}_\eta^\omega[\Gamma; \Delta \vdash T]\\
P \in \mathcal{T}_\eta^\omega[u{:}A, \Gamma; \Delta \vdash T] \textit{ iff } \forall R \in \mathcal{T}_\eta^\omega[y{:}A].(\nu u)(!u(y).\hat{\omega}(R) \mid \hat{\omega}(P)) \in \mathcal{T}_\eta^\omega[\Gamma; \Delta \vdash T]
\end{aligned}
$$

4.3 Proving Termination

In proofs, it will be useful to have "logical representatives" of the dependencies specified in the left-hand side typing. Recall that notation $\Omega \triangleright \Gamma$ concerns well-formedness conditions wrt type variables (cf. Theorem 2.7).

Definition 4.7. *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, respectively, such that $\Omega \triangleright \Delta$ and $\Omega \triangleright \Gamma$, for some Ω. Letting $I = \{1, \ldots, k\}$ and $J = \{1, \ldots, n\}$, $\omega : \Omega$, $\eta : \omega$, we define the sets of processes $\mathcal{C}_\Gamma^{\omega,\eta}$ and $\mathcal{C}_\Delta^{\omega,\eta}$ as:*

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

Proposition 4.8. *Let Γ and Δ be a non-linear and a linear typing environment, respectively, such that $\Omega \triangleright \Gamma$ and $\Omega \triangleright \Delta$, for some Ω. Assuming $\omega : \Omega$ and $\eta : \omega$, then for all $Q \in \mathcal{C}_\Gamma^{\omega,\eta}$ and for all $R \in \mathcal{C}_\Delta^{\omega,\eta}$, we have $Q \Downarrow$ and $R \Downarrow$. Moreover, $Q \not\longrightarrow$.*

The following lemma formalizes the purpose of "logical representatives": it will allow us to move from predicates for sequents with non empty left-hand side typings to predicates with an empty left-hand side typing, provided processes have been appropriately closed.

Lemma 4.9. *Let P be a process such that $\Omega; \Gamma; \Delta \vdash P :: T$, with $\Gamma = u_1{:}B_1, \ldots, u_k{:}B_k$ and $\Delta = x_1{:}A_1, \ldots, x_n{:}A_n$, with $\Omega \triangleright \Delta$, $\Omega \triangleright \Gamma$, $\omega : \Omega$, and $\eta : \omega$.*
We have: $P \in \mathcal{T}_\eta^\omega[\Gamma; \Delta \vdash T]$ iff $\forall Q \in \mathcal{C}_\Gamma^{\omega,\eta}, \forall R \in \mathcal{C}_\Delta^{\omega,\eta}, (\nu\widetilde{u}, \widetilde{x})(P \mid Q \mid R) \in \mathcal{T}_\eta^\omega[T]$.

Proof. Straightforward from Def. 4.6 and 4.7. □

Theorem 4.10 (The Logical Predicate is a Reducibility Candidate). *If $\Omega \vdash A$ type, $\omega : \Omega$, and $\eta : \omega$ then $\mathcal{T}_\eta^\omega[z{:}A]$ is a reducibility candidate at $z{:}\hat{\omega}(A)$.*

Proof. By induction on the stricture of A; see Appendix B for details. □

Theorem 4.11 (Compositionality). *For any type B, the following holds:*

$$P \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B)]}_{\eta[X\mapsto\mathcal{T}_\eta^\omega[-:B]]}[x{:}A] \;\; \textit{iff} \;\; P \in \mathcal{T}_\eta^\omega[x{:}A\{B/X\}]$$

Proof. By induction on the structure of A.

Case: $A = X$

$P \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B)]}_{\eta[X\mapsto\mathcal{T}_\eta^\omega[-:B]]}[x{:}X]$	assumption
$P \in \eta[X \mapsto \mathcal{T}_\eta^\omega[z{:}B]](X)(x)$	by Def. 4.5
$P \in \mathcal{T}_\eta^\omega[x{:}B]$	by Def. of η

$P \in \mathcal{T}_\eta^\omega[x{:}B]$	assumption
T.S: $P \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B)]}_{\eta[X\mapsto\mathcal{T}_\eta^\omega[-:B]]}[x{:}X]$	
S.T.S: $P \in \mathcal{T}_\eta^\omega[x{:}B]$	follows by assumption

Case: $A = \forall Y.A_1$

$P \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B)]}_{\eta[X\mapsto\mathcal{T}_\eta^\omega[-:B]]}[x{:}\forall X.A_1]$	by assumption
$\forall C, P', \mathsf{R}[-:C].\ (C \text{ type} \wedge P \stackrel{x(C)}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B),Y\mapsto C]}_{\eta[X\mapsto\mathcal{T}_\eta^\omega[-:B],Y\mapsto\mathsf{R}[-:C]]}[x{:}A_1]$	by Def. 4.5
Choose P' arbitrarily, such that $P \stackrel{x(C)}{\Longrightarrow} P'$, for any C type and $\mathsf{R}[-{:}C]$:	
$P' \in \mathcal{T}^{\omega[Y\mapsto C]}_{\eta[Y\mapsto\mathsf{R}[-:C]]}[x{:}A_1\{B/X\}]$	by i.h.

$P \in \mathcal{T}^{\omega}_{\eta}[x{:}\forall X.A_1\{B/X\}]$ by Def. 4.5, satisfying $\Rightarrow$

$P \in \mathcal{T}^{\omega}_{\eta}[x{:}\forall Y.A_1\{B/X\}]$ by assumption
$\forall C, P', \mathsf{R}[-:C].\ (C\ \mathsf{type} \wedge P \overset{x(C)}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}^{\omega[Y\mapsto C]}_{\eta[Y\mapsto \mathsf{R}[-:C]]}[x{:}A_1\{B/X\}]$ by Def. 4.5
Choose P' arbitrarily, such that $P \overset{x(C)}{\Longrightarrow} P'$, for any C type and $\mathsf{R}[-{:}C]$:
$P' \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B),Y\mapsto C]}_{\eta[X\mapsto\mathcal{T}^{\omega}_{\eta}[-:B],Y\mapsto\mathsf{R}[-:C]]}[x{:}A_1]$ by i.h
$P \in \mathcal{T}^{\omega[X\mapsto\hat{\omega}(B)]}_{\eta[X\mapsto\mathcal{T}^{\omega}_{\eta}[-:B]]}[x{:}\forall X.A_1]$ by Def. 4.5, satisfying $\Leftarrow$

All other cases are similar to the above, following by a straightforward induction.

□

Lemma 4.12 (Renaming). *Let A be a well-formed type. If $P \in \mathcal{T}^{\omega}_{\eta}[z{:}A]$ then $P\{x/z\} \in \mathcal{T}^{\omega}_{\eta}[x{:}A]$.*

Proof. Immediate from the definition of the logical predicate. □

Lemma 4.13 (Weakening). *Let P, Q be processes such that $P \in \mathcal{T}^{\omega}_{\eta}[T]$ and $Q \in \mathcal{T}^{\omega}_{\eta}[-:\mathbf{1}]$. Then $(P \mid Q) \in \mathcal{T}^{\omega}_{\eta}[T]$*

Proof. By induction on the structure of T, following the lines of [23]. First, it is worth observing that $P \in \mathcal{T}^{\omega}_{\eta}[T]$ and $Q \in \mathcal{T}^{\omega}_{\eta}[-{:}\mathbf{1}]$ imply $\cdot;\cdot \vdash P :: T$ and $\cdot;\cdot \vdash Q :: -{:}\mathbf{1}$, respectively. Hence, we can derive the typing $\cdot;\cdot \vdash P \mid Q :: T$ (cf. the derived rule (comp)). In fact, the type of Q indicates it cannot offer any visible action to its environment, and so it is "independent" from it.

If $T = -:\mathbf{1}$ then $P \mid Q$ represents the parallel composition of two terminating processes that cannot interact with each other. Hence, for all R such that $P \mid Q \Longrightarrow R$ and $R \not\rightarrow$ we have that $R \equiv_! \mathbf{0}$, and so $P \mid Q \in \mathcal{T}^{\omega}_{\eta}[-{:}\mathbf{1}]$. The cases in which $T \neq -:\mathbf{1}$ rely on the fact that if $P \overset{\alpha}{\Longrightarrow} P'$ then there exists a process R such that $P \mid Q \overset{\alpha}{\Longrightarrow} R$. The proof is by induction on $k = \mathsf{mlen}Q$. If $k = 0$ then $Q \not\equiv_!$ and for every weak transition $P \overset{\alpha}{\Longrightarrow} P'$, we have $P \mid Q \overset{\alpha}{\Longrightarrow} P' \mid Q = R$. In the inductive case, we assume $k > 0$, and so reductions (or the action α) from P may go interleaved with reductions from Q. Given $P \overset{\alpha}{\Longrightarrow} P'$ then by induction hypothesis there is an R' such that $P \mid Q \overset{\alpha}{\Longrightarrow} P' \mid Q' = R'$, with Q reducing to Q' in $k-1$ steps. Then, if $Q' \equiv_! Q''$ we would have $P \mid Q \overset{\alpha}{\Longrightarrow} P' \mid Q' \longrightarrow P' \mid Q''$ which is equivalent to write $P \mid Q \overset{\alpha}{\Longrightarrow} R$, with $R = P' \mid Q''$, and we are done. Finally, we observe that, given $P \overset{\alpha}{\Longrightarrow} P'$, process Q (and its derivatives) pose no difficulties when decomposing P' into smaller processes (in the case $T = z{:}A \otimes B$, for instance). Hence, we can conclude that if $P \in \mathcal{T}^{\omega}_{\eta}[T]$ then $P \mid Q \in \mathcal{T}^{\omega}_{\eta}[T]$, as desired. □

Lemma 4.14 (Strengthening). *If $P \in \mathcal{T}^{\omega[X\mapsto B]}_{\eta[X\mapsto\mathsf{R}[-:B]]}[x{:}A]$ and $X \notin fv(A)$ then $P \in \mathcal{T}^{\omega}_{\eta}[x:A]$*

Proof. Straightforward by induction on A. □

Theorem 4.15 (Fundamental Theorem). *If $\Omega; \Gamma; \Delta \vdash P :: T$ then, for all $\omega : \Omega$ and $\eta : \omega$, we have that $\hat{\omega}(P) \in \mathcal{T}^{\omega}_{\eta}[\Gamma; \Delta \vdash T]$.*

Proof. By induction on typing. The proof is identical to that in [23], with two extra cases for $\forall\mathsf{R}$ and $\forall\mathsf{L}$. We appeal to Lemma 4.9 and show that every $M = (\nu\widetilde{u}, \widetilde{x})(P \mid G \mid D)$ with $G \in \mathcal{C}^{\omega,\eta}_{\Gamma}$ and $D \in \mathcal{C}^{\omega,\eta}_{\Delta}$ is in $\mathcal{T}^{\omega}_{\eta}[T]$. We show only the interesting new cases.

