

# Linear Logic Propositions as Session Types

Luis Caires<sup>1</sup>, Frank Pfenning<sup>2</sup> and Bernardo Toninho<sup>1,2</sup>

<sup>1</sup> *Faculdade de Ciências e Tecnologia and CITI, Universidade Nova de Lisboa, Lisboa, Portugal*

<sup>2</sup> *Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA*

*Received 18 January 2012*

Throughout the years, several typing disciplines for the  $\pi$ -calculus have been proposed. Arguably, the most widespread of these typing disciplines consists of session types. Session types describe the input/output behavior of processes and traditionally provide strong guarantees about this behavior (i.e., deadlock freedom and fidelity). While these systems exploit a fundamental notion of linearity, the precise connection between linear logic and session types has not been well understood.

This paper proposes a type system for the  $\pi$ -calculus that corresponds to a standard sequent calculus presentation of intuitionistic linear logic, interpreting linear propositions as session types and thus providing a purely logical account of all key features and properties of session types. We show the deep correspondence between linear logic and session types by exhibiting a tight operational correspondence between cut elimination steps and process reductions. We also discuss an alternative presentation of linear session types based on classical linear logic, and compare our development with other more traditional session type systems.<sup>†</sup>

## 1. Introduction

Linear logic has been intensively explored in the analysis of  $\pi$ -calculus models for communicating and mobile system, given its essential ability to deal with resources, effects, and non-interference. The fundamental way it provides for analyzing notions of sharing versus uniqueness, captured by the exponential “!”, seems to have been a source of inspiration for Milner when introducing replication in the  $\pi$ -calculus (Milner 1992). Following the early works of Abramsky (Abramsky 1993), several authors have exploited variants of  $\pi$ -calculi to express proof reductions (e.g., (Bellin and Scott 1994)) or game semantics (e.g., (Hyland and Ong 1995)) in systems of linear logic.

In the field of concurrency, many research directions have also drawn inspiration from linear logic for developing type-theoretic analyses of mobile processes, motivated by the works of Kobayashi, Pierce, and Turner (Kobayashi *et al.* 1996); a similar influence is already noticeable in the first publications by Honda on session types (Honda 1993). Many expressive type disciplines for  $\pi$ -calculi in which linearity frequently plays a key

<sup>†</sup> LC: Added this last sentence please check.

role have been proposed since then (e.g., (Kobayashi 1998; Honda *et al.* 1998; Yoshida *et al.* 2007; Giunti and Vasconcelos 2010)). However, linearity has been usually employed in such systems in indirect ways, exploiting the fine grained type context management techniques it provides, or the assignment of usage multiplicities to channels (Kobayashi *et al.* 1996), rather than the deeper type-theoretic significance of linear logical operators.

In this paper we present a type system for the  $\pi$ -calculus that exactly corresponds to the standard sequent calculus proof system for dual intuitionistic linear logic.

The key to our correspondence is a new, perhaps surprising, interpretation of intuitionistic linear logic formulas as a form of session types (Honda 1993; Honda *et al.* 1998), in which the programming language is a session-typed  $\pi$ -calculus, and the type structure consists precisely of the connectives of intuitionistic linear logic, retaining their standard proof-theoretic interpretation. In session-based concurrency, processes communicate through so-called session channels, connecting exactly two subsystems, and communication is disciplined by session protocols so that actions always occur in dual pairs: when one partner sends, the other receives; when one partner offers a selection, the other chooses; when a session terminates, no further interaction may occur. New sessions may be dynamically created by invocation of shared servers. Such a model exhibits concurrency in the sense that several sessions, not necessarily causally related, may be executing simultaneously, although races in unshared resources are forbidden; in fact this is the common situation in disciplined concurrent programming idioms. Mobility is also present, since both session and server names may be passed around (delegated) in communications. Session types have been introduced to discipline interactions in session-based concurrency, an important paradigm in communication-centric programming (Dezani-Ciancaglini and de' Liguoro 2010).

It turns out that the connectives of linear logic suffice to express all the essential features of finite session disciplines.

While in the linear  $\lambda$ -calculus types are assigned to terms (denoting functions and values), in our interpretation types are assigned to names (denoting communication channels) and describe their session protocol. The essence of our interpretation may already be found in the interpretation of the linear logic multiplicatives as behavioral prefix operators. Traditionally, an object of type  $A \multimap B$  denotes a linear function that given an object of type  $A$  returns an object of type  $B$  (Girard and Lafont 1987). In our interpretation, an object of type  $A \multimap B$  denotes a session  $x$  that first inputs a session channel of type  $A$ , and then behaves as  $B$ , where  $B$  specifies again an interactive behavior, rather than a closed value. Linearity of  $\multimap$  is essential, otherwise the behavior of the input session after communication could not be ensured. An object of type  $A \otimes B$  denotes a session that first sends a session channel of type  $A$  and afterwards behaves as  $B$ . But notice that objects of type  $A \otimes B$  really consist of two objects: the sent session of type  $A$  and the continuation session, of type  $B$ . These two sessions are separate and non-interfering, as enforced by the canonical semantics of the linear multiplicative conjunction ( $\otimes$ ). Our interpretation of  $A \otimes B$  appears asymmetric, in the sense that, of course, a channel of type  $A \otimes B$  is in general not typeable by  $B \otimes A$ . In fact, the symmetry captured by the proof of  $A \otimes B \vdash B \otimes A$  is realized by an appropriately typed process that coerces any session of type  $A \otimes B$  to a session of type  $B \otimes A$ . The other linear constructors are also

given compatible interpretations, in particular, the  $!A$  type is naturally interpreted as a type of a shared server for sessions of type  $A$ , and additive product and sum, to branch and choice session type operators. We thus obtain the first purely logical account of both shared and linear features of session types.

We briefly summarize the structure and contributions of the paper. After introducing our basic process model (Section 2), we describe a system of session types for the  $\pi$ -calculus that corresponds to the sequent calculus for dual intuitionistic linear logic DILL (Section 3). The correspondence is bidirectional and tight, in the sense that (a) any  $\pi$ -calculus computation can be simulated by proof reductions on typing derivations (Theorem 4.1), thus establishing a strong form of subject reduction (Theorem 4.4), and (b) that any proof reduction or conversion corresponds either to a computation step or to a process equivalence on the  $\pi$ -calculus side (Theorems 4.2 and 4.3). An intrinsic consequence of the logical typing is a global progress property, that ensures the absence of deadlock for systems with an arbitrary number of open sessions (Theorem 4.5). In (Section 5) we propose a version of our linear session type structure based on classical linear logic, and offer some preliminary comparison it with the intuitionistic formulation. We close the technical part of the paper with some discussion comparing our session types based on linear logic with other traditional type systems for session types (Section 6). Finally, in Section 7 we comment related work and present some closing remarks. <sup>‡</sup>

## 2. Process Model

We briefly introduce the syntax and operational semantics of the process model: the synchronous  $\pi$ -calculus (Sangiorgi and Walker 2001) extended with (binary) guarded choice.