Case (T$\forall$R): $\Omega; \Gamma; \Delta \vdash z(X).P :: z{:}\forall X.A$

$\Omega, X; \Gamma; \Delta \vdash P :: z{:}A$ by inversion
$\hat{\omega}(P)\{B/X\} \in \mathcal{T}^{\omega'}_{\eta'}[\Gamma; \Delta \vdash z{:}A]$ by i.h., for some B, $\omega' = \omega[X \mapsto B]$ and $\eta' = \eta[X \mapsto R[-:B]]$
Pick any $G \in \mathcal{C}^{\omega',\eta'}_{\Gamma}$, $D \in \mathcal{C}^{\omega',\eta'}_{\Delta}$:
$G \Downarrow, G \not\longrightarrow, D \Downarrow$ by Prop. 4.8
$(\nu\widetilde{u}, \widetilde{x})(\hat{\omega}(P)\{B/X\} \mid G \mid D) \in \mathcal{T}^{\omega'}_{\eta'}[z{:}A]$ (a) by Lemma 4.9
S.T.S: $M = (\nu\widetilde{u}, \widetilde{x})(\hat{\omega}(z(X).P) \mid G \mid D) \in \mathcal{T}^{\omega}_{\eta}[z{:}\forall X.A]$

$M \overset{z(B)}{\Longrightarrow} (\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(P)\{B/X\} \mid G \mid D') = M_1$
$M_1 \in \mathcal{T}_{\eta'}^{\omega'}[z{:}A]$ by (a) and forward closure
$M \in \mathcal{T}_{\eta'}^{\omega'}[z{:}\forall X.A]$ by Def. 4.5
$M \in \mathcal{T}_{\eta}^{\omega}[z{:}\forall X.A]$ by Lemma 4.14

Case (T$\forall$L): $\Omega; \Gamma; \Delta, x : \forall X.A \vdash x\langle B\rangle.P :: T$

$\Omega \vdash B$ type by inversion
$\Omega; \Gamma; \Delta, x : A\{B/X\} \vdash P :: T$ by inversion
$\hat{\omega}(P) \in \mathcal{T}_{\eta}^{\omega}[\Gamma; \Delta, x : A\{B/X\} \vdash T]$ by i.h.
Pick any $G \in \mathcal{C}_{\Gamma}^{\omega,\eta}$, $D \in \mathcal{C}_{\Delta}^{\omega,\eta}$:
$G \Downarrow, G \not\longrightarrow, D \Downarrow$ by Prop. 4.8
$(\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(P) \mid G \mid D \mid \mathcal{T}_{\eta}^{\omega}[x : A\{B/X\}]) \in \mathcal{T}_{\eta}^{\omega}[T]$ (a) by Lemma 4.9
S.T.S: $(\nu \widetilde{u}, \widetilde{x}, x)(\hat{\omega}(x\langle B\rangle.P) \mid G \mid D \mid \mathcal{T}_{\eta}^{\omega}[x : \forall X.A]) \in \mathcal{T}_{\eta}^{\omega}[T]$
Pick $R \in \mathcal{T}_{\eta}^{\omega}[x{:}\forall X.A]$
S.T.S: $M = (\nu \widetilde{u}, \widetilde{x}, x)(x\langle\hat{\omega}(B)\rangle.\hat{\omega}(P)) \mid G \mid D \mid R) \in \mathcal{T}_{\eta}^{\omega}[T]$
$R \overset{x(\hat{\omega}(B))}{\Longrightarrow} R'$ by Theorem 4.10 and XYZ
$\forall \mathsf{R}[- : \hat{\omega}(B)].R' \in \mathcal{T}_{\eta[X \mapsto \mathsf{R}[-:\hat{\omega}(B)]]}^{\omega[X \mapsto \hat{\omega}(B)]}[x{:}A]$ by Def. 4.5
$R' \in \mathcal{T}_{\eta[X \mapsto \mathcal{T}_{\eta}^{\omega}[-:B]]}^{\omega[X \mapsto \hat{\omega}(B)]}[x{:}A]$ by Theorem 4.10
$R' \in \mathcal{T}_{\eta}^{\omega}[x{:}A\{B/X\}]$ by Theorem 4.11
$M \Longrightarrow (\nu \widetilde{u}, \widetilde{x}, x)(\hat{\omega}(P)) \mid G \mid D' \mid R') = M_1$
$M_1 \in \mathcal{T}_{\eta}^{\omega}[T]$ by (a) and forward closure
$M \in \mathcal{T}_{\eta}^{\omega}[T]$ by backward closure

Case (T$\exists$R): $\Omega; \Gamma; \Delta \vdash z\langle B\rangle.P :: z{:}\exists X.A$

$\Omega; \Gamma; \Delta \vdash P :: z{:}A\{B/X\}$ by inversion
$\Omega \vdash B$ type by inversion
$\hat{\omega}(P) \in \mathcal{T}_{\eta}^{\omega}[\Gamma; \Delta \vdash z{:}A\{B/X\}]$ by i.h.
Pick any $G \in \mathcal{C}_{\Gamma}^{\omega,\eta}$, $D \in \mathcal{C}_{\Delta}^{\omega,\eta}$:
$G \Downarrow, G \not\longrightarrow, D \Downarrow$ by Prop. 4.8
$M_1 = (\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(P) \mid G \mid D) \in \mathcal{T}_{\eta}^{\omega}[z{:}A\{B/X\}]$ (a) by Lemma 4.9
S.T.S: $(\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(z\langle B\rangle.P) \mid G \mid D) \in \mathcal{T}_{\eta}^{\omega}[z{:}\exists X.A]$
S.T.S: $M = (\nu \widetilde{u}, \widetilde{x})(z\langle\hat{\omega}(B)\rangle.\hat{\omega}(P) \mid G \mid D) \in \mathcal{T}_{\eta}^{\omega}[z{:}\exists X.A]$
S.T.S: $\exists C.\mathsf{R}[-{:}C].(C \text{ type} \wedge M \overset{z\langle C\rangle}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_{\eta[X \mapsto \mathcal{R}[-:C]]}^{\omega[X \mapsto C]}[z{:}A]$
S.T.S: $\exists C.\mathsf{R}[- : C].(C \text{ type} \wedge M \overset{z\langle C\rangle}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_{\eta}^{\omega}[z{:}A\{C/X\}]$ (b) by Theorem 4.11
Pick $C = \hat{\omega}(B)$ and $\mathsf{R}[- : C]$ as $\mathcal{T}_{\eta}^{\omega}[- : \hat{\omega}(B)]$
$M \overset{z(\hat{\omega}(B))}{\Longrightarrow} (\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(P) \mid G \mid D')$
$M_1 \Longrightarrow (\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(P) \mid G \mid D')$
$(\nu \widetilde{u}, \widetilde{x})(\hat{\omega}(P) \mid G \mid D') \in \mathcal{T}_{\eta}^{\omega}[z{:}A\{B/X\}]$ by forward closure, satisfying (b)

Case (T$\exists$L): $\Omega; \Gamma; \Delta, x : \exists X.A \vdash x(X).P :: T$

$\Omega, X; \Gamma; \Delta, x{:}A \vdash P :: T$ by inversion
$\hat{\omega}(P)\{B/X\} \in \mathcal{T}_{\eta'}^{\omega'}[\Gamma; \Delta, x{:}A \vdash T]$ by i.h., for some B, $\omega' = \omega[X \mapsto B]$ and $\eta' = \eta[X \mapsto R[-{:}B]]$
Pick any $G \in \mathcal{C}_{\Gamma}^{\omega,\eta}$, $D \in \mathcal{C}_{\Delta}^{\omega,\eta}$:
$G \Downarrow, G \not\longrightarrow, D \Downarrow$ by Prop. 4.8
$(\nu \widetilde{u}, \widetilde{x}, x)(\hat{\omega}(P)\{B/X\} \mid G \mid D \mid \mathcal{T}_{\eta'}^{\omega'}[x{:}A]) \in \mathcal{T}_{\eta'}^{\omega'}[T]$ (a) by Lemma 4.9
Pick $R' \in \mathcal{T}_{\eta'}^{\omega'}[x{:}A]$:

$(\nu\widetilde{u},\widetilde{x},x)(\hat{\omega}(P)\{B/X\} \mid G \mid D \mid R') \in \mathcal{T}^{\omega'}_{\eta'}[T]$ (b) by (a)
S.T.S: $(\nu\widetilde{u},\widetilde{x},x)(x(X).\hat{\omega}(P) \mid G \mid D \mid \mathcal{T}^{\omega}_{\eta}[x{:}\exists X.A]) \in \mathcal{T}^{\omega}_{\eta}[T]$
Pick $R = x\langle B\rangle.R'$
$R \stackrel{x\langle B\rangle}{\Longrightarrow} R'$ and $R' \in \mathcal{T}^{\omega'}_{\eta'}[x{:}A]$
$R \in \mathcal{T}^{\omega}_{\eta}[x{:}\exists X.A]$ by Def. 4.5
S.T.S: $M = (\nu\widetilde{u},\widetilde{x},x)(x(X).\hat{\omega}(P) \mid G \mid D \mid R) \in \mathcal{T}^{\omega}_{\eta}[T]$
$M \Longrightarrow (\nu\widetilde{u},\widetilde{x},x)(\hat{\omega}(P)\{B/X\} \mid G \mid D \mid R') \in \mathcal{T}^{\omega'}_{\eta'}[T]$ (c) by (b)
$M \in \mathcal{T}^{\omega'}_{\eta'}[T]$ by backward closure and (c)
$M \in \mathcal{T}^{\omega}_{\eta}[T]$ by Lemma 4.14

Case (Tid): $\Omega; \Gamma; x{:}A \vdash [x \leftrightarrow z] :: z{:}A$

Pick any $G \in \mathcal{C}^{\omega,\eta}_{\Gamma}$:
$G \Downarrow, G \not\longrightarrow$ by Prop. 4.8
$D \in \mathcal{T}^{\omega}_{\eta}[x{:}A]$ (a)
S.T.S: $M = (\nu\widetilde{u},x)([x \leftrightarrow z] \mid G \mid D) \in \mathcal{T}^{\omega}_{\eta}[z{:}A]$
$M \longrightarrow (\nu\widetilde{u})(G\{z/x\} \mid D\{z/x\}) \equiv_! D\{z/x\} = M'$ (b) since $x \notin fn(G)$
$M' \in \mathcal{T}^{\omega}_{\eta}[z{:}A]$ (c) by (a) and Lemma 4.12
$M \in \mathcal{T}^{\omega}_{\eta}[z{:}A]$ (d) by (b), (c), and backward closure
$[x \leftrightarrow z] \in \mathcal{T}^{\omega}_{\eta}[\Gamma; x : A \vdash z : A]$ by (d) and Lemma 4.9

□

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

Theorem 4.16 (Termination). *If $\Omega; \Gamma; \Delta \vdash P :: T$ then $\hat{\omega}(P)\Downarrow$, for every $\omega : \Omega$.*

Proof. Follows from previously proven facts:

$\Omega; \Gamma; \Delta \vdash P :: T$	[Assumption]	(a)
$\hat{\omega}(P) \in \mathcal{T}^{\omega}_{\eta}[\Gamma; \Delta \vdash T]$, for all $\omega : \Omega, \eta : \omega$	[By Theorem 4.15 and (a)]	(b)
Pick any $G \in \mathcal{C}^{\omega,\eta}_{\Gamma}$, $D \in \mathcal{C}^{\omega,\eta}_{\Delta}$:		
$G\Downarrow, D\Downarrow$	[By Prop 4.8]	(c)
$(\nu\widetilde{u},\widetilde{x})(\hat{\omega}(P) \mid G \mid D) \in \mathcal{T}^{\omega}_{\eta}[T]$	[By Lemma 4.9 on (b)]	(d)
$(\nu\widetilde{u},\widetilde{x})(\hat{\omega}(P) \mid G \mid D)\Downarrow$	[From (d) and Def 4.5]	(e)
$\hat{\omega}(P)\Downarrow$	[Consequence of (c) and (e)]	

□

5 Relational Parametricity for Session-Typed Processes

We present a behavioral theory for polymorphic session-typed processes. First, we define a typed variant of *barbed congruence*, $\cong$, the usual notion of contextual equivalence in concurrency. We then introduce *logical equivalence*, $\approx_{\mathrm{L}}$, the natural extension of $\mathcal{T}^{\omega}_{\eta}[\Gamma; \Delta \vdash T]$ to the binary setting. Using $\approx_{\mathrm{L}}$ we prove a *parametricity* result for well-typed processes (Theorem 5.16), along the lines of Reynold's abstraction theorem [27], which allows us to ultimately show that $\approx_{\mathrm{L}}$ and $\cong$ *coincide* (Theorems 5.19 and 5.20). Below, we follow closely the presentation by Harper [18] for System F.

5.1 Barbed Congruence

Barbed congruence is defined as the largest equivalence relation on processes that is (i) closed under internal actions; (ii) preserves *barbs*— arguably the most basic observable on the behavior of processes; and is (iii) *contextual*, i.e., preserved by every admissible process context. We will now make these three *desiderata* formally precise, extending barbed congruence to our typed-setting through a notion of type-respecting relations. Below, we use $\mathcal{S}$ to range over sequents of the form $\Gamma; \Delta \vdash T$.

Definition 5.1 (Type-respecting relations). *A (binary)* type-respecting 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. Also, we write $\Omega; \Gamma; \Delta \vdash P \,\mathcal{R}\, Q :: T$ to mean that*

(i) $\Omega; \Gamma; \Delta \vdash P :: T$ *and* $\Omega; \Gamma; \Delta \vdash Q :: T$,
(ii) $(P, Q) \in \mathcal{R}_{\Omega;\Gamma;\Delta\vdash T}$.

We also need to define what constitutes a *type-respecting* equivalence relation. In what follows, we will always assume type-respecting relations and ommit the adjective "type-respecting".

Definition 5.2. *A relation $\mathcal{R}$ is said to be*

- Reflexive*, if $\Omega; \Gamma; \Delta \vdash P :: T$ implies $\Omega; \Gamma; \Delta \vdash P \,\mathcal{R}\, P :: T$;*
- Symmetric*, if $\Omega; \Gamma; \Delta \vdash P \,\mathcal{R}\, Q :: T$ implies $\Omega; \Gamma; \Delta \vdash Q \,\mathcal{R}\, P :: T$;*
- Transitive*, $\Omega; \Gamma; \Delta \vdash P \,\mathcal{R}\, P' :: T$ and $\Omega; \Gamma; \Delta \vdash P' \,\mathcal{R}\, Q :: T$ imply $\Omega; \Gamma; \Delta \vdash P \,\mathcal{R}\, Q :: T$.*

Moreover, $\mathcal{R}$ is said to be an equivalence *if it is reflexive, symmetric, and transitive.*

We can now define the three *desiderata* for barbed congruence: τ-closed, barb preserving and contextuality.