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

$P ::=$	$\mathbf{0}$	(inaction)
	$P \mid Q$	(parallel composition)
	$(\nu y)P$	(name restriction)
	$x\langle y \rangle.P$	(output)
	$x(y).P$	(input)
	$!x(y).P$	(replicated / shared input)
	$x.\text{inl}; P$	(left choice)
	$x.\text{inr}; P$	(right choice)
	$x.\text{case}(P, Q)$	(case offer)

The operators  $\mathbf{0}$  (inaction),  $P \mid Q$  (parallel composition), and  $(\nu y)P$  (name restriction) comprise the static fragment of any  $\pi$ -calculus. We then have  $x\langle y \rangle.P$  (send  $y$  on  $x$  and proceeds as  $P$ ),  $x(y).P$  (receive a name  $z$  on  $x$  and proceed as  $P$  with the input parameter  $y$  replaced by  $z$ ), and  $!x(y).P$  which denotes replicated (or persistent) input. The

<sup>‡</sup> LC: Added this summary, since the paper is already a bit long.

$$\begin{array}{c}
\frac{P \xrightarrow{\alpha} Q}{(\nu y)P \xrightarrow{\alpha} (\nu y)Q} \text{(res)} \quad \frac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R} \text{(par)} \quad \frac{P \xrightarrow{\bar{\alpha}} P' \quad Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \text{(com)} \\
\\
\frac{P \xrightarrow{(\nu y)\bar{x}(y)} P' \quad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')} \text{(close)} \quad \frac{P \xrightarrow{\bar{x}(y)} Q}{(\nu y)P \xrightarrow{(\nu y)\bar{x}(y)} Q} \text{(open)} \quad x\langle y \rangle.P \xrightarrow{x(y)} P \text{(out)} \\
x(y).P \xrightarrow{x(z)} P\{z/y\} \text{(in)} \quad !x(y).P \xrightarrow{x(z)} P\{z/y\} \mid !x(y).P \text{(rep)} \quad x.\text{inl}; P \xrightarrow{x.\text{inl}} P \text{(lout)} \\
x.\text{inr}; P \xrightarrow{x.\text{inr}} P \text{(rout)} \quad x.\text{case}(P, Q) \xrightarrow{x.\text{inl}} P \text{(lin)} \quad x.\text{case}(P, Q) \xrightarrow{x.\text{inr}} Q \text{(rin)}
\end{array}$$

Fig. 1.  $\pi$ -calculus Labeled Transition System.

remaining three operators define a minimal labeled choice mechanism, comparable to the  $n$ -ary branching constructs found in standard session  $\pi$ -calculi (see eg., (Honda *et al.* 1998)). For the sake of minimality and without loss of generality we restrict our model to binary choice. In restriction  $(\nu y)P$  and input  $x(y).P$  the distinguished occurrence of the name  $y$  is binding, with scope the process  $P$ . For any process  $P$ , we denote the set of *free names* of  $P$  by  $fn(P)$ . A process is *closed* if it does not contain free occurrences of names. We identify process up to consistent renaming of bound names, writing  $\equiv_{\alpha}$  for this congruence. We write  $P\{x/y\}$  for the process obtained from  $P$  by capture avoiding substitution of  $x$  for  $y$  in  $P$ . Structural congruence expresses basic identities on the structure of processes, while reduction expresses the behavior of processes.

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

$$\begin{array}{lll}
P \mid \mathbf{0} \equiv P & \text{(S0)} & P \equiv_{\alpha} Q \Rightarrow P \equiv Q & \text{(S}\alpha\text{)} \\
P \mid Q \equiv Q \mid P & \text{(S|C)} & P \mid (Q \mid R) \equiv (P \mid Q) \mid R & \text{(S|A)} \\
(\nu x)\mathbf{0} \equiv \mathbf{0} & \text{(S}\nu\mathbf{0}\text{)} & x \notin fn(P) \Rightarrow P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) & \text{(S}\nu\mid\text{)} \\
(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P & \text{(S}\nu\nu\text{)} & & 
\end{array}$$

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

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

Notice that reduction is closed (by definition) under structural congruence. Reduction specifies the computations a process performs on its own. To characterize the interactions a process may perform with its environment, we introduce a labeled transition system; the standard early transition system for the  $\pi$ -calculus (Sangiorgi and Walker 2001) extended with appropriate labels and transition rules for the choice constructs. A transition  $P \xrightarrow{\alpha} Q$  denotes that process  $P$  may evolve to process  $Q$  by performing the action represented by

the label  $\alpha$ . Transition labels are given by

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

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

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

We recall some basic facts about reduction, structural congruence, and labeled transition, namely: closure of labeled transitions under structural congruence, and coincidence of  $\tau$ -labeled transition and reduction (Sangiorgi and Walker 2001). We write  $\rho_1\rho_2$  for relation composition (e.g.,  $\xrightarrow{\tau} \equiv$ ).

**Proposition 2.5.**

- 1 if  $P \equiv \xrightarrow{\alpha} Q$ , then  $P \xrightarrow{\alpha} \equiv Q$
- 2  $P \rightarrow Q$  if and only if  $P \xrightarrow{\tau} \equiv Q$ .

### 3. Intuitionistic Linear Logic as Session Types

In this section, we detail our main contribution of interpreting a sequent calculus formulation of dual intuitionistic linear logic as a session typing discipline for the process calculus of the previous section (we name this system  $\pi\text{DILL}$ ). We assume some familiarity with linear logic and sequent calculi, but nonetheless we will introduce each connective and its process interpretation incrementally, for the sake of presentation.

We consider a sequent calculus for intuitionistic linear logic in the style of DILL (Barber and Plotkin 1997), augmented with a faithful proof term assignment which allows us to refer to proofs as syntactic objects. A sequent is written as  $\Gamma; \Delta \vdash D : A$ , denoting that  $D$  is a proof of proposition  $A$ , under the linear assumptions in  $\Delta$  and the unrestricted (or exponential) assumptions in  $\Gamma$ . It turns out that the sequent calculus rules can be directly translated into session typing rules for processes in which the session behavior is described by the linear proposition  $A$ . We make this correspondence explicit through the judgment  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z : A$ , denoting that proof  $D$  maps to process  $P$ , which in turn provides a session behavior typed by  $A$  along channel  $z$ , provided it is composed with processes implementing the session behaviors specified by  $\Gamma$  and  $\Delta$  along the appropriate channels. Typing is defined modulo structural congruence, as often adopted in systems for process calculi. Furthermore, we tacitly assume that all channels declared in  $\Delta$ ,  $\Gamma$  and the channel  $z$  are distinct.

As will become clear in the following sections, sequent calculus right rules correspond to rules which detail how a process can implement the session behavior described by the considered connective. Dually, left rules explain how a process can make use of a session of a given type. Traditionally, session types are equipped with some notion of behavioral duality, in which the behavior of the inhabitants of a type is in some sense symmetric to the behavior of the inhabitants of its dual (e.g., the output session is dual to the input session, the choice session is dual to the branch session). In our setting a notion of behavioral duality also arises naturally from the additive and multiplicative nature of linear logic propositions.

Multiplicative conjunction  $\otimes$  and implication  $\multimap$  are dual in the sense that using a session of one type is equivalent to implementing a session of the other (the type  $A \otimes B$  types processes that output a session of type  $A$  and proceeds as specified by  $B$ , while the type  $A \multimap B$  types processes that input a session of type  $A$  and proceed as specified by  $B$ ). The same applies to additive conjunction and disjunction (which correspond to branching and choice, respectively). Namely, the type  $A \oplus B$  types processes that may choose either a left option of type  $A$  or a right option of type  $B$  respectively, while the type  $A \& B$  types processes that offer a choice between both a type  $A$  behavior and a type  $B$  behavior. Composition of the two dual endpoints of an open session is then logically represented by the cut rule, that matches a positive occurrence of a session type with a negative occurrence of the same type, hiding the communication channel.

Throughout the following section we illustrate our type system with a simple example, typically used to motivate session based process interactions (see, e.g., (Gay and Hole 2005)), involving a server that offers a buy and a quote operation and the respective client.

### 3.1. Linear Cut

In logic, the cut rule allows us to reason using lemmas. A proof of  $C$  (the theorem) is well-formed if it is obtained by the composition of a proof of  $C$  under the assumption of  $A$  (the lemma) and a proof of  $A$ . In linear logic, the cut rule is written as:

$$\frac{\Gamma; \Delta \vdash D : A \quad \Gamma; \Delta', x:A \vdash E : C}{\Gamma; \Delta, \Delta' \vdash \text{cut } D(x.E) : C} \text{ cut}$$

In essence, cut allows us to compose two proofs – one providing  $A$  and the other one using  $A$  to provide  $C$ . This principle of composition is captured in the process interpretation as follows:

$$\frac{\Gamma; \Delta \vdash D \rightsquigarrow P :: x : A \quad \Gamma; \Delta', x:A \vdash E \rightsquigarrow Q :: z : C}{\Gamma; \Delta, \Delta' \vdash \text{cut } D(x.E) \rightsquigarrow (\nu x)(P \mid Q) :: z : C} \text{ Tcut}$$

The process  $P$  implements session  $A$  along channel  $x$ , while process  $Q$  implements session  $C$  along channel  $z$ , under the assumption that a session of type  $A$  is available on  $x$ . Furthermore, since we follow a linear typing discipline,  $Q$  requires *all* the behavior supplied by  $P$  along  $x$  and therefore composing the two processes must necessarily restrict the scope of  $x$  to the two processes.

This identification of cut with typed composition is not arbitrary, and turns out to be

much deeper than it might first seem. The point of composing two processes is for them to interact with each other. Generally, both  $P$  and  $Q$  may interact with the “external” process environment (captured by  $\Delta$  and  $\Delta'$ , respectively), but the interesting interactions are those in which both  $P$  and  $Q$  communicate with each other and evolve together to some residual processes  $P'$  and  $Q'$ . All these interactions (both with the environment and between the two processes) can be understood proof-theoretically through the process of cut elimination in a proof (called interchangeably cut reduction). Throughout this development, we take the correspondence of principal cut reductions (when a right rule is cut with the corresponding left rule) and process reductions as the guiding principle in our design, in the same manner as the correspondence between proof reductions and  $\lambda$ -term reductions guide the Curry-Howard isomorphism. The interpretation of the logical cut as composition-plus-hiding over process behaviors was introduced by (Abramsky 1993), even if in a simpler setting of CSP style trace models.

### 3.2. Linear Implication

Implication in linear logic, written  $A \multimap B$  is commonly understood as a proof transformation process: provide me with exactly one proof of  $A$  and I shall make full use of it to produce exactly one proof of  $B$ . Dually, using  $A \multimap B$  requires us to exhibit a proof of  $A$ , which then warrants the use of a proof of  $B$ . The rules for implication are:

$$\frac{\Gamma; \Delta, x : A \vdash D : B}{\Gamma; \Delta \vdash \multimap R (x.D) : A \multimap B} \multimap R \quad \frac{\Gamma; \Delta_1 \vdash E_1 : A \quad \Gamma; \Delta_2, x : B \vdash E_2 : C}{\Gamma; \Delta_1, \Delta_2, x : A \multimap B \vdash \multimap L x E_1 (x.E_2) : C} \multimap L$$

We can also interpret the implication  $A \multimap B$  as an object that *inputs*  $A$  and then produces a  $B$ . Using such an object therefore requires an output of  $A$  which then allows for the use of  $B$ . Thus, the process interpretation for implication is as input on the right and output on the left, as follows:

$$\frac{\Gamma; \Delta, x : A \vdash D \rightsquigarrow P :: z : B}{\Gamma; \Delta \vdash \multimap R (x.D) \rightsquigarrow z(x).P :: z : A \multimap B} T\multimap R$$

$$\frac{\Gamma; \Delta_1 \vdash E_1 \rightsquigarrow P :: y : A \quad \Gamma; \Delta_2, x : B \vdash E_2 \rightsquigarrow Q :: z : C}{\Gamma; \Delta_1, \Delta_2, x : A \multimap B \vdash \multimap L x E_1 (x.E_2) \rightsquigarrow (\nu y)x\langle y \rangle.(P \mid Q) :: z : C} T\multimap L$$

Note how in the left rule, we output a fresh name  $y$ , on which the process  $P$  implements the session behavior  $A$ . The fresh output, combined with the context splitting ensures that  $Q$  does not interfere with  $P$  in any way. Throughout our development we will restrict ourselves to outputs of fresh names, in the style of (Boreale 1998).

As mentioned in the previous section, we can validate our correspondence by considering the principal cut elimination steps for linear implication, which are given below in proof term form ( $\Rightarrow$  stands for cut reduction and  $\hat{D}^x$  stands for the process that corresponds to proof term  $D$ , which depends on name  $x$ ):

$$\begin{aligned} \text{cut} (\multimap R (y.D_y)) (x.\multimap L x E_1 (x.E_{2x})) &\rightsquigarrow (\nu x)((x(y).\hat{D}^x) \mid (\nu y)x\langle y \rangle.(\hat{E}_1^y \mid \hat{E}_2^z)) \\ \Rightarrow &\quad \quad \quad \rightarrow \\ \text{cut} (\text{cut } E_1 (y.D_y)) (x.E_{2x}) &\rightsquigarrow (\nu x)(\nu y)(\hat{D}^x \mid \hat{E}_1^y \mid \hat{E}_2^z) \end{aligned}$$

### 3.3. Multiplicative Unit and Conjunction

The multiplicative unit of intuitionistic linear logic, written  $\mathbf{1}$ , is a proposition that is shown using no resources. Dually, we use  $\mathbf{1}$  by just silently consuming it. The sequent calculus rules for this connective are:

$$\frac{}{\Gamma; \cdot \vdash \mathbf{1R} : \mathbf{1}} \mathbf{1R} \quad \frac{\Gamma; \Delta \vdash D : C}{\Gamma; \Delta, x : \mathbf{1} \vdash \mathbf{1L} x D : C} \mathbf{1L}$$

The process interpretation for the multiplicative unit is the terminated session, or the inactive process:

$$\frac{}{\Gamma; \cdot \vdash \mathbf{1R} \rightsquigarrow \mathbf{0} :: z : \mathbf{1}} T\mathbf{1R} \quad \frac{\Gamma; \Delta \vdash D \rightsquigarrow P : C}{\Gamma; \Delta, x : \mathbf{1} \vdash \mathbf{1L} x D \rightsquigarrow P : C} T\mathbf{1L}$$

The intuition is that we provide a session of type  $\mathbf{1}$  with the terminated process (no further ambient resources can be used), and “use it” by simply erasing. This is one of the two cases where no process reduction actually takes place in composition, since the inactive process and the scope restriction are erased through *structural congruence*, not through reduction.