Definition 5.3 (τ-closed). *A relation $\mathcal{R}$ is* τ-closed *if $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: T$ and $P \longrightarrow P'$ imply there exists a Q' such that $Q \Longrightarrow Q'$ and $\Omega; \Gamma; \Delta \vdash P' \mathcal{R} Q' :: T$.*

The following definition of observability predicates, or *barbs*, extends the classical presentation with the observables for type input and output:

Definition 5.4 (Barbs). *Given a name x, let $O_x = \{\overline{x}, x, \overline{x.\mathsf{inl}}, \overline{x.\mathsf{inr}}, x.\mathsf{inl}, x.\mathsf{inr}\}$ be the set of* basic observables *under x. Given a well-typed process P, we write:*

- $\mathsf{barb}(P, \overline{x})$*, if* $P \xrightarrow{\overline{(\nu y)x\langle y\rangle}} P'$*;*
- $\mathsf{barb}(P, \overline{x})$*, if* $P \xrightarrow{\overline{x\langle A\rangle}} P'$*, for some A, P';*
- $\mathsf{barb}(P, x)$*, if* $P \xrightarrow{x(A)} P'$*, for some A, P';*
- $\mathsf{barb}(P, x)$*, if* $P \xrightarrow{x(y)} P'$*, for some y, P';*
- $\mathsf{barb}(P, \alpha)$*, if* $P \xrightarrow{\alpha} P'$*, for some P' and $\alpha \in O_x \setminus \{x, \overline{x}\}$.*

Given some $o \in O_x$, we write $\mathsf{wbarb}(P, o)$ *if there exists a P' such that $P \Longrightarrow P'$ and* $\mathsf{barb}(P', o)$ *holds.*

Definition 5.5 (Barb preserving relation). *A relation $\mathcal{R}$ is a* barb preserving *if, for every name x, $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: T$ and* $\mathsf{barb}(P, o)$ *imply* $\mathsf{wbarb}(Q, o)$*, for any $o \in O_x$.*

In preparation to the definition of *contextuality*, we introduce a natural notion of *(typed) process contexts*. Intuitively, a context is a process that contains one hole, noted $\bullet$. Holes are *typed*: a hole can only be filled in with a process matching its type. We shall use $K, K', \ldots$ for ranging over properly defined contexts, in the sense given next. We rely on left- and right-hand side typings for defining contexts and their properties precisely. We consider contexts with exactly one hole, but our definitions are easy to generalize.

We define a minimal extension of the syntax of processes with $\bullet$. We also extend sequents, in the following way:

$$\mathcal{H}; \Omega; \Gamma; \Delta \vdash K :: S$$

Intuitively, $\mathcal{H}$ contains a description of a hole occurring in (context) K: we have that $\bullet_{\Omega;\Gamma;\Delta\vdash T};\Omega;\Gamma;\Delta'\vdash K :: S$ is the type of a context K whose hole is to be substituted by some process P such that $\Gamma;\Delta\vdash P :: T$. As a result of the substitution, we obtain process $\Omega;\Gamma;\Delta'\vdash K[P] :: S$. Since we consider at most one hole, $\mathcal{H}$ is either empty or has exactly one element. If $\mathcal{H}$ is empty then K is a process and we obtain the usual typing rules. We write $\Omega;\Gamma;\Delta\vdash R :: T$ rather than $\cdot;\Omega;\Gamma;\Delta\vdash R :: T$. The definition of typed contexts is completed by extending the type system with the following two rules:

$$\frac{}{\bullet_{\Omega;\Gamma;\Delta\vdash T};\Omega;\Gamma;\Delta\vdash\bullet :: T}\ T\mathsf{hole} \qquad \frac{\Omega;\Gamma;\Delta\vdash R :: T \quad \bullet_{\Omega;\Gamma;\Delta\vdash T};\Omega;\Gamma;\Delta'\vdash K :: S}{\Omega;\Gamma;\Delta'\vdash K[R] :: S}\ T\mathsf{fill}$$

Axiom (Thole) allows to introduce holes into typed contexts. In rule (Tfill), R is a process (it does not have any holes), and K is a context with a hole of type $\Omega;\Gamma;\Delta\vdash T$. The substitution of occurrences of $\bullet$ in K with R, noted $K[R]$ is sound as long as the typings of R coincide with those declared in $\mathcal{H}$ for K.

Definition 5.6 (Contextuality). *A relation $\mathcal{R}$ is* contextual *if it satisfies the conditions in Figure 7 (Page 17).*

Having made precise the necessary conditions, we can now define barbed congruence:

Definition 5.7 (Barbed Congruence). Barbed congruence*, noted $\cong$, is the largest equivalence on well-typed processes that is τ-closed, barb preserving, and contextual.*

5.2 Logical Equivalence

We now move on to define a notion of *binary* logical equivalence relation for well-typed processes: it arises as a natural extension of the logical predicate of Section 4 to the relational setting. It can be also seen as the natural generalization of the typed bisimilarity introduced in [23]. Using this logical equivalence, we show that it is possible to precisely characterize barbed congruence ($\cong$, cf. Definition 5.7), therefore setting logical equivalence as a sound and complete proof technique for barbed congruence. We begin by defining the crucial notion of *equivalence candidate*. Equivalence candidates consist of equivalence relations between well-typed processes satisfying certain basic closure conditions.

Definition 5.8 (Equivalence Candidate). *Let A, B be types. An* equivalence candidate *$\mathcal{R}$ at $z{:}A$ and $z{:}B$, noted $\mathcal{R} :: z{:}A \leftrightarrow B$, is a binary relation on processes such that, for every $(P,Q)\in\mathcal{R} :: z{:}A\leftrightarrow B$ both $\cdot\vdash P :: z{:}A$ and $\cdot\vdash Q :: z{:}B$ hold, together with the following conditions:*

1. *If $(P,Q)\in\mathcal{R} :: z{:}A\leftrightarrow B$, $\cdot\vdash P\cong P' :: z{:}A$, and $\cdot\vdash Q\cong Q' :: z{:}B$ then $(P',Q')\in\mathcal{R} :: z{:}A\leftrightarrow B$.*
2. *If $(P,Q)\in\mathcal{R} :: z{:}A\leftrightarrow B$ then, for all P_0 such that $P_0\Longrightarrow P$, we have $(P_0,Q)\in\mathcal{R} :: z{:}A\leftrightarrow B$. Similarly for Q: If $(P,Q)\in\mathcal{R} :: z{:}A\leftrightarrow B$ then, for all Q_0 such that $Q_0\Longrightarrow Q$ then $(P,Q_0)\in\mathcal{R} :: z{:}A\leftrightarrow B$.*

We often write $(P,Q)\in\mathcal{R} :: z{:}A\leftrightarrow B$ as $P\,\mathcal{R}\,Q :: z{:}A\leftrightarrow B$.

Definition 5.8 follows closely the presentation by Harper [18]. Observe the use of barbed congruence in item (1), representing the fact that equivalence candidates are closed wrt $\cong$. Item (2) can be shown to be redundant. As in our definition of logical predicate, we require some auxiliary notation. We recall that $\omega : \Omega$ denotes a type substitution ω that assigns a closed type to type variables in Ω. Given two type substitutions $\omega : \Omega$ and $\omega' : \Omega$, we define an equivalence candidate assignment η between ω and ω' as a mapping of a delayed (in the sense of the mapping η of Section 4) equivalence candidate $\eta(X) :: -{:}\omega(X)\leftrightarrow\omega'(X)$ to the type variables in Ω. We write $\eta(X)(z)$ for the instantiation of the delayed equivalence candidate with the name z. We write $\eta : \omega\leftrightarrow\omega'$ to denote that η is a (delayed) equivalence candidate assignment between ω and ω'.

We define a sequent-indexed family of process relations, that is, a set of pairs of processes (P,Q), written $\Gamma;\Delta\vdash P\approx_{\mathrm{L}} Q :: T[\eta : \omega\leftrightarrow\omega']$, satisfying some conditions, is assigned to any sequent of the form $\Omega;\Gamma;\Delta\vdash T$, with $\omega : \Omega$, $\omega' : \Omega$ and $\eta : \omega\leftrightarrow\omega'$. The relation is defined inductively on the structure of the sequents, similar to the definition of the logical predicate for termination, with a base case which considers empty left-hand side typings (abbreviated to $P\approx_{\mathrm{L}} Q :: T[\eta : \omega\leftrightarrow\omega']$), and an inductive case which considers arbitrary typing contexts.