Multiplicative conjunction, written  $A \otimes B$ , requires us to split our resources in order to provide both an  $A$  and a  $B$ . Using such a proposition simply adds  $A$  and  $B$  to the context:

$$\frac{\Gamma; \Delta \vdash D_1 : A \quad \Gamma; \Delta' \vdash D_2 : B}{\Gamma; \Delta, \Delta' \vdash \otimes R D_1 D_2 : A \otimes B} \otimes R \quad \frac{\Gamma; \Delta, y : A, x : B \vdash E : C}{\Gamma; \Delta, x : A \otimes B \vdash \otimes L x (y.x.E) : C} \otimes L$$

The process interpretation for  $\otimes$  is the exact behavior dual of  $\multimap$ . While the right rule for  $\multimap$  corresponds to input and the left rule corresponds to output, the right and left rules for  $\otimes$  correspond to output and input, respectively:

$$\frac{\Gamma; \Delta \vdash D_1 \rightsquigarrow P :: y : A \quad \Gamma; \Delta' \vdash D_2 \rightsquigarrow Q :: z : B}{\Gamma; \Delta, \Delta' \vdash \otimes R D_1 D_2 \rightsquigarrow (\nu y)z \langle y \rangle . (P \mid Q) :: z : A \otimes B} T\otimes R$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash E \rightsquigarrow P :: z : C}{\Gamma; \Delta, x : A \otimes B \vdash \otimes L x (y.x.E) \rightsquigarrow x(y).P :: z : C} T\otimes L$$

Notice how in the right rule for  $\otimes$  we create a fresh name  $y$ , along which the session of type  $A$  is offered by process  $P$ , while  $B$  is offered along the residual channel  $z$ , by process  $Q$ . The left rule simply inputs along  $x$ , binding the input name to  $y$  (which offers session  $A$ ) in the continuation  $P$ , where  $x$  now offers  $B$ . The proof reductions that validate this interpretation (as well as the corresponding process reductions) are given below:

$$\begin{aligned} \text{cut} (\otimes R D_1 D_2) (x. \otimes L x (y.x. E_{xy})) &\rightsquigarrow (\nu x)((\nu y)x \langle y \rangle . (\hat{D}_1^y \mid \hat{D}_2^z)) \mid x(y). \hat{E}^z \\ \Rightarrow &\rightarrow \\ \text{cut} D_1 (y. \text{cut} D_2 (x. E_{xy})) &\rightsquigarrow (\nu x)(\nu y)(\hat{D}_1^y \mid \hat{D}_2^z \mid \hat{E}^z) \end{aligned}$$

**3.3.1. Example** We now consider a simple example that illustrates the connectives introduced thus far. We model a client process that wishes to perform a “buy” operation on a remote server. The client does so by sending to the server a product name and a

credit card number, after which it receives back a receipt. From the client perspective, the session protocol exposed by the server can be specified by the following type:

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

We assume that  $N$  and  $I$  are types representing shareable values such as strings and integers. To simplify, we set  $N = I = \mathbf{1}$  (an extension of this system with basic and dependent data types is given in (Toninho *et al.* 2011)). Assuming  $s$  to be the name of the session channel along which the client and the server interact, the following process implements the client:

$$\text{BCIntBody}_s \triangleq (\nu tea)s\langle tea \rangle.(\nu cc)s\langle cc \rangle.s(r).\mathbf{0}$$

The process above specifies a client that buys tea from the server (we abstract away what the client does with the receipt from the server). First it sends the identification of the product to the server, then its credit card information and finally receives the appropriate receipt. We then have that the following is derivable:

$$;s : \text{ServerProto} \vdash \text{BCIntBody}_s :: - : \mathbf{1}$$

We write  $-$  for an anonymous variable that does not appear in the typed process. This is possible since the inactive process  $\mathbf{0}$  is typed by  $x : \mathbf{1}$  and does not make use of  $x$ . The server code is as follows:

$$\text{SrvBody}_s \triangleq s(pn).s(cn)(\nu rc)s\langle rc \rangle.\mathbf{0}$$

It is straightforward to see that  $; \cdot \vdash \text{SrvBody}_s :: s : \text{ServerProto}$  is derivable. By composing the two processes with a cut, we obtain the following:

$$; \cdot \vdash (\nu s)(\text{SrvBody}_s \mid \text{BCIntBody}_s) :: - : \mathbf{1}$$

In this simple example we have only introduced processes that interact along a single session. However, our system accommodates the full generality of binary session types (e.g., a process interacting along multiple sessions is  $x : A \multimap \mathbf{1}, y : A \otimes \mathbf{1} \vdash y(w).(\nu k)x\langle k \rangle.\mathbf{0} :: - : \mathbf{1}$ ).

### 3.4. Additive Conjunction and Disjunction

We now consider additive conjunction  $A \& B$  and disjunction  $A \oplus B$ . Additive conjunction represents alternative availability of resources (i.e., we can provide  $A$  and  $B$ , but only one of them):

$$\frac{\Gamma; \Delta \vdash D_1 : A \quad \Gamma; \Delta \vdash D_2 : B}{\Gamma; \Delta \vdash \&\text{R } D_1 \ D_2 : A \& B} \&\text{R} \quad \frac{\Gamma; \Delta, x : A \vdash E : C}{\Gamma; \Delta, x : A \& B \vdash \&\text{L}_1 x (x.E) : C} \&\text{L}_1$$

$$\frac{\Gamma; \Delta, x : B \vdash E : C}{\Gamma; \Delta, x : A \& B \vdash \&\text{L}_2 x (x.E) : C} \&\text{L}_2$$

The process interpretation of  $\&$  is a form of (binary) branching. A process implementing a session of type  $A \& B$  offers the alternative between a session of type  $A$  and one of type

B. Using such a session entails selecting the appropriate alternative (either  $\text{inl}$  or  $\text{inr}$ ):

$$\frac{\Gamma; D \vdash D_1 \rightsquigarrow P :: z : A \quad \Gamma; \Delta \vdash D_2 \rightsquigarrow Q :: z : B}{\Gamma; \Delta \vdash \&\mathbf{R} D_1 D_2 \rightsquigarrow z.\text{case}(P_1, P_2) :: z : A \& B} T\&\mathbf{R}$$

$$\frac{\Gamma; \Delta, x : A \vdash E \rightsquigarrow Q :: z : C}{\Gamma; \Delta, x : A \& B \vdash \&\mathbf{L}_1 x (x.E) \rightsquigarrow x.\text{inl}; Q :: z : C} T\&\mathbf{L}_1$$

$$\frac{\Gamma; \Delta, x : B \vdash E \rightsquigarrow Q :: z : C}{\Gamma; \Delta, x : A \& B \vdash \&\mathbf{L}_2 x (x.E) \rightsquigarrow x.\text{inr}; Q :: z : C} T\&\mathbf{L}_2$$

The proof and process reductions that validate this interpretation are as follows (for the sake of presentation, we omit the proof reduction for the second left rules since they are identical):

$$\begin{array}{ccc} \text{cut } (\&\mathbf{R} D_1 D_2) (x.\&\mathbf{L}_1 x (x.E_x)) & \rightsquigarrow & (\nu x)(x.\text{case}(\hat{D}_1^x, \hat{D}_2^x) \mid x.\text{inl}; \hat{E}^z) \\ \Rightarrow & & \rightarrow \\ \text{cut } D_1 (x.E_x) & \rightsquigarrow & (\nu x)(\hat{D}_1^x \mid \hat{E}^z) \end{array}$$

Additive disjunction is the dual of additive conjunction. While additive conjunction represents alternative availability of resources (i.e., both resources are available to the client, and he chooses which to use), the additive disjunction  $A \oplus B$  represents alternative availability of resources in which the choice is made by the one supplying the resources, that is, the client does not know *a priori* if its  $A$  or  $B$  that is actually available (and hence needs to branch on the two possibilities):

$$\frac{\Gamma; \Delta \vdash D : A}{\Gamma; \Delta \vdash \oplus\mathbf{R}_1 D : A \oplus B} \oplus\mathbf{R}_1 \quad \frac{\Gamma; \Delta \vdash D : B}{\Gamma; \Delta \vdash \oplus\mathbf{R}_2 D : A \oplus B} \oplus\mathbf{R}_2$$

$$\frac{\Gamma; \Delta, x : A \vdash E_1 : C \quad \Gamma; \Delta, x : B \vdash E_2 : C}{\Gamma; \Delta, x : A \oplus B \vdash \oplus\mathbf{L} x (x.E_1) (x.E_2) : C} \oplus\mathbf{L}$$

The process interpretation captures the duality mentioned above in a precise sense. The right rules for  $\oplus$  correspond to the left rules for additive conjunction (either a choice of  $\text{inl}$  or  $\text{inr}$ , respectively), while the left rule for  $\oplus$  corresponds to the right rule (a case analysis):

$$\frac{\Gamma; \Delta \vdash D \rightsquigarrow P :: z : A}{\Gamma; \Delta \vdash \oplus\mathbf{R}_1 D \rightsquigarrow z.\text{inr}; P :: z : A \oplus B} T\oplus\mathbf{R}_1$$

$$\frac{\Gamma; \Delta \vdash D \rightsquigarrow P :: z : B}{\Gamma; \Delta \vdash \oplus\mathbf{R}_2 D \rightsquigarrow z.\text{inr}; P :: z : A \oplus B} T\oplus\mathbf{R}_2$$

$$\frac{\Gamma; \Delta, x : A \vdash E_1 \rightsquigarrow Q_1 :: z : C \quad \Gamma; \Delta, x : B \vdash E_2 \rightsquigarrow Q_2 :: z : C}{\Gamma; \Delta, x : A \oplus B \vdash \oplus\mathbf{L} x (x.E_1) (x.E_2) \rightsquigarrow x.\text{case}(Q_1, Q_2) :: z : C} T\oplus\mathbf{L}$$

Similarly, we obtain the following proof and process reductions (we show only the reductions for the first right rule, the reductions for the remaining one are identical):

$$\begin{array}{ccc} \text{cut } (\oplus\mathbf{R}_1 D) (x.\oplus\mathbf{L} x (x.E_{1x}) (x.E_{2x})) & \rightsquigarrow & (\nu x)(x.\text{inl}; \hat{D}^x \mid x.\text{case}(\hat{E}_1^z, \hat{E}_2^z)) \\ \Rightarrow & & \rightarrow \\ \text{cut } D (x.E_{1x}) & \rightsquigarrow & (\nu x)(\hat{D}^x \mid \hat{E}_1^z) \end{array}$$

3.4.1. *Extending the Example* We can now easily extend our earlier example of the client and server to include branching. Consider the following type for the server interface:

$$\text{ServerProto2} \triangleq (N \multimap I \multimap (N \otimes \mathbf{1})) \& (N \multimap (I \otimes \mathbf{1}))$$

The type above models a server that offers the “buy” operation from before, but also a “quote” operation in which the client sends a product name and the server replies with the price for the respective product. The ability to offer multiple services is modelled with the additive conjunction. The client code from before can be easily extended by first choosing the appropriate operation:

$$\text{BCIntBody2}_s \triangleq s.\text{in!}; (\nu\text{tea})s\langle\text{tea}\rangle.(\nu\text{cc})s\langle\text{cc}\rangle.s(r).\mathbf{0}$$

The server code is now extended with the appropriate communication steps:

$$\text{SrvBody2}_s \triangleq s.\text{case}(s\langle\text{pn}\rangle.s\langle\text{cn}\rangle(\nu\text{rc})s\langle\text{rc}\rangle.\mathbf{0}, s\langle\text{pn}\rangle.(\nu\text{pr})s\langle\text{pr}\rangle.\mathbf{0})$$

It is straightforward to see that both the server and client processes have the appropriate types, and we thus obtain the following composed system through an instance of cut:

$$\frac{;\cdot \vdash \text{SrvBody2}_s :: s : \text{ServerProto2} \quad ;s : \text{ServerProto2} \vdash \text{BCIntBody2}_s :: - : \mathbf{1}}{;\cdot \vdash (\nu s)(\text{SrvBody2}_s \mid \text{BCIntBody2}_s) :: - : \mathbf{1}}$$

### 3.5. Exponential

The linear logic exponential  $!A$  enables a form of controlled weakening and contraction. A proposition  $!A$  provides an arbitrary number of copies of the resource  $A$ , including 0. To prove  $!A$ , the usage of linear resources is therefore disallowed (otherwise this would limit the number of times  $A$  can be used):

$$\frac{\Gamma; \cdot \vdash D : A}{\Gamma; \cdot \vdash !R D : !A} !R$$

Using a proof of  $!A$  simply moves the proposition  $A$  to the unrestricted context, accurately capturing the fact that it can be used arbitrarily often. However, to actually be able to make use of  $A$ , one must explicitly *copy* it to the linear context (this rule is hence called *copy*):

$$\frac{\Gamma, u : A; \Delta \vdash D : C}{\Gamma; \Delta, x : !A \vdash !L x (u. D) : C} !L \quad \frac{\Gamma, u : A; \Delta, y : A \vdash D : C}{\Gamma, u : A; \Delta \vdash \text{copy } u (y. D) : C} \text{copy}$$

The process interpretation of the exponential is a slightly more subtle, albeit not surprising. Given that  $!A$  denotes the ability to offer  $A$  arbitrarily often, a process that implements this behavior consists of a replicated process implementing a session of type  $A$ :

$$\frac{\Gamma; \cdot \vdash D \rightsquigarrow P :: y : A}{\Gamma; \cdot \vdash !R D \rightsquigarrow !z(y).P :: z : !A} T!R$$

We consider only a guarded form of replicated inputs. The process in the above rule waits for an input along  $z$ , after which it will spawn a copy of  $P$  that will offer the appropriate session along the channel that was initially input. The left process rule for  $!$  is silent,

since it does not actually make use of the replicated process. The actual action takes place in the copy rule, where the replicated input is matched by a corresponding (fresh) output:

$$\frac{\Gamma, u : A; \Delta \vdash E \rightsquigarrow Q :: z : C}{\Gamma; \Delta \vdash !L x E \rightsquigarrow Q\{x/u\} :: z : C} T!L$$

$$\frac{\Gamma, u : A; \Delta, y : A \vdash D \rightsquigarrow P :: z : C}{\Gamma, u : A; \Delta \vdash \text{copy } u (y. D) \rightsquigarrow (\nu y)u\langle y \rangle.P :: z : C} T\text{copy}$$

The use of unrestricted hypotheses requires an additional cut principle, in which the cut formula is added to the unrestricted context (and the linear context is empty in the first premise):

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

Just as linear cut corresponds to composition of linear resources, the exponential cut allows for composition of unrestricted resources:

$$\frac{\Gamma; \cdot \vdash D \rightsquigarrow P :: y : A \quad \Gamma, u : A; \Delta \vdash E \rightsquigarrow Q :: z : C}{\Gamma; \Delta \vdash \text{cut}^! D (u. E) \rightsquigarrow (\nu u)(!u(y).P \mid Q) :: z : C} \text{cut}^!$$