A relation $\mathcal{R}$ is contextual if

0. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: y{:}A$ and $\Omega; \Gamma; y{:}A \vdash [y \leftrightarrow z] :: z{:}A$ imply
   $\Omega; \Gamma; \Delta \vdash (\nu y)(P \mid [y \leftrightarrow z]) \mathcal{R} (\nu y)(Q \mid [y \leftrightarrow z]) :: z{:}A$
1. $\Omega; \Gamma; \Delta, y{:}A \vdash P \mathcal{R} Q :: x{:}B$ implies $\Omega; \Gamma; \Delta \vdash x(y).P \mathcal{R} x(y).Q :: x{:}A \multimap B$
2. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: y{:}A$ and $\Omega; \Gamma; \Delta' \vdash S \mathcal{R} T :: x{:}B$ imply
   $\Omega; \Gamma; \Delta, \Delta' \vdash (\nu y)x\langle y\rangle.(P \mid S) \mathcal{R} (\nu y)x\langle y\rangle.(Q \mid T) :: x{:}A \otimes B$
3. $\Omega; \Gamma; \Delta' \vdash P \mathcal{R} Q :: x{:}B$ and $\Omega; \Gamma; \Delta' \vdash S \mathcal{R} T :: y{:}B$ imply
   $\Omega; \Gamma; \Delta, \Delta' \vdash (\nu y)x\langle y\rangle.(S \mid P) \mathcal{R} (\nu y)x\langle y\rangle.(T \mid Q) :: x{:}A \otimes B$
4. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: x{:}A$ and $\Omega; \Gamma; \Delta' \vdash S \mathcal{R} T :: x{:}B$ imply
   $\Omega; \Gamma; \Delta \vdash x.\mathtt{case}(P, S) \mathcal{R} x.\mathtt{case}(Q, T) :: x{:}A \,\&\, B$
5. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: x{:}B$ and $\Omega; \Gamma; \Delta' \vdash S \mathcal{R} T :: x{:}A$ imply
   $\Omega; \Gamma; \Delta \vdash x.\mathtt{case}(S, P) \mathcal{R} x.\mathtt{case}(T, Q) :: x{:}A \,\&\, B$
6. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: x{:}A$ implies $\Omega; \Gamma; \Delta \vdash x.\mathsf{inl}; P \mathcal{R} x.\mathsf{inl}; Q :: x{:}A \oplus B$
7. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: x{:}B$ implies $\Omega; \Gamma; \Delta \vdash x.\mathsf{inr}; P \mathcal{R} x.\mathsf{inr}; Q :: x{:}A \oplus B$
8. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: x{:}A$ and $\Omega; \Gamma; \Delta', x{:}A \vdash S \mathcal{R} H :: T$ imply
   $\Omega; \Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid S) \mathcal{R} (\nu x)(Q \mid H) :: T$
9. $\Omega; \Gamma; \Delta, x{:}A \vdash P \mathcal{R} Q :: T$ and $\Omega; \Gamma; \Delta' \vdash S \mathcal{R} H :: x{:}A$ imply
   $\Omega; \Gamma; \Delta, \Delta' \vdash (\nu x)(S \mid P) \mathcal{R} (\nu x)(H \mid Q) :: T$
10. $\Omega; \Gamma; \cdot \vdash P \mathcal{R} Q :: y{:}A$ and $\Omega; \Gamma, u{:}A; \Delta \vdash S \mathcal{R} H :: T$ imply
    $\Omega; \Gamma; \Delta \vdash (\nu u)(!u(y).P \mid S) \mathcal{R} (\nu u)(!u(y).Q \mid H) :: T$
11. $\Omega; \Gamma, u{:}A; \Delta \vdash P \mathcal{R} Q :: T$ and $\Omega; \Gamma; \cdot \vdash S \mathcal{R} H :: y{:}A$ and imply
    $\Omega; \Gamma; \Delta \vdash (\nu u)(!u(y).S \mid P) \mathcal{R} (\nu u)(!u(y).H \mid Q) :: T$
12. $\Omega; \Gamma, u{:}A; \Delta \vdash P\{u/x\} \mathcal{R} Q\{u/x\} :: T$ implies $\Omega; \Gamma; \Delta, x{:}!A \vdash P \mathcal{R} Q :: T$
13. $\Omega; \Gamma; \cdot \vdash P \mathcal{R} Q :: y{:}A$ implies $\Omega; \Gamma; \cdot \vdash !x(y).P \mathcal{R} !x(y).Q :: x{:}!A$
14. $\Omega; \Gamma; \Delta, y{:}A, x{:}B \vdash P \mathcal{R} Q :: T$ implies $\Omega; \Gamma; \Delta, x{:}A \otimes B \vdash x(y).P \mathcal{R} x(y).Q :: T$
15. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: y{:}A$ and $\Omega; \Gamma; \Delta', x{:}B \vdash S \mathcal{R} H :: T$ imply
    $\Omega; \Gamma; \Delta, \Delta', x{:}A \multimap B \vdash (\nu y)x\langle y\rangle.(P \mid S) \mathcal{R} (\nu y)x\langle y\rangle.(Q \mid H) :: T$
16. $\Omega; \Gamma; \Delta, x{:}B \vdash P \mathcal{R} Q :: T$ and $\Omega; \Gamma; \Delta' \vdash S \mathcal{R} H :: y{:}A$ imply
    $\Omega; \Gamma; \Delta, \Delta', x{:}A \multimap B \vdash (\nu y)x\langle y\rangle.(P \mid S) \mathcal{R} (\nu y)x\langle y\rangle.(Q \mid H) :: T$
17. $\Omega; \Gamma, u{:}A; \Delta, y{:}A \vdash P \mathcal{R} Q :: T$ implies $\Omega; \Gamma, u{:}A; \Delta \vdash (\nu y)u\langle y\rangle.P \mathcal{R} (\nu y)u\langle y\rangle.Q :: T$
18. $\Omega; \Gamma; \Delta, x{:}A \vdash P \mathcal{R} Q :: T$ and $\Omega; \Gamma; \Delta, x{:}B \vdash S \mathcal{R} H :: T$ imply
    $\Omega; \Gamma; \Delta, x{:}A \oplus B \vdash x.\mathtt{case}(P, S) \mathcal{R} x.\mathtt{case}(Q, H) :: T$
19. $\Omega; \Gamma; \Delta, x{:}B \vdash P \mathcal{R} Q :: T$ and $\Omega; \Gamma; \Delta, x{:}A \vdash S \mathcal{R} H :: T$ imply
    $\Omega; \Gamma; \Delta, x{:}A \oplus B \vdash x.\mathtt{case}(S, P) \mathcal{R} x.\mathtt{case}(H, Q) :: T$
20. $\Omega; \Gamma; \Delta, x{:}A \vdash P \mathcal{R} Q :: T$ implies $\Omega; \Gamma; \Delta, x{:}A \,\&\, B \vdash x.\mathsf{inl}; P \mathcal{R} x.\mathsf{inl}; Q :: T$
21. $\Omega; \Gamma; \Delta, x{:}B \vdash P \mathcal{R} Q :: T$ implies $\Omega; \Gamma; \Delta, x{:}A \,\&\, B \vdash x.\mathsf{inr}; P \mathcal{R} x.\mathsf{inr}; Q :: T$
22. $\Omega, X; \Gamma; \Delta \vdash P \mathcal{R} Q :: z{:}A$ implies $\Omega; \Gamma; \Delta \vdash z(X).P \mathcal{R} z(X).Q :: z{:}\forall X.A$
23. $\Omega; \Gamma; \Delta \vdash P \mathcal{R} Q :: z{:}A\{B/X\}$ and $\Omega \vdash B$ type imply $\Omega; \Gamma; \Delta \vdash z\langle B\rangle.P \mathcal{R} z\langle B\rangle.Q :: z{:}\exists X.A$
24. $\Omega; \Gamma; \Delta, x : A\{B/X\} \vdash P \mathcal{R} Q :: T$ and $\Omega \vdash B$ type imply
    $\Omega; \Gamma; \Delta, x : \forall X.A \vdash x\langle B\rangle.P \mathcal{R} x\langle B\rangle.Q :: T$
25. $\Omega, X; \Gamma; \Delta, x : A \vdash P \mathcal{R} Q :: T$ implies $\Omega; \Gamma; \Delta, x : \exists X.A \vdash x(X).P \mathcal{R} x(X).Q :: T$

Fig. 7. Conditions for contextual type-respecting relations (see Definition 5.6)

Definition 5.9 (Logical Equivalence - Base Case). *Given any type A and mappings ω, ω', η, we define* logical equivalence

$$P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$$

as the largest binary relation that contains all pairs of processes (P, Q) such that (i) $\cdot \vdash \hat{\omega}(P) :: z{:}\hat{\omega}(A)$; (ii) $\cdot \vdash \hat{\omega}'(Q) :: z{:}\hat{\omega}'(A)$; and (iii) satisfies the conditions in Figure 8 (Page 18).

Definition 5.10 (Logical Equivalence - Inductive Case). *Let Γ, Δ be non empty typing environments. Given the sequent $\Omega; \Gamma; \Delta \vdash T$, the binary relation on processes $\Gamma; \Delta \vdash P \approx_{\mathrm{L}} Q :: T[\eta : \omega \leftrightarrow \omega']$ (with $\omega, \omega' : \Omega$ and*

$$
\begin{aligned}
P \approx_{\mathrm{L}} Q :: z{:}X[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; (P, Q) \in \eta(X)(z) \\
P \approx_{\mathrm{L}} Q :: z{:}\mathbf{1}[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; \forall P', Q'.(P \Longrightarrow P' \wedge P' \not\longrightarrow \wedge\, Q \Longrightarrow Q' \wedge Q' \not\longrightarrow) \Rightarrow \\
&\qquad (P' \equiv_! \mathbf{0} \wedge Q' \equiv_! \mathbf{0}) \\
P \approx_{\mathrm{L}} Q :: z{:}A \multimap B[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; \forall P', y.\,(P \xrightarrow{z(y)} P') \Rightarrow \exists Q'.Q \overset{z(y)}{\Longrightarrow} Q' \; s.t. \\
&\qquad \forall R_1, R_2.\, R_1 \approx_{\mathrm{L}} R_2 :: y{:}A[\eta : \omega \leftrightarrow \omega'] \\
&\qquad (\nu y)(P' \mid R_1) \approx_{\mathrm{L}} (\nu y)(Q' \mid R_2) :: z{:}B[\eta : \omega \leftrightarrow \omega'] \\
P \approx_{\mathrm{L}} Q :: z{:}A \otimes B[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; \forall P', y.(P \xrightarrow{\overline{(\nu y) z\langle y \rangle}} P') \Rightarrow \exists Q'.Q \overset{\overline{(\nu y) z\langle y \rangle}}{\Longrightarrow} Q' \; s.t. \\
&\qquad \forall R_1, R_2, n.\, y{:}A \vdash R_1 \approx_{\mathrm{L}} R_2 :: n{:}\mathbf{1}[\eta : \omega \leftrightarrow \omega'] \\
&\qquad (\nu y)(P' \mid R_1) \approx_{\mathrm{L}} (\nu y)(Q' \mid R_2) :: z{:}B[\eta : \omega \leftrightarrow \omega'] \\
P \approx_{\mathrm{L}} Q :: z{:}!A[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; \forall P'.(P \xrightarrow{z(y)} P') \Rightarrow \exists Q'.Q \overset{z(y)}{\Longrightarrow} Q' \wedge \\
&\qquad \forall R_1, R_2, n.\, y{:}A \vdash R_1 \approx_{\mathrm{L}} R_2 :: n{:}\mathbf{1}[\eta : \omega \leftrightarrow \omega'] \\
&\qquad (\nu y)(P' \mid R_1) \approx_{\mathrm{L}} (\nu y)(Q' \mid R_2) :: z{:}!A[\eta : \omega \leftrightarrow \omega'] \\
P \approx_{\mathrm{L}} Q :: z{:}A \,\&\, B[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff } \\
&\qquad (\forall P'.(P \xrightarrow{z.\mathsf{inl}} P') \Rightarrow \exists Q'.(Q \overset{z.\mathsf{inl}}{\Longrightarrow} Q' \wedge P' \approx_{\mathrm{L}} Q' :: z{:}A[\eta : \omega \leftrightarrow \omega'])) \wedge \\
&\qquad (\forall P'.(P \xrightarrow{z.\mathsf{inr}} P') \Rightarrow \exists Q'.(Q \overset{z.\mathsf{inr}}{\Longrightarrow} Q' \wedge P' \approx_{\mathrm{L}} Q' :: z{:}B[\eta : \omega \leftrightarrow \omega'])) \\
P \approx_{\mathrm{L}} Q :: z{:}A \oplus B[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff } \\
&\qquad (\forall P'.(P \xrightarrow{\overline{z.\mathsf{inl}}} P') \Rightarrow \exists Q'.(Q \overset{\overline{z.\mathsf{inl}}}{\Longrightarrow} Q' \wedge P' \approx_{\mathrm{L}} Q' :: z{:}A[\eta : \omega \leftrightarrow \omega'])) \wedge \\
&\qquad (\forall P'.(P \xrightarrow{\overline{z.\mathsf{inr}}} P') \Rightarrow \exists Q'.(Q \overset{\overline{z.\mathsf{inr}}}{\Longrightarrow} Q' \wedge P' \approx_{\mathrm{L}} Q' :: z{:}B[\eta : \omega \leftrightarrow \omega'])) \\
P \approx_{\mathrm{L}} Q :: z{:}\forall X.A[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; \forall B_1, B_2, P', \mathcal{R} :: -{:}B_1 \leftrightarrow B_2.(P \xrightarrow{z(B_1)} P') \Rightarrow \\
&\qquad \exists Q'.Q \overset{z(B_2)}{\Longrightarrow} Q',\; P' \approx_{\mathrm{L}} Q' :: z{:}A[\eta[X \mapsto \mathcal{R}] : \omega[X \mapsto B_1] \leftrightarrow \omega'[X \mapsto B_2]] \\
P \approx_{\mathrm{L}} Q :: z{:}\exists X.A[\eta : \omega \leftrightarrow \omega'] \;&\text{ iff }\; \exists B_1, B_2, \mathcal{R} :: -{:}B_1 \leftrightarrow B_2.(P \xrightarrow{z\langle B \rangle} P') \Rightarrow \\
&\qquad \exists Q'.Q \overset{z\langle B \rangle}{\Longrightarrow} Q',\; P' \approx_{\mathrm{L}} Q' :: z{:}A[\eta[X \mapsto \mathcal{R}] : \omega[X \mapsto B_1] \leftrightarrow \omega'[X \mapsto B_2]]
\end{aligned}
$$

Fig. 8. Logical equivalence (base case). See Definition 5.9.

$\eta : \omega \leftrightarrow \omega'$) as is inductively defined as follows:

$$
\begin{aligned}
\Gamma; \Delta, y : A \vdash P \approx_{\mathrm{L}} Q :: T[\eta : \omega \leftrightarrow \omega'] \;&\textit{ iff }\; \forall R_1, R_2.\, \textit{s.t.}\; R_1 \approx_{\mathrm{L}} R_2 :: y{:}A[\eta : \omega \leftrightarrow \omega'], \\
&\qquad \Gamma; \Delta \vdash (\nu y)(\hat{\omega}(P) \mid \hat{\omega}(R_1)) \approx_{\mathrm{L}} (\nu y)(\hat{\omega}'(Q) \mid \hat{\omega}'(R_2)) :: T[\eta : \omega \leftrightarrow \omega'] \\
\Gamma, u : A; \Delta \vdash P \approx_{\mathrm{L}} Q :: T[\eta : \omega \leftrightarrow \omega'] \;&\textit{ iff }\; \forall R_1, R_2.\, \textit{s.t.}\; R_1 \approx_{\mathrm{L}} R_2 :: y{:}A[\eta : \omega \leftrightarrow \omega'], \\
&\qquad \Gamma; \Delta \vdash (\nu y)(\hat{\omega}(P) \mid !u(y).\hat{\omega}(R_1)) \approx_{\mathrm{L}} (\nu y)(\hat{\omega}'(Q) \mid !u(y).\hat{\omega}'(R_2)) :: T[\eta : \omega \leftrightarrow \omega']
\end{aligned}
$$

This way, logical equivalence turns out to be a generalization of the logical predicate $\mathcal{T}^{\omega}_{\eta}[\Gamma; \Delta \vdash T]$ (Definition 4.5) to the binary setting. The key difference lies in the definition of candidate (here called equivalence candidate), which instead of guaranteeing termination, enforces closure under barbed congruence.

5.3 Parametricity and Coincidence Results

We are in place to state a *parametricity* result in our setting. The theorem below is the binary analog of the Fundamental Theorem (Theorem 4.15). Its proof follows a similar path: we first establish that our logical relation is one of the equivalence candidates, and then show that well-typed processes are logically equivalent to themselves.