Similar to the case for **1**, the principal (linear) cut for **!** does not map to a process reduction. Proof-theoretically, such a cut reduces to an instance of  $\text{cut}^!$ , for which the following reduction is obtained, whenever an instance of  $\text{copy}$  is reached:

$$\begin{aligned} \text{cut}^! D (u. \text{copy } u (y. E_{uy})) &\rightsquigarrow (\nu u)((!u(y). \hat{D}^y) \mid (\nu y) u\langle y \rangle. \hat{E}^z) \\ \Rightarrow &\quad \rightarrow \\ \text{cut } D (y. \text{cut}^! D (u. E_{uy})) &\rightsquigarrow (\nu y)(\hat{D}^y \mid (\nu u)((!u(y). \hat{D}^y) \mid \hat{E}^z)) \end{aligned}$$

3.5.1. *Replicating the Example* We now elaborate on our running example, in order to illustrate sharing and session initiation. Consider now a different client, that picks the “quote” rather than the “buy” operation, and the corresponding composed system.

$$\begin{aligned} \text{QCIntBody}_s &\triangleq s.\text{inr}; (\nu \text{cof})s\langle \text{cof} \rangle.s\langle \text{pr} \rangle.\mathbf{0} \\ \text{QSimple} &\triangleq (\nu s)(\text{SrvBody2}_s \mid \text{QCIntBody}_s) \end{aligned}$$

We have the typings  $\cdot; s:\text{ServerProto2} \vdash \text{QCIntBody}_s :: -:\mathbf{1}$  and  $\cdot; \cdot \vdash \text{QSimple} :: -:\mathbf{1}$ .

In these examples, there is a single installed pair client-server, where the session is already initiated, and only known to the two partners. To illustrate sharing, we now consider a replicated server. Such a replicated server is able to spawn a fresh session instance for each initial invocation, each one conforming to the general behavior specified by  $\text{ServerProto2}$ , and can be typed by  $!\text{ServerProto2}$ . Correspondingly, clients must initially invoke the replicated server to instantiate a new session (cf. the  $T\text{copy}$  rule).

$$\begin{aligned} \text{QClient} &\triangleq (\nu s)c\langle s \rangle.\text{QCIntBody}_s \\ \text{BClient} &\triangleq (\nu s)c\langle s \rangle.\text{BCIntBody2}_s \\ \text{Server} &\triangleq !c\langle s \rangle.\text{SrvBody2}_s \\ \text{SharSys} &\triangleq (\nu c)(\text{Server} \mid \text{BClient} \mid \text{QClient}) \end{aligned}$$

For the shared server, by  $T!R$ , we type  $\cdot; \cdot \vdash \text{Server} :: c:!\text{ServerProto2}$ . We also have, for

the clients, by  $T_{\text{copy}}$  the typings  $c:\text{ServerProto2} ; \cdot \vdash \text{BClient} :: -:\mathbf{1}$  and  $c:\text{ServerProto2} ; \cdot \vdash \text{QClient} :: -:\mathbf{1}$ . By  $T!L$ , and  $T_{\text{cut}}$  we obtain the intended typing for the whole system:  $;\cdot \vdash \text{SharSys} :: -:\mathbf{1}$ . Notice how the session instantiation protocol is naturally explained by the logical interpretation of the  $!$  operator.

### 3.6. Identity

In proof theory, the identity theorem for a sequent calculus presentation of logic entails the internal completeness of logic: one can always prove an assumed proposition. Just as the computational content of cut elimination corresponds to process reduction, the computational content of the identity theorem yields a form of expansion.

**Proposition 3.1.** For any type  $A$  and distinct names  $x, y$ , there is a process  $id_A(x, y)$  and a cut-free derivation  $D$  such that  $;\cdot x : A \vdash D \rightsquigarrow id_A(x, y) :: y : A$ .

The identity process  $id_A(x, y)$ , containing exactly the free names  $x, y$ , implements a synchronous mediator that carries out the communication protocol specified by the type  $A$  between the two channels  $x$  and  $y$ . To clarify, we analyze the interpretation of the sequent  $A \otimes B \vdash B \otimes A$ :

$$x : A \otimes B \vdash F \rightsquigarrow x(z).(\nu n)y\langle n \rangle.(id_B(x, n) \mid id_A(z, y)) :: y : B \otimes A$$

where  $F = \otimes L x (z.x. \otimes R D E)$ ,  $D \rightsquigarrow id_B(x, n)$  and  $E \rightsquigarrow id_A(z, y)$ . The process given above coerces a session of type  $A \otimes B$  on channel  $x$  to one of type  $B \otimes A$  on  $y$  by first inputting a session of type  $A$  (bound to  $z$ ) and afterwards sending on  $y$  a session of type  $B$  (carried out by coercing the continuation of  $x$  to  $n$ ), after which it progresses with a session of type  $A$  along  $y$  (by coercing the continuation of  $z$  to  $y$ ).

### 3.7. Summary

In this section, we summarize the contributions of this section. Specifically, we present here the complete type system  $\pi\text{DILL}$  that arises from our interpretation of intuitionistic linear logic, which for the sake of presentation we developed incrementally throughout this section. The rules are exactly those presented before, obtained by erasing the sequent calculus proof terms, and are given in Fig. 2 (where  $T$  stands for  $z : C$ ).

The extraction  $\rightsquigarrow$  of well-typed processes from sequent calculus proof terms is summarized in Fig. 3. Extraction is unique up to structural congruence, since typing is by definition closed under  $\equiv$ .

Finally, we summarize the several proof conversions and their correspondent process reductions or equivalences. As detailed throughout the previous sections, process reductions correspond to computational proof conversions (Fig. 4). The structural conversions in Fig. 5 correspond to structural equivalences in the  $\pi$ -calculus, since they just change the order of cuts, e.g.,  $(\text{cut}/-/\text{cut}_1)$  translates to

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

In addition, we have two special conversions. Among those,  $(\text{cut}/\mathbf{1R}/\mathbf{1L})$  is not needed

$$\begin{array}{c}
\frac{\Gamma; \Delta \vdash P :: T}{\Gamma; \Delta, x:\mathbf{1} \vdash P :: T} \text{ (T1L)} \quad \frac{}{\Gamma; \cdot \vdash \mathbf{0} :: x:\mathbf{1}} \text{ (T1R)} \\
\frac{\Gamma; \Delta, y:A, x:B \vdash P :: T}{\Gamma; \Delta, x:A \otimes B \vdash x(y).P :: T} \text{ (T}\otimes\text{L)} \quad \frac{\Gamma; \Delta \vdash P :: y:A \quad \Gamma; \Delta' \vdash Q :: x:B}{\Gamma; \Delta, \Delta' \vdash (\nu y)x(y).(P \mid Q) :: x:A \otimes B} \text{ (T}\otimes\text{R)} \\
\frac{\Gamma; \Delta \vdash P :: y:A \quad \Gamma; \Delta', x:B \vdash Q :: T}{\Gamma; \Delta, \Delta', x:A \multimap B \vdash (\nu y)x(y).(P \mid Q) :: T} \text{ (T}\multimap\text{L)} \quad \frac{\Gamma; \Delta, y:A \vdash P :: x:B}{\Gamma; \Delta \vdash x(y).P :: x:A \multimap B} \text{ (T}\multimap\text{R)} \\
\frac{\Gamma; \Delta \vdash P :: x:A \quad \Gamma; \Delta', x:A \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: T} \text{ (Tcut)} \quad \frac{\Gamma; \cdot \vdash P :: y:A \quad \Gamma, u:A; \Delta \vdash Q :: T}{\Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: T} \text{ (Tcut!)} \\
\frac{\Gamma, u:A; \Delta, y:A \vdash P :: T}{\Gamma, u:A; \Delta \vdash (\nu y)u(y).P :: T} \text{ (Tcopy)} \\
\frac{\Gamma, u:A; \Delta \vdash P :: T}{\Gamma; \Delta, x:!A \vdash P\{x/u\} :: T} \text{ (T!L)} \quad \frac{\Gamma; \cdot \vdash Q :: y:A}{\Gamma; \cdot \vdash !x(y).Q :: x:!A} \text{ (T!R)} \\
\frac{\Gamma; \Delta, x:A \vdash P :: T \quad \Gamma; \Delta, x:B \vdash Q :: T}{\Gamma; \Delta, x:A \oplus B \vdash x.\text{case}(P, Q) :: T} \text{ (T}\oplus\text{L)} \quad \frac{\Gamma; \Delta, x:B \vdash P :: T}{\Gamma; \Delta, x:A \& B \vdash x.\text{inr}; P :: T} \text{ (T}\&\text{L}_2\text{)} \\
\frac{\Gamma; \Delta \vdash P :: x:A \quad \Gamma; \Delta \vdash Q :: x:B}{\Gamma; \Delta \vdash x.\text{case}(P, Q) :: x:A \& B} \text{ (T}\&\text{R)} \quad \frac{\Gamma; \Delta, x:A \vdash P :: T}{\Gamma; \Delta, x:A \& B \vdash x.\text{inl}; P :: T} \text{ (T}\&\text{L}_1\text{)} \\
\frac{\Gamma; \Delta \vdash P :: x:A}{\Gamma; \Delta \vdash x.\text{inl}; P :: x:A \oplus B} \text{ (T}\oplus\text{R}_1\text{)} \quad \frac{\Gamma; \Delta \vdash P :: x:B}{\Gamma; \Delta \vdash x.\text{inr}; P :: x:A \oplus B} \text{ (T}\oplus\text{R}_2\text{)}
\end{array}$$

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

in order to simulate the  $\pi$ -calculus reduction, while (cut/!R/!L) is. In cut-elimination procedures, these are always used from left to right. Here, they are listed as equivalences because the corresponding  $\pi$ -calculus terms are structurally congruent. The root cause for this is that the rules **1L** and **!L** are *silent*: the extracted terms in the premise and conclusion are the same, modulo renaming. The structural conversions in Fig. 7 push cut<sup>!</sup> into the derivation. From a proof theoretic perspective, since cut<sup>!</sup> cuts a persistent variable  $u$ , cut<sup>!</sup> may be duplicated or erased. On the  $\pi$ -calculus side, these no longer correspond to structural congruences, but, quite remarkably, to behavioral equivalences, derivable from known properties of typed processes, the (sharpened) Replication Theorems (Sangiorgi and Walker 2001). These hold in our system, due to our interpretation of **!** types. Our operational correspondence results also depend on six commuting conversions, four in Fig. 6 plus two symmetric versions. The commuting conversions push a cut up (or inside) the **1L** and **!L** rules. During the usual cut elimination procedures, these are used from left to right. In the correspondence with the sequent calculus, the situation is more complex. Because the **1L** and **!L** rules do not affect the extracted term, cuts have to be permuted with these two rules in order to simulate  $\pi$ -calculus reduction. From the process calculus perspective, such conversions correspond to identity. There is a second group of commuting conversions (not shown), not necessary for our current development. Those do not correspond to structural congruence nor to strong bisimilarities on  $\pi$ -calculus, as they may not preserve process behavior in the general untyped setting, since they

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

Fig. 3. Proof  $D$  extracts to process  $\hat{D}^z$ .

promote an action prefix from a subexpression to the top level. Such equations denote behavioral identities under a natural definition of typed observational congruence for our calculus (Pérez et al.(2012)).

**Definition 3.2 (Relations on derivations induced by conversions).** (1)  $\equiv$ : the least congruence on derivations generated by the structural conversions (I) and the commuting conversions (II); (2)  $\simeq_s$ : the least congruence on derivations generated by all structural conversions (I-III). We extend  $\simeq_s$  to processes as the congruence generated by the process equations on the right. (3)  $\Rightarrow$ : the reduction on derivations obtained by orienting all conversions in the direction shown, from left to right or top to bottom.

#### 4. Computational Correspondence, Progress and Preservation

We now present the results stating the key properties of our type system and logical interpretation. Theorem 4.1 states the existence of a simulation between reductions in the typed  $\pi$ -calculus and proof conversions / reductions, expressing a strong form of subject reduction for our type system. The proof relies on several auxiliary lemmas that relate process reduction with proof reduction at a particular type. The lemmas themselves are all very similar, so we only present the lemmas for  $\otimes$  and  $!$ . The remaining lemmas, and their proofs, are detailed in Appendix A.

**Lemma 4.1.** Assume

- 1  $\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow P_1 :: x:C_1 \otimes C_2$  with  $P_1 \xrightarrow{(\nu y)x\langle y \rangle} P'_1$ ;
- 2  $\Gamma; \Delta_2, x:C_1 \otimes C_2 \vdash D_2 \rightsquigarrow Q_1 :: z:C$  with  $Q_1 \xrightarrow{x(y)} Q'_1$ .

Then

- 1  $\mathbf{cut} D_1(x.D_2) \equiv \equiv D$  for some  $D$ ;
- 2  $\Gamma; \Delta_1, \Delta_2 \vdash D \rightsquigarrow Q_2 :: z : C$  for some  $Q_2 \equiv (\nu x)(P'_1 | Q'_1)$ .

*Proof.* See Appendix A.6.

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

Fig. 4. Computational Conversions

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

Fig. 5. Structural Conversions (I): Cut Conversions

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

Fig. 6. Structural Conversions (II): Commuting Conversions

$$\begin{array}{lcl}
\text{cut}^! D (u. \text{cut } E_u (y. F_{uy})) & \rightsquigarrow & (\nu u)(!u \langle y \rangle. \hat{D}^y \mid (\nu y)(\hat{E}^y \mid \hat{F}^z)) \\
\cong & \cong & \\
\text{cut } (\text{cut}^! D (u. E_u)) (y. \text{cut}^! D (u. F_{uy})) & \rightsquigarrow & (\nu y)((\nu u)(!u \langle y \rangle. \hat{D}^y \mid \hat{E}^y) \mid \\
& & (\nu u)(!u \langle y \rangle. \hat{D}^y \mid \hat{F}^z)) \\
\text{cut}^! D (u. \text{cut}^! E_u (v. F_{uv})) & \rightsquigarrow & (\nu u)(!u \langle y \rangle. \hat{D}^y \mid (\nu v)(!v \langle y \rangle. \hat{E}^y \mid \hat{F}^z)) \\
\cong & \cong & \\
\text{cut}^! (\text{cut}^! D (u. E_u)) (v. \text{cut}^! D (u. F_{uv})) & \rightsquigarrow & (\nu v)((!v \langle y \rangle. (\nu u)(!u \langle y \rangle. \hat{D}^y \mid \hat{E}^y)) \mid \\
& & (\nu u)(!u \langle y \rangle. \hat{D}^y \mid \hat{F}^z)) \\
\text{cut}^! (\text{cut}^! D (u. E_u)) (v. F_v) & \rightsquigarrow & (\nu v)(!v \langle y \rangle. (\nu u)(!u \langle y \rangle. \hat{D}^y \mid \hat{E}^y)) \mid \hat{F}^z \\
\cong & \cong & \\
\text{cut}^! D (u. \text{cut}^! E_u (v. F_v)) & \rightsquigarrow & (\nu u)(!u \langle y \rangle. \hat{D}^y \mid (\nu v)(!v \langle y \rangle. \hat{E}^y \mid \hat{F}^z)) \\
\text{cut}^! D (u. E) & \rightsquigarrow & (\nu u)(!u \langle y \rangle. \hat{D}^y \mid \hat{E}^z) \\
\cong & \cong & \\
E & \rightsquigarrow & \hat{E}^z \quad (\text{for } u \notin FN(\hat{E}^z))
\end{array}$$