Lemma 5.11. *Suppose $P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$. If, for any P_0, we have $P_0 \Longrightarrow P$ then $P_0 \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$. Also: if, for any Q_0, we have $Q_0 \Longrightarrow Q$ then $P \approx_{\mathrm{L}} Q_0 :: z{:}A[\eta : \omega \leftrightarrow \omega']$.*

Proof. By induction on the structure of A, relying on Subject Reduction (Theorem 2.11) and the definition of η. □

Lemma 5.12. *Suppose* $P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$. *If* $\cdot \vdash P \cong P' :: z{:}\hat{\omega}(A)$ *and* $\cdot \vdash Q \cong Q' :: z{:}\hat{\omega'}(A)$ *then* $P' \approx_{\mathrm{L}} Q' :: z{:}A[\eta : \omega \leftrightarrow \omega']$.

Proof. By induction on the structure of A, using contextuality of $\cong$. □

Theorem 5.13 (Logical Equivalence is an Equivalence Candidate). *Relation* $P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$ *is an equivalence candidate at* $z{:}\hat{\omega}(A)$ *and* $z{:}\hat{\omega'}(A)$*, in the sense of Definition 5.8.*

Proof. Immediate by Lemmas 5.11 and 5.12. □

The following is a generalization of Lemma 5.12:

Lemma 5.14. *Suppose* $\Gamma; \Delta \vdash P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$. *If* $\Gamma; \Delta \vdash P \cong P' :: z{:}\hat{\omega}(A)$ *and* $\Gamma; \Delta \vdash Q \cong Q' :: z{:}\hat{\omega'}(A)$ *then* $\Gamma; \Delta \vdash P' \approx_{\mathrm{L}} Q' :: z{:}A[\eta : \omega \leftrightarrow \omega']$.

Proof. By induction on the combined size of Γ and Δ, following Definition 5.10 and Theorem 5.13; the base case uses Lemma 5.12. □

Theorem 5.15 (Compositionality). *Let* B *be any type. Then,* $P \approx_{\mathrm{L}} Q :: z{:}A\{B/X\}[\eta : \omega \leftrightarrow \omega']$ *if and only if*

$$P \approx_{\mathrm{L}} Q :: z{:}A[\eta[X \mapsto \mathcal{R}] \ : \ \omega[X \mapsto \hat{\omega}(B)] \leftrightarrow \omega'[X \mapsto \hat{\omega'}(B)]]$$

where $\mathcal{R} :: -{:}\hat{\omega}(B) \leftrightarrow \hat{\omega'}(B)$ *stands for logical equivalence (cf. Definition 5.9).*

Proof. By induction on the structure of A, relying on Definition 5.9. □

Theorem 5.16 (Parametricity). *If* $\Omega; \Gamma; \Delta \vdash P :: z{:}A$ *then, for all* $\omega, \omega' : \Omega$ *and* $\eta : \omega \leftrightarrow \omega'$*, we have*

$$\Gamma; \Delta \vdash \hat{\omega}(P) \approx_{\mathrm{L}} \hat{\omega'}(P) :: z{:}A[\eta : \omega \leftrightarrow \omega']$$

Proof. By induction on the typing, exploiting Lemma 5.11, and Theorems 5.13 and 5.15. □

We now address the coincidence of barbed congruence and logical equivalence.

Lemma 5.17. *Logical equivalence (cf. Definition 5.10) is a contextual relation, in the sense of Definition 5.6.*

Proof. By induction on the clauses of Definition 5.6. □

Lemma 5.18. *If* $P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$ *then* $\cdot \vdash P \cong Q :: z{:}A$

Proof. Immediate from Lemma 5.12 (which ensures that $\approx_{\mathrm{L}}$ respects $\cong$) and reflexivity of $\cong$. □

Theorem 5.19 ($\approx_{\mathrm{L}}$ implies $\cong$ - Soundness). *If* $\Gamma; \Delta \vdash P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$ *holds for any* $\omega, \omega' : \Omega$ *and* $\eta : \omega \leftrightarrow \omega'$*, then* $\Omega; \Gamma; \Delta \vdash P \cong Q :: z{:}A$

Proof. Follows by contextuality of $\approx_{\mathrm{L}}$ (Lemma 5.17) which allows to simplify the analysis to the case of an empty left-hand side typing environment. In that point, the thesis follows by Lemma 5.18. □

Theorem 5.20 ($\cong$ implies $\approx_{\mathrm{L}}$ - Completeness). *If* $\Omega; \Gamma; \Delta \vdash P \cong Q :: z{:}A$ *then* $\Gamma; \Delta \vdash P \approx_{\mathrm{L}} Q :: z{:}A[\eta : \omega \leftrightarrow \omega']$ *for some* $\omega, \omega' : \Omega$ *and* $\eta : \omega \leftrightarrow \omega'$.

Proof. Follows by combining Theorem 5.16 and Lemma 5.14. □

Applications. Parametricity allows us to establish interesting behavioral properties of processes merely through typing. In particular, we can establish representation independence results. For instance, in the boolean server given in Section 2 where we can show that alternative equivalent implementations of booleans will result in processes that are behaviorally equivalent to those presented previously. Furthermore, we can appeal to parametricity to characterize processes, given their typing. For instance, using parametricity we can show that a session of type $z{:}\forall X.X{\multimap}X$ must be implemented by a process equivalent to $z(X).z(x).[x \leftrightarrow z]$, which makes precise the intuitive interaction of universal types and linearity (e.g., the only way to produce an arbitrary X is to input a session of that type and then consume it in its entirety).

6 Encoding System F

We describe the main ideas of a faithful encoding of System F into our polymorphic session-typed π-calculus. The encoding arises as a natural translation from second-order intuitionistic logic to second-order intuitionistic *linear* logic, where we interpret the implication $A \to B$ as $!A{\multimap}B$ (which by propositions-as-types can be read as a translation from functions to linear functions), and the second-order quantifier is mapped to its respective counterpart in the second-order linear setting.

The System F syntax is standard: terms M, N consist of functions over terms and types ($\lambda x{:}\tau$ and $\Lambda X.M$, resp.), and their applications ($M\ N$ and $M[\tau]$, resp.). Types τ, σ are functions and impredicative polymorphic quantification ($\tau \to \sigma$ and $\forall X.\tau$, resp.), with the usual formation/typing rules. Judgment $\Omega; \Gamma \vdash M{:}\sigma$ denotes that M has type σ, assuming the free variables in by Γ have the appropriate types, which might have free type variables that are recorded in Ω. In our encoding, we first translate System F into *Linear F* (a polymorphic linear λ-calculus); then, using a canonical translation from natural deduction to sequent calculus, we interpret the linear calculus in our session-typed π-calculus. For space reasons, we refrain from detailing on Linear F.

Definition 6.1 (From System F to Linear F). *The encoding of System F in Linear F, written $[\![\cdot]\!]$, is defined inductively on types and terms by the rules below:*

$$
\begin{array}{llll}
[\![\tau \to \sigma]\!] & \triangleq (!{[\![\tau]\!]}){\multimap}{[\![\sigma]\!]} & [\![\forall X.A]\!] & \triangleq \forall X.[\![A]\!] \\
[\![X]\!] & \triangleq X & [\![x]\!] & \triangleq u_x \\
[\![M\ N]\!] & \triangleq [\![M]\!]\,(![\![N]\!]) & [\![\lambda x:\tau.\, M]\!] & \triangleq \lambda x : !{[\![\tau]\!]}.\, \mathsf{let}\ !u_x = x\ \mathsf{in}\ [\![M]\!] \\
[\![\Lambda X.M]\!] & \triangleq \Lambda X.[\![M]\!] & [\![M[\tau]]\!] & \triangleq [\![M]\!][[\![\tau]\!]]
\end{array}
$$

The above encoding is a generalization of one of Girard's original encodings [16] to the second order setting (in particular, a call-by-name translation), mapping quantified types to quantified types in the expected way. The following correctness result follows by induction on typing (we denote by $[\![\Gamma]\!]$ the pointwise extension of $[\![\ ]\!]$ to Γ).

Theorem 6.2. *If $\Omega; \Gamma \vdash M : \tau$ then $\Omega; [\![\Gamma]\!]; \cdot \vdash [\![M]\!] : [\![\tau]\!]$.*

Due to space reasons, we omit the translation from Linear F into the session-typed π-calculus, and present the translation from System F into the π-calculus: the composition of $[\![\ \cdot\ ]\!]$ with the canonical translation from natural deduction to sequent calculus.

Definition 6.3 (From System F to Polymorphic π-calculus). *The encoding of System F terms in our session-typed polymorphic π-calculus is given by the following:*

$$
\begin{array}{llll}
{[\![x]\!]}_z & \triangleq (\nu x)u_x\langle x\rangle.[x \leftrightarrow z] & {[\![\Lambda X.M]\!]}_z & \triangleq\ z(X).{[\![M]\!]}_z \\
{[\![\lambda x.M]\!]}_z & \triangleq z(x).(\nu y)([x \leftrightarrow y] \mid {[\![M]\!]}_z\{y/u_x\}) & {[\![M[\tau]]\!]}_z & \triangleq\ (\nu x)({[\![M]\!]}_x \mid x\langle[\![\tau]\!]\rangle.[x \leftrightarrow z]) \\
{[\![M\ N]\!]}_z & \triangleq (\nu w)({[\![M]\!]}_w \mid (\nu y)w\langle y\rangle.((!y(x).{[\![N]\!]}_x) \mid [w \leftrightarrow z]))
\end{array}
$$

Theorem 6.4. *If $\Omega; \Gamma \vdash M : \tau$ then $\Omega; [\![\Gamma]\!]; \cdot \vdash {[\![M]\!]}_z :: z{:}[\![\tau]\!]$.*

Our encoding preserves the operational behavior of call-by-name System F. As expected in encodings of λ-calculus in the π-calculus, we must characterize (higher-order) substitution in the λ-calculus with (first-order) communication in the π-calculus. To this end, we identify process ${[\![M\{N/x\}]\!]}_z$ with $(\nu x)({[\![M]\!]}_z \mid !x(y).{[\![N]\!]}_y)$. Following [30], we define an auxiliary relation which allows us to identify the λ-term $M\{N/x\}$ with a process that is structurally equivalent to $(\nu x)({[\![M]\!]}_z \mid !x(y).{[\![N]\!]}_y)$. More precisely:

Definition 6.5. *For a given λ-term M and process P, we write $M \ll P$ if there exist $\widetilde{x}, \widetilde{u}$ and $\widetilde{E}$ such that: $P \equiv (\nu \widetilde{u}, \widetilde{x})(\llbracket R \rrbracket_z \mid !\widetilde{u(x).\llbracket E \rrbracket_x})$ and $M = R\{\widetilde{E}/\widetilde{x}\}$.*

Theorem 6.6 (Operational Correspondence). *Let $\llbracket \cdot \rrbracket_z$ be the encoding in Def. 6.3.*

1. *If $\Omega; \Gamma \vdash M : \tau$ and $M \to N$ then $\llbracket M \rrbracket_z \to^* P$ such that $N \ll P$.*
2. *If $\llbracket M \rrbracket_z \to P$ then there exists Q and N such that $P \to^* Q$, $M \to N$ and $N \ll Q$.*

Proof. By induction on the operational semantics of System F for (1) and on the operational semantics of the π-calculus for (2); see Appendix C.1 and C.2 for details.

7 Related Work

Although a number of works have studied polymorphism in the π-calculus, our work seems to be the first one in developing strong normalization and relational parametricity results for polymorphic session types, by relying solely on linear logic principles.

Turner [31] investigated a form of polymorphism based only on existential quantification for a simply-typed π-calculus (roughly, the discipline in which types of names describe the objects they can carry). In processes, polymorphism is expressed as explicit type parameters in input/output prefixes: messages are composed of a type and values of that type, and so type variables are relevant at the level of messages. Sangiorgi and Pierce [25] proposed a behavioral theory for Turner's framework. Neither of these works address termination of well-typed processes nor studies relational parametricity. Building up on [25], Jeffrey and Rathke [22] show that weak bisimulation is fully abstract for observational equivalence for an asynchronous polymorphic π-calculus with name equality testing.

In contrast to [31], and as we propose here, Berger et al. [5,6] proposed a polymorphic π-calculus with both universal and existential quantification. The type system considered in [5,6] is very different from ours, though: while we consider session types based solely on linear logic principles, typing in [5,6] relies on a combination of linearity, so-called action types, and duality principles. As in our case, they prove strong normalization of well-typed processes using reducibility candidates; however, due to the differences on the typing systems given above, the technical details of their proof cannot be compared to our developments. In particular, our application of the reducibility candidates technique generalizes the linear logical relations we have defined in [23]. While in [5] a relational parametricity result is reported, in [6] a behavioral theory based on generic transitions is put forward, together with a fully abstract embedding of System F. Here again detailed comparisons with our developments are not possible, because of the different typing disciplines considered in each case.

Previous works on polymorphism for session types include [14,12,7,32]. Based on the session types discipline in [13], Gay [14] studied *bounded polymorphism*, i.e., the form of polymorphism in which quantification over types is limited using subtyping. The bounded polymorphism of [14] is associated exclusively to branch and choice types: each branch is quantified by a type variable with upper and lower bounds; the scope of a variable is a whole branch of communication. Special types Bot and Top can be used as bounds, thus allowing for some forms of unbounded polymorphism. Dezani et al. [12] propose bounded polymorphism for a session-typed, object-oriented language. Differently from [14], the bounded polymorphism in [12] allows to bound all received values in session types. Bono and Padovani [7] use *unbounded* polymorphism in a variant of session types that is used to ensure communication correctness for copyless message passing programs. Independently, Wadler [32] has recently proposed an interpretation of classical linear logic as session types (along the lines of [9]) which accounts for parametric polymorphism. However, and in sharp contrast with our work, [32] does not develop the theory of polymorphic session types. In particular, results of strong normalization via linear logical relations/reducibility candidates and relational parametricity are not reported in [32].

8 Concluding Remarks

We have presented a theory of polymorphic session types, which results as the natural generalization of an interpretation of linear logic as session types [8,9]. In our framework, universal and existential quantification are interpreted

as a mechanism for type exchange; hence, parametric polymorphism enforces a form of communication of abstract protocols. We have shown that the polymorphic session calculus is strongly normalizing, and that it enjoys a principle of relational parametricity. Our framework defines a novel account for polymorphism in a concurrent setting, as it naturally allows for *behavioral* genericity, which is well beyond usual genericity in data/values.

Having defined a basis for parametric polymorphism, a promising topic for future work is a precise understanding of inductive and co-inductive types in a concurrent setting. We would also like to explore subtyping and develop an understanding of type operators in the style of System F_ω, extended to our concurrent setting. Following a different axis of the λ cube, we would like to develop a fully dependent type theory for concurrency, generalizing the work in [29,10] to allow for processes as indices to types.

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. E. Beffara. A concurrent model for linear logic. *Electron. Notes Theor. Comput. Sci.*, 155:147–168, May 2006.
4. G. Bellin and P. J. Scott. On the π-calculus and linear logic. *Theor. Comput. Sci.*, 135:11–65, April 1992.
5. M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. In *Proc. of FoSSaCS*, volume 2620 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2003.
6. M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. *Acta Inf.*, 42(2-3):83–141, 2005.
7. V. Bono and L. Padovani. Polymorphic endpoint types for copyless message passing. In *Proc. of ICE'11*, volume 59 of *EPTCS*, pages 52–67, 2011.
8. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR'2010*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
9. L. Caires, F. Pfenning, and B. Toninho. Linear logic propositions as session types, 2012. Under Revision - `http://www.cs.cmu.edu/~fp/papers/sessions12.pdf`.
10. L. Caires, F. Pfenning, and B. Toninho. Towards concurrent type theory. In *TLDI'12*, pages 1–12, New York, NY, USA, 2012. ACM.
11. 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.
12. M. Dezani-Ciancaglini, E. Giachino, S. Drossopoulou, and N. Yoshida. Bounded session types for object oriented languages. In *Proc. of FMCO'06*, volume 4709 of *Lecture Notes in Computer Science*, pages 207–245. Springer, 2007.
13. S. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42:191–225, November 2005.
14. S. J. Gay. Bounded polymorphism in session types. *Math. Struc. in Comp. Sci.*, 18(5):895–930, 2008.
15. J. Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination de coupures dans l'analyse et la théorie des types. In *Proc. of the 2nd Scandinavian Logic Symposium*, pages 63–92. North-Holland Publishing Co., 1971.
16. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
17. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, 1989.
18. R. Harper. *Practical Foundations for Programming Languages*. Working Draft, 2012.
19. K. Honda. Types for dynamic interaction. In *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.
20. K. Honda and O. Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theor. Comput. Sci.*, 411(22-24):2223–2238, 2010.
21. 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.
22. A. Jeffrey and J. Rathke. Full abstraction for polymorphic pi-calculus. *Theor. Comput. Sci.*, 390(2-3):171–196, 2008.
23. J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations for session-based concurrency. In *Proc. of ESOP*, volume 7211 of *LNCS*, pages 539–558. Springer, 2012.
24. 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.

25. B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *J. ACM*, 47(3):531–584, 2000.
26. J. C. Reynolds. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation*, pages 408–423, London, UK, UK, 1974. Springer-Verlag.
27. J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523. Elsevier Science Publishers B. V., 1983.
28. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.
29. 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.
30. B. Toninho, L. Caires, and F. Pfenning. Functions as session-typed processes. In *Proc. of FoSSaCS'12*, volume 7213 of *LNCS*, pages 346–360. Springer, 2012.
31. D. Turner. The polymorphic pi-calculus: Theory and implementation. Technical report, ECS-LFCS-96-345, Univ. of Edinburgh, 1996.
32. P. Wadler. Propositions as sessions. In P. Thiemann and R. B. Findler, editors, *ICFP*, pages 273–286. ACM, 2012.

A Proofs

A.1 Proof of Lemma 2.9 (Page 5)

Proof. By simultaneous induction on D and E. The possible cases for D are $\forall\mathsf{R}$, $\mathbf{1}\mathsf{L}$, $!\mathsf{L}$, cut and $\mathsf{cut}^!$. The possible cases for E are $\forall\mathsf{L}$, $\mathbf{1}\mathsf{L}$, $!\mathsf{L}$, cut and $\mathsf{cut}^!$.

Case: $D = \forall\mathsf{R}\ (X.\ D_1)$ and $E = \forall\mathsf{L}\ x\ A\ (x.\ E_1)$.

$D \rightsquigarrow x(X).P_1$ and $E \rightsquigarrow x\langle A\rangle.Q_1$
$\mathsf{cut}\ D\ (x.\ E) \equiv\Rightarrow\equiv \mathsf{cut}\ D_1\{A/X\}\ (x.\ E_1) = F$ by $(\mathsf{cut}/\forall\mathsf{R}/\forall\mathsf{L})$
$F \rightsquigarrow (\nu x)(P_1\{A/X\} \mid Q_1)$
$P' \equiv P_1\{A/X\}$ and $Q' \equiv Q_1$

Case: $D = \mathbf{1}\mathsf{L}\ y\ D_1$ and E arbitrary.

$D_1 \rightsquigarrow P$ by inversion
$\mathsf{cut}\ D\ (x.\ E) = \mathsf{cut}\ (\mathbf{1}\mathsf{L}\ y\ D_1)\ (x.\ E)$
$\equiv \mathbf{1}\mathsf{L}\ y\ (cut\ D_1\ (x.E))$ by $(\mathsf{cut}/\mathbf{1}\mathsf{L}/-)$
$\Rightarrow \mathbf{1}\mathsf{L}\ y\ F' = F$ for some $F' \rightsquigarrow (\nu x)(P' \mid Q')$ by i.h. on D_1 and E
$F \rightsquigarrow (\nu x)(P' \mid Q')$

Case: $D = !\mathsf{L}\ n\ (u.\ D')$ and E arbitrary.

$D' \rightsquigarrow P$ by inversion
$\mathsf{cut}\ D\ (x.\ E) = \mathsf{cut}\ (!\mathsf{L}\ n\ (u.\ D'))\ (x.\ E)$
$\equiv !\mathsf{L}\ n\ (u.\ \mathsf{cut}\ D'\ (x.\ E))$ by $(\mathsf{cut}/!\mathsf{L}/-)$
$\Rightarrow !\mathsf{L}\ n\ (u.\ F') = F$ for some $F' \rightsquigarrow (\nu x)(P' \mid Q')$ by i.h. D' and E
$F \rightsquigarrow (\nu x)(P' \mid Q')$

Case: $D = \mathsf{cut}\ D_1\ (n.\ D_2)$ and E arbitrary.

$D_1 \rightsquigarrow P_1^n$ and $D_2 \rightsquigarrow P_2^x$ and $P_2^x \overset{x(A)}{\rightarrow} P_2'^x$ where
$P = (\nu n)(P_1^n \mid P_2^x) \overset{x(A)}{\rightarrow} (\nu n)(P_1^n \mid P_2'^x) = P'^x$ by inversion
$\mathsf{cut}\ D\ (x.\ E) = \mathsf{cut}\ (\mathsf{cut}\ D_1\ (n.\ D_2))\ (x.\ E)$
$\equiv \mathsf{cut}\ D_1\ (n.\ \mathsf{cut}\ D_2\ (x.\ E))$ by $(\mathsf{cut}/\mathsf{cut}/-)$
$\Rightarrow \mathsf{cut}\ D_1\ (n.\ F_2) = F$ where $F_2 \rightsquigarrow (\nu x)(P_2'^x \mid Q'^z)$ by i.h. on D_2 and E
$F \rightsquigarrow (\nu n)(P_1^n \mid (\nu x)(P_2'^x \mid Q'^z))$
$\equiv (\nu x)((\nu n)(P_1^n \mid P_2'^x) \mid Q'^z)$
$= (\nu x)(P'^x \mid Q'^z)$

Case: $D = \mathsf{cut}^!\ D_1\ (u.\, D_2)$ and E arbitrary.

$D_1 \rightsquigarrow (!u(n).P_1^n)$ and $D_2 \rightsquigarrow P_2^x$ and $P_2^x \overset{x(A)}{\to} P_2'^x$ where
$P = (\nu u)((!u(n).P_1^n) \mid P_2^x) \overset{x(A)}{\to} (\nu u)((!u(n).P_1^n) \mid P_2'^x) = P'^x$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ (\mathsf{cut}^!\ D_1\ (u.\, D_2))\ (x.\, E)$
$\equiv \mathsf{cut}^!\ D_1\ (u.\, \mathsf{cut}\ D_2\ (x.\, E))$ by $(\mathsf{cut}/\mathsf{cut}^!/-)$
$\Rightarrow \mathsf{cut}\ D_1\ (u.\, F_2) = F$ where $F_2 \rightsquigarrow (\nu x)(P_2'^x \mid Q'^z)$ By i.h. on D_2 and E
$F \rightsquigarrow (\nu u)((!u(n).P_1^n) \mid (\nu x)(P_2'^x \mid Q'^z))$
$\equiv (\nu x)((\nu u)((!u(n).P_1^n) \mid P_2'^x) \mid Q'^z)$
$= (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathbf{1}\mathsf{L}\ n\ E'$ and D arbitrary.

$E' \rightsquigarrow Q^z$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathbf{1}\mathsf{L}\ n\ E')$
$\equiv \mathbf{1}\mathsf{L}\ n\ (\mathsf{cut}\ D\ (x.\, E'))$ By $(\mathsf{cut}/-/\mathbf{1}\mathsf{L})$
$\Rightarrow \mathbf{1}\mathsf{L}\ n\ F' = F$ for some $F' \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$ by i.h. on D and E'
$F \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$

Case: $E = !\mathsf{L}\ n\ (u.\, E')$ and D arbitrary.

$E' \rightsquigarrow Q^z$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, !\mathsf{L}\ (u.\, E')$
$\equiv !\mathsf{L}\ n\ (u.\, \mathsf{cut}\ D\ (x.\, E'))$ by rule $(\mathsf{cut}/-/!\mathsf{L})$
$\Rightarrow !\mathsf{L}\ n\ (u.\, F') = F$ for some $F' \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$ by i.h. on D and E'
$F \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathsf{cut}\ E_1\ (n.\, E_2)$ and $x \in FV(E_1)$.

$E_1 \rightsquigarrow Q_1^n$ and $E_2 \rightsquigarrow Q_2^z$ for $Q_1^n \overset{\overline{x\langle A\rangle}}{\to} Q_1'^n$ where
$Q = (\nu n)(Q_1^n \mid Q_2^z) \overset{\overline{x\langle A\rangle}}{\to} (\nu n)(Q_1'^n \mid Q_2^z)$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathsf{cut}\ E_1\ (n.\, E_2))$
$= \mathsf{cut}\ (\mathsf{cut}\ D\ (x.\, E_1))\ (n.\, E_2)$ by $(\mathsf{cut}/\mathsf{cut}/-)$ and $x \notin FV(E_2)$
$\Rightarrow \mathsf{cut}\ F_1\ (n.\, E_2) = F$ for some $F_1 \rightsquigarrow (\nu x)(P'^x \mid Q_1'^n)$ by i.h. on D and E_1
$F \rightsquigarrow (\nu n)((\nu x)(P'^x \mid Q_1'^n) \mid Q_2^z)$
$\equiv (\nu x)(P'^x \mid (\nu n)(Q_1'^n \mid Q_2^z))$
$= (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathsf{cut}\ E_1\ (n.\, E_2)$ and $x \in FV(E_2)$.

$E_1 \rightsquigarrow Q_1^n$ and $E_2 \rightsquigarrow Q_2^z$ for $Q_2^z \overset{\overline{x\langle A\rangle}}{\to} Q_2'^z$ where
$Q = (\nu n)(Q_1^n \mid Q_2^z) \overset{\overline{x\langle A\rangle}}{\to} (\nu n)(Q_1^n \mid Q_2'^z)$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathsf{cut}\ E_1\ (n.\, E_2))$
$= \mathsf{cut}\ E_1\ (n.\, \mathsf{cut}\ D\ (x.\, E_2))$ by $(\mathsf{cut}/-/\mathsf{cut})$ and $x \notin FV(E_1)$
$\Rightarrow \mathsf{cut}\ E_1\ (n.\, F_2) = F$ for some $F_2 \rightsquigarrow (\nu x)(P'^x \mid Q_2'^z)$ by i.h. on D and E_2
$F \rightsquigarrow (\nu n)(Q_1^n \mid (\nu x)(P'^x \mid Q_2'^z))$
$\equiv (\nu x)(P'^x \mid (\nu n)(Q_1^n \mid Q_2'^z))$
$= (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathsf{cut}^!\ E_1\ (u.\, E_2)$

$x \in FV(E_2)$ by inversion

$E_1 \rightsquigarrow (!u(n).Q_1^n)$ and $E_2 \rightsquigarrow Q_2^z$ for $Q_2^z \overset{\overline{x\langle A\rangle}}{\to} Q_2'^z$ where

$Q = (\nu u)((!u(n).Q_1^n) \mid Q_2^z) \overset{\overline{x\langle A\rangle}}{\to} (\nu u)((!u(n).Q_1^n) \mid Q_2'^z)$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathsf{cut}^!\ E_1\ (u.\, E_2))$
$= \mathsf{cut}^!\ E_1\ (u.\, \mathsf{cut}\ D\ (x.\, E_2))$ by $(\mathsf{cut}/-/\mathsf{cut}^!)$
$\Rightarrow \mathsf{cut}^!\ E_1\ (u.\, F_2) = F$ for some $F_2 \rightsquigarrow (\nu x)(P'^x \mid Q_2'^z)$ by i.h. on D and E_2

$F \rightsquigarrow (\nu u)((!u(n).Q_1^n) \mid (\nu x)(P'^x \mid Q_2'^z))$
$\equiv (\nu x)(P'^x \mid (\nu u)((!u(n).Q_1^n) \mid Q_2'^z))$
$= (\nu x)(P'^x \mid Q'^z)$

□

A.2 Proof of Lemma 2.10 (Page 6)

Proof. By simultaneous induction on D and E. The possible cases for D are $\exists\mathsf{R}$, $\mathbf{1}\mathsf{L}$, $!\mathsf{L}$, cut and $\mathsf{cut}^!$. The possible cases for E are $\exists\mathsf{L}$, $\mathbf{1}\mathsf{L}$, $!\mathsf{L}$, cut and $\mathsf{cut}^!$.

Case: $D = \exists\mathsf{R}\ A\ D_1$ and $E = \exists\mathsf{L}\ x\ (X.x.\, E_1)$.

$D \rightsquigarrow x\langle A\rangle.P_1$ and $E \rightsquigarrow x(X).Q_1$
$\mathsf{cut}\ D\ (x.\, E) \equiv \Rightarrow \equiv \mathsf{cut}\ D_1\ (x.\, E_1\{A/X\}) = F$ by $(\mathsf{cut}/\exists\mathsf{R}/\exists\mathsf{L})$
$F \rightsquigarrow (\nu x)(P_1 \mid Q_1\{A/X\})$
$P' \equiv P_1$ and $Q' \equiv Q_1\{A/X\}$

Case: $D = \mathbf{1}\mathsf{L}\ y\ D_1$ and E arbitrary.

$D_1 \rightsquigarrow P$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ (\mathbf{1}\mathsf{L}\ y\ D_1)\ (x.\, E)$
$\equiv \mathbf{1}\mathsf{L}\ y\ (cut\ D_1\ (x.E))$ by $(\mathsf{cut}/\mathbf{1}\mathsf{L}/-)$
$\Rightarrow \mathbf{1}\mathsf{L}\ y\ F' = F$ for some $F' \rightsquigarrow (\nu x)(P' \mid Q')$ by i.h. on D_1 and E
$F \rightsquigarrow (\nu x)(P' \mid Q')$

Case: $D = !\mathsf{L}\ n\ (u.\, D')$ and E arbitrary.

$D' \rightsquigarrow P$ by inversion
$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ (!\mathsf{L}\ n\ (u.\, D'))\ (x.\, E)$
$\equiv !\mathsf{L}\ n\ (u.\, \mathsf{cut}\ D'\ (x.\, E))$ by $(\mathsf{cut}/!\mathsf{L}/-)$
$\Rightarrow !\mathsf{L}\ n\ (u.\, F') = F$ for some $F' \rightsquigarrow (\nu x)(P' \mid Q')$ by i.h. D' and E
$F \rightsquigarrow (\nu x)(P' \mid Q')$

Case: $D = \mathsf{cut}\ D_1\ (n.\, D_2)$ and E arbitrary.

$D_1 \rightsquigarrow P_1^n$ and $D_2 \rightsquigarrow P_2^x$ and $P_2^x \overset{\overline{x\langle A\rangle}}{\to} P_2'^x$ where
$P = (\nu n)(P_1^n \mid P_2^x) \overset{\overline{x\langle A\rangle}}{\to} (\nu n)(P_1^n \mid P_2'^x) = P'^x$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ (\mathsf{cut}\ D_1\ (n.\, D_2))\ (x.\, E)$
$\equiv \mathsf{cut}\ D_1\ (n.\, \mathsf{cut}\ D_2\ (x.\, E))$ by $(\mathsf{cut}/\mathsf{cut}/-)$
$\Rightarrow \mathsf{cut}\ D_1\ (n.\, F_2) = F$ where $F_2 \rightsquigarrow (\nu x)(P_2'^x \mid Q'^z)$ by i.h. on D_2 and E

$F \rightsquigarrow (\nu n)(P_1^n \mid (\nu x)(P_2'^x \mid Q'^z))$
$\equiv (\nu x)((\nu n)(P_1^n \mid P_2'^x) \mid Q'^z)$
$= (\nu x)(P'^x \mid Q'^z)$

Case: $D = \mathsf{cut}^!\ D_1\ (u.\, D_2)$ and E arbitrary.

$D_1 \rightsquigarrow (!u(n).P_1^n)$ and $D_2 \rightsquigarrow P_2^x$ and $P_2^x \overset{\overline{x\langle A\rangle}}{\to} P_2'^x$ where

$P = (\nu u)((!u(n).P_1^n) \mid P_2^x) \overset{\overline{x\langle A\rangle}}{\to} (\nu u)((!u(n).P_1^n) \mid P_2'^x) = P'^x$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ (\mathsf{cut}^!\ D_1\ (u.\, D_2))\ (x.\, E)$

$\equiv \mathsf{cut}^!\ D_1\ (u.\, \mathsf{cut}\ D_2\ (x.\, E))$ by $(\mathsf{cut}/\mathsf{cut}^!/-)$

$\Rightarrow \mathsf{cut}\ D_1\ (u.\, F_2) = F$ where $F_2 \rightsquigarrow (\nu x)(P_2'^x \mid Q'^z)$ By i.h. on D_2 and E

$F \rightsquigarrow (\nu u)((!u(n).P_1^n) \mid (\nu x)(P_2'^x \mid Q'^z))$

$\equiv (\nu x)((\nu u)((!u(n).P_1^n) \mid P_2'^x) \mid Q'^z)$

$= (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathbf{1}\mathsf{L}\ n\ E'$ and D arbitrary.

$E' \rightsquigarrow Q^z$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathbf{1}\mathsf{L}\ n\ E')$

$\equiv \mathbf{1}\mathsf{L}\ n\ (\mathsf{cut}\ D\ (x.\, E'))$ By $(\mathsf{cut}/-/\mathbf{1}\mathsf{L})$

$\Rightarrow \mathbf{1}\mathsf{L}\ n\ F' = F$ for some $F' \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$ by i.h. on D and E'

$F \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$

Case: $E = !\mathsf{L}\ n\ (u.\, E')$ and D arbitrary.

$E' \rightsquigarrow Q^z$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, !\mathsf{L}\ (u.\, E')$

$\equiv !\mathsf{L}\ n\ (u.\, \mathsf{cut}\ D\ (x.\, E'))$ by rule $(\mathsf{cut}/-/!\mathsf{L})$

$\Rightarrow !\mathsf{L}\ n\ (u.\, F') = F$ for some $F' \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$ by i.h. on D and E'

$F \rightsquigarrow (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathsf{cut}\ E_1\ (n.\, E_2)$ and $x \in FV(E_1)$.

$E_1 \rightsquigarrow Q_1^n$ and $E_2 \rightsquigarrow Q_2^z$ for $Q_1^n \overset{x(A)}{\to} Q_1'^n$ where

$Q = (\nu n)(Q_1^n \mid Q_2^z) \overset{x(A)}{\to} (\nu n)(Q_1'^n \mid Q_2^z)$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathsf{cut}\ E_1\ (n.\, E_2))$

$= \mathsf{cut}\ (\mathsf{cut}\ D\ (x.\, E_1))\ (n.\, E_2)$ by $(\mathsf{cut}/\mathsf{cut}/-)$ and $x \notin FV(E_2)$

$\Rightarrow \mathsf{cut}\ F_1\ (n.\, E_2) = F$ for some $F_1 \rightsquigarrow (\nu x)(P'^x \mid Q_1'^n)$ by i.h. on D and E_1

$F \rightsquigarrow (\nu n)((\nu x)(P'^x \mid Q_1'^n) \mid Q_2^z)$

$\equiv (\nu x)(P'^x \mid (\nu n)(Q_1'^n \mid Q_2^z))$

$= (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathsf{cut}\ E_1\ (n.\, E_2)$ and $x \in FV(E_2)$.

$E_1 \rightsquigarrow Q_1^n$ and $E_2 \rightsquigarrow Q_2^z$ for $Q_2^z \overset{x(A)}{\to} Q_2'^z$ where

$Q = (\nu n)(Q_1^n \mid Q_2^z) \overset{x(A)}{\to} (\nu n)(Q_1^n \mid Q_2'^z)$ by inversion

$\mathsf{cut}\ D\ (x.\, E) = \mathsf{cut}\ D\ (x.\, \mathsf{cut}\ E_1\ (n.\, E_2))$

$= \mathsf{cut}\ E_1\ (n.\, \mathsf{cut}\ D\ (x.\, E_2))$ by $(\mathsf{cut}/-/\mathsf{cut})$ and $x \notin FV(E_1)$

$\Rightarrow \mathsf{cut}\ E_1\ (n.\, F_2) = F$ for some $F_2 \rightsquigarrow (\nu x)(P'^x \mid Q_2'^z)$ by i.h. on D and E_2

$F \rightsquigarrow (\nu n)(Q_1^n \mid (\nu x)(P'^x \mid Q_2'^z))$

$\equiv (\nu x)(P'^x \mid (\nu n)(Q_1^n \mid Q_2'^z))$

$= (\nu x)(P'^x \mid Q'^z)$

Case: $E = \mathsf{cut}^!\ E_1\ (u.\, E_2)$

$x \in FV(E_2)$ by inversion

$E_1 \rightsquigarrow (!u(n).Q_1^n)$ and $E_2 \rightsquigarrow Q_2^z$ for $Q_2^z \overset{x(A)}{\to} Q_2'^z$ where

$Q = (\nu u)((!u(n).Q_1^n) \mid Q_2^z) \overset{x(A)}{\rightarrow} (\nu u)((!u(n).Q_1^n) \mid Q_2'^z)$ by inversion

$\mathsf{cut}\ D\ (x.\ E) = \mathsf{cut}\ D\ (x.\ \mathsf{cut}^!\ E_1\ (u.\ E_2))$
$= \mathsf{cut}^!\ E_1\ (u.\ \mathsf{cut}\ D\ (x.\ E_2))$ by $(\mathsf{cut}/-/\mathsf{cut}^!)$
$\Rightarrow \mathsf{cut}^!\ E_1\ (u.\ F_2) = F$ for some $F_2 \rightsquigarrow (\nu x)(P'^x \mid Q_2'^z)$ by i.h. on D and E_2

$F \rightsquigarrow (\nu u)((!u(n).Q_1^n) \mid (\nu x)(P'^x \mid Q_2'^z))$
$\equiv (\nu x)(P'^x \mid (\nu u)((!u(n).Q_1^n) \mid Q_2'^z))$
$= (\nu x)(P'^x \mid Q'^z)$

□

B Proof of Theorem 4.10 (The Logical Predicate is a Reducibility Candidate, Page 11)

Proof. We proceed by induction on the structure of the type A.

Case: $A = X$

$P \in \eta(X)(z)$ by Def.
Result follows by Def. of $\eta(X)(z)$.

Case: $A = A_1 \multimap A_2$

T.S: If for all P_i such that $P \Longrightarrow P_i$ we have $P_i \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$ then $P \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$
S.T.S: $\forall P''y.(P \overset{z(y)}{\Longrightarrow} P'') \Rightarrow \forall Q \in \mathcal{T}_\eta^\omega[y : A].(\nu y)(P'' \mid Q) \in \mathcal{T}_\eta^\omega[z{:}B]$
$\forall P''y.(P_i \overset{z(y)}{\Longrightarrow} P'') \Rightarrow \forall Q \in \mathcal{T}_\eta^\omega[y : A].(\nu y)(P'' \mid Q) \in \mathcal{T}_\eta^\omega[z{:}B]$ (a) by Def. 4.5
$P \Longrightarrow P_i \overset{z(y)}{\Longrightarrow} P''$ by assumption and Def.
$P \overset{z(y)}{\Longrightarrow} P''$ (b) by the reasoning above
$\forall Q \in \mathcal{T}_\eta^\omega[y : A].(\nu y)(P'' \mid Q) \in \mathcal{T}_\eta^\omega[z{:}B]$ (c) by (a) and (b)
$P \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$ by (c), satisfying (3)

T.S: If $P \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$ then $P \Downarrow P \overset{z(y)}{\Longrightarrow} P'$ (a) by assumption, for some P'
$\forall Q \in \mathcal{T}_\eta^\omega[y : A].(\nu y)(P' \mid Q) \in \mathcal{T}_\eta^\omega[z{:}B]$ by (a) and $P \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$
$Q \Downarrow$ by i.h.
$(\nu y)(P' \mid Q) \Downarrow$ (b) by i.h.
$P' \Downarrow$ (c) by (b)
$P \Downarrow$ by (c) and (a), satisfying (2)
T.S.: If $P \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$ and $P \Longrightarrow P'$ then $P' \in \mathcal{T}_\eta^\omega[z{:}A_1 \multimap A_2]$
Follows directly from definition of weak transition.

Case: $A = A_1 \otimes A_2$
Similar to above.
Case: $A = A_1 \,\&\, A_2$
Similar to above.
Case: $A = A_1 \oplus A_2$
Similar to above.
Case: $A = !A_1$
Similar to above.
Case: $A = \forall X.A_1$

T.S: If for all P_i such that $P \Longrightarrow P_i$ we have $P_i \in \mathcal{T}_\eta^\omega[z{:}\forall X.A_1]$ then $P \in \mathcal{T}_\eta^\omega[z{:}\forall X.A_1]$
S.T.S: $\forall B, P', \mathsf{R}[w : B].\ (B\ \mathsf{type} \wedge P \overset{z(B)}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}_{\eta[X \mapsto \mathsf{R}[w:B]]}^{\omega[X \mapsto B]}[z{:}A]$

$\forall B, P', \mathsf{R}[w:B].\ (B\ \mathsf{type} \wedge P_i \overset{z(B)}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}^{\omega[X\mapsto B]}_{\eta[X\mapsto \mathsf{R}[w:B]]}[z{:}A])$

(a) by Def. of $P_i \in \mathcal{T}^{\omega}_{\eta}[z{:}\forall X.A_1]$

$P \Longrightarrow P_i \overset{z(B)}{\Longrightarrow} P'$ (b) by assumption and (a)

$P \overset{z(B)}{\Longrightarrow} P'$ (c)

$P' \in \mathcal{T}^{\omega[X\mapsto B]}_{\eta[X\mapsto \mathsf{R}[w:B]]}[z{:}A]$ (d) by (a) and (b)

$P \in \mathcal{T}^{\omega}_{\eta}[z{:}\forall X.A_1]$ by (c) and (d), satisfying (3)

T.S: If $P \in \mathcal{T}^{\omega}_{\eta}[z{:}\forall X.A_1]$ then $P \Downarrow$

$\forall B, P', \mathsf{R}[w:B].\ (B\ \mathsf{type} \wedge P \overset{z(B)}{\Longrightarrow} P') \Rightarrow P' \in \mathcal{T}^{\omega[X\mapsto B]}_{\eta[X\mapsto \mathsf{R}[w:B]]}[z{:}A])$ (a) by Def. 4.5

$P \overset{z(B)}{\Longrightarrow} P'$ (b) for some P', B

$P' \in \mathcal{T}^{\omega[X\mapsto B]}_{\eta[X\mapsto \mathsf{R}[w:B]]}[z{:}A]$ by (a) and (b)

$P' \Downarrow$ (c) by i.h.

$P \Downarrow$ by (b) and (c), satisfying (2)

T.S.: If $P \in \mathcal{T}^{\omega}_{\eta}[z{:}\forall X.A_1]$ and $P \Longrightarrow P'$ then $P' \in \mathcal{T}^{\omega}_{\eta}[z{:}\forall X.A_1]$

Follows directly from definition of weak transition.

□

C System F Encoding

We need the following compositionality lemma to handle type substitution.

Lemma C.1 (Compositionality). $[\![M\{\tau/X\}]\!]_z \equiv [\![M]\!]_z\{[\![\tau]\!]/X\}$

Proof. Straightforward induction on the structure of M. □

C.1 Proof of Theorem 6.6 (1)

Proof. **Case:** $(\lambda x.M)\, N \to M\{N/x\}$

$[\![(\lambda x.M)\, N]\!]_z \to^* (\nu u)([\![M]\!]_z \mid\ !u(x).[\![N]\!]_x)$ by def. 6.3, satisfying def. 6.5.

Case: $(\Lambda X.M)[\tau] \to M\{\tau/X\}$

$[\![(\Lambda X.M)[\tau]]\!]_z \to^* [\![M]\!]_z\{[\![\tau]\!]/X\}$ by def. 6.3

$[\![M]\!]_z\{[\![\tau]\!]/X\} \equiv [\![M\{\tau/X\}]\!]_z$ by Lemma C.1, satisfying def. 6.5

Case: $M\, N \to M'\, N$ with $M \to M'$

$[\![M]\!]_w \to^* P'$ with $M' \ll P'$ (a) by i.h.

$[\![M\, N]\!]_z \to^* (\nu w)(P' \mid (\nu y)w\langle y\rangle.((!y(x).[\![N]\!]_x) \mid [w \leftrightarrow z])) = Q$ by def. 6.3 and (a)

$[\![M'\, N]\!]_z = (\nu w)([\![M']\!]_w \mid (\nu y)w\langle y\rangle.((!y(x).[\![N]\!]_x) \mid [w \leftrightarrow z]))$ by def. 6.3

T.S: $M'\, N \ll Q$

S.T.S: $Q \equiv (\nu \widetilde{u}, \widetilde{x})([\![R]\!]_z \mid \widetilde{!u(x).[\![E]\!]_x})$ with $M'\, N = R\{\widetilde{E}/\widetilde{x}\}$

$P' \equiv (\nu \widetilde{u'}, \widetilde{x'})([\![R']\!]_w \mid \widetilde{!u(x).[\![E']\!]_x})$ with $M' = R'\{\widetilde{E'}/\widetilde{x'}\}$ by (a)

$Q \equiv (\nu w)((\nu \widetilde{u'}, \widetilde{x'})([\![R']\!]_w \mid \widetilde{!u(x).[\![E']\!]_x}) \mid (\nu y)w\langle y\rangle.((!y(x).[\![N]\!]_x) \mid [w \leftrightarrow z])) \equiv$

$(\nu w, \widetilde{u'}, \widetilde{x'})([\![R']\!]_w \mid (\nu y)w\langle y\rangle.((!y(x).[\![N]\!]_x) \mid [w \leftrightarrow z]) \mid \widetilde{!u(x).[\![E']\!]_x})$

$[\![R'\, N]\!]_z = (\nu w)([\![R']\!]_w \mid (\nu y)w\langle y\rangle.((!y(x).[\![N]\!]_x) \mid [w \leftrightarrow z]))$ by def. 6.3

$(R'\, N)\{\widetilde{E'}/\widetilde{x'}\} = R'\{\widetilde{E'}/\widetilde{x'}\}\, N = M'\, N$

satisfying def. 6.5 with $R = R'\, N$, $\widetilde{E} = \widetilde{E'}$ and $\widetilde{x} = \widetilde{x'}$

Other cases follow by similar reasoning.

□

C.2 Proof of Theorem 6.6 (2)

Proof. By induction on the operational semantics of the π-calculus and case analysis on def. 6.3. The possibilities for M are $M_1\, M_2$ and $M_1[\tau]$, otherwise the process in the image of the translation cannot offer a reduction.

Case: $M = M_1 M_2$

$$[\![M]\!]_z = (\nu w)([\![M_1]\!]_w \mid (\nu y)w\langle y\rangle.((!y(x).[\![M_2]\!]_x) \mid [w \leftrightarrow z]))$$

with $[\![M_1]\!]_w \to P$

$P \to^* Q'$, $M_1 \to N_1$ with $N_1 \ll Q'$, for some Q' (a) by i.h.
$[\![M]\!]_z \to^* (\nu w)(Q' \mid (\nu y)w\langle y\rangle.((!y(x).[\![M_2]\!]_x) \mid [w \leftrightarrow z])) = P'$ by (a)
$M_1\, M_2 \to N_1\, M_2$ by (a)
S.T.S: $N_1\, M_2 \ll P'$
S.T.S: $P' \equiv (\nu \widetilde{u}, \widetilde{x})([\![R]\!]_z \mid \widetilde{!u(x).[\![E]\!]_x})$ with $N_1\, M_2 = R\{\widetilde{E}/\widetilde{x}\}$
$Q' \equiv (\nu \widetilde{u'}, \widetilde{x'})([\![R']\!]_w \mid \widetilde{!u(x).[\![E']\!]_x})$ with $N_1 = R'\{\widetilde{E'}/\widetilde{x'}\}$ by (a)
$P' \equiv (\nu w)((\nu \widetilde{u'}, \widetilde{x'})([\![R']\!]_w \mid \widetilde{!u(x).[\![E']\!]_x}) \mid (\nu y)w\langle y\rangle.((!y(x).[\![M_2]\!]_x) \mid [w \leftrightarrow z])) \equiv$
$(\nu w, \widetilde{u'}, \widetilde{x'})([\![R']\!]_w \mid (\nu y)w\langle y\rangle.((!y(x).[\![M_2]\!]_x) \mid [w \leftrightarrow z]) \mid \widetilde{!u(x).[\![E']\!]_x})$
$[\![R'\, M_2]\!]_z = (\nu w)([\![R']\!]_w \mid (\nu y)w\langle y\rangle.((!y(x).[\![M_2]\!]_x) \mid [w \leftrightarrow z]))$ by def. 6.3
$(R'\, M_2)\{\widetilde{E'}/\widetilde{x'}\} = R'\{\widetilde{E'}/\widetilde{x'}\}\, N = N_1\, M_2$
satisfying def. 6.5 with $R = R'\, N$, $\widetilde{E} = \widetilde{E'}$ and $\widetilde{x} = \widetilde{x'}$

Case: $M = M_1\, M_2$ with $M_1 = \lambda x.M_1'$

$$[\![M]\!]_z = (\nu w)(w(x).(\nu y')([x \leftrightarrow y'] \mid [\![M_1']\!]_w\{y'/u_x\}) \mid (\nu y)w\langle y\rangle.((!y(x).[\![M_2]\!]_x) \mid [w \leftrightarrow z]))$$

$[\![M]\!]_z \to^* \equiv (\nu y)([\![M_1']\!]_z \mid !y(x).[\![M_2]\!]_x) = Q$
$M \to M_1'\{M_2/x\} = N$
$N \ll Q$ immediate by def. 6.5

Case: $M = M_1[\tau]$ with $[\![M_1]\!]_x \to P$
Identical to $M = M_1 M_2$ with a reduction in $[\![M_1]\!]_x$.
Case: $M = M_1[\tau]$ with $M_1 = \Lambda X.M_1'$

$$[\![M]\!]_z = (\nu x)(x(X).[\![M_1']\!]_x \mid x\langle[\![\tau]\!]\rangle.[x \leftrightarrow z])$$

$[\![M]\!]_z \to^* [\![M_1']\!]_z\{[\![\tau]\!]/X\} = Q$
$M \to M_1\{\tau/X\} = N$
$Q \equiv [\![N]\!]_z$ by Lemma C.1, satisfying def. 6.5
□