Fig. 7. Structural Conversions (III): Cut! Conversions

□

**Lemma 4.2.** Assume

- 1  $\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow P_1 :: x:!A$  with  $P_1 \xrightarrow{x(y)} P'_1$ ;
- 2  $\Gamma; \Delta_2, x:!A \vdash D_2 \rightsquigarrow Q_1 :: z:C$  with  $Q_1 \xrightarrow{(\nu y)x(y)} Q'_1$ .

Then

- 1  $\text{cut } D_1(x. D_2) \equiv \equiv D$  for some  $D$ ;
- 2  $\Gamma; \Delta_1, \Delta_2 \vdash D \rightsquigarrow Q_2 :: z : C$  for some  $Q_2 \equiv (\nu x)(\nu y)(P'_1 \mid Q'_1)$ .

*Proof.* See Appendix A.8.

□

The key idea of the lemmas above is that by relating process reduction with proof reduction at a given type we can conclude a strong form of type preservation, as follows.

**Theorem 4.1.** Let  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:A$  and  $P \rightarrow Q$ . Then there is an  $E$  such that  $D \equiv \equiv E$  and  $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$

*Proof.* See Appendix A.1.

□

Theorems 4.2 and 4.3 state that any proof reduction or conversion also corresponds to either a process equivalence or to a reduction step on the  $\pi$ -calculus.

**Theorem 4.2.** Let  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:A$  and  $D \simeq_s E$ . Then there is a  $Q$  where  $P \simeq_s Q$  and  $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$ .

*Proof.* Following the commuting squares relating  $\equiv$ ,  $\rightsquigarrow$  and  $\simeq$  in Figures 5, 6 and 7. □

**Theorem 4.3.** Let  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:A$  and  $D \Rightarrow E$ . Then there is a  $Q$  such that  $P \rightarrow Q$  and  $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$ .

*Proof.* Following the commuting squares relating  $\Rightarrow$ ,  $\rightsquigarrow$  and  $\rightarrow$  in Figure 4. □

Notice that the simulation of  $\pi$ -calculus reductions by proof term conversions provided by Theorem 4.1, and from which subject reduction follows, is very tight indeed, as reduction is simulated up to structural congruence, which is a very fine equivalence on processes. To that end, structural conversions need to be applied symmetrically (as equations), unlike in a standard proof of cut-elimination, where they are usually considered as directed computational steps. Under the assumptions of Theorem 4.1, we can also prove that there is an  $E$  such that  $D \Rightarrow E$  and  $\Gamma; \Delta \vdash E \rightsquigarrow R :: z:A$ , for  $Q \simeq_s R$ . Thus, even if one considers the proof conversions as directed reduction rules ( $\Rightarrow$ ), we still obtain a sound simulation up to typed strong behavioral congruence.

We now state type preservation and progress results for our type system. The subject reduction property (Theorem 4.4) directly follows from Theorem 4.1.

**Theorem 4.4 (Subject Reduction).** If  $\Gamma; \Delta \vdash P :: z:A$  and  $P \rightarrow Q$  then  $\Gamma; \Delta \vdash Q :: z:A$ .

Together with direct consequences of linear typing, Theorem 4.4 ensures session fidelity. Our type discipline also enforces a global progress property. For any  $P$ , define

$$\text{live}(P) \text{ iff } P \equiv (\nu \bar{n})(\pi.Q \mid R) \text{ for some } \pi.Q, R, \bar{n}$$

where  $\pi.Q$  is a *non-replicated* guarded process. We first establish the following contextual progress property, from which Theorem 4.5 follows as a corollary. Lemma 4.3 relies on an inversion lemma that relates types with action labels (Lemma A.1) and on a lemma that characterizes the typing of non-live processes (Lemma A.11).

**Lemma 4.3.** Let  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:C$ . If  $\text{live}(P)$  then there is a  $Q$  such that either

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

*Proof.* See Appendix A.12 □

**Theorem 4.5 (Progress).** If  $.; \cdot \vdash D \rightsquigarrow P :: x:1$  and  $\text{live}(P)$  then exists  $Q$  st.  $P \rightarrow Q$ .

*Proof.* See Appendix A.2 □

## 5. On Duality, and a Classical Linear Logic Interpretation

Although we have based our development on intuitionistic linear logic, the linear logic interpretation of session types also naturally extends to the classical setting. In this section, we briefly discuss such an alternative system, leaving a more comprehensive analysis for future investigation. The main characteristic of the classical interpretation is that it supports a full duality on the type structure. In traditional session type systems, e.g., (Gay and Hole 2005), a duality relation is defined on session types, in such a way that every session type  $S$  has a unique dual  $\bar{S}$ , given by:

$$\begin{array}{lcl} \overline{\text{end}} & \triangleq & \text{end} \\ \overline{T?.S} & \triangleq & T!.S \quad \overline{T!.S} & \triangleq & T?.S \\ \overline{S \oplus T} & \triangleq & \bar{S} \& \bar{T} \quad \overline{S \& T} & \triangleq & \bar{S} \oplus \bar{T} \end{array}$$

In our case, this would correspond to

$$\begin{array}{lcl} \bar{\mathbf{1}} & \triangleq & \mathbf{1} \\ \overline{T \multimap S} & \triangleq & T \otimes \bar{S} \quad \overline{T \otimes S} & \triangleq & T \multimap \bar{S} \\ \overline{S \oplus T} & \triangleq & \bar{S} \& \bar{T} \quad \overline{S \& T} & \triangleq & \bar{S} \oplus \bar{T} \end{array}$$

This duality relation does not talk about the type of shared channels. In traditional session types, the type structure is stratified, so that one distinguishes between “session types” and “standard types”, the latter intended to type (shared) session initiation channels. In our case, shared channels are naturally typed by the exponential  $!A$ , without any stratification on the type structure whatsoever. Nevertheless, in our system a related notion of polarity on types is already implicitly reflected by the left/right duality of intuitionistic sequents, in the sense that we can move all channels to the left-hand side of the sequent.

**Proposition 5.1.** Let  $A$  be an exponential-free type.  
Then  $\Gamma; \Delta \vdash P :: x:A$  implies  $\Gamma; \Delta, x:\bar{A} \vdash P :: -:\mathbf{1}$ .

*Proof.* By induction on the structure of the given proof (see Appendix A.4).  $\square$

So if we have  $P :: x:A$  and  $Q :: x:\bar{A}$  (where  $A$  has no occurrence of ‘!’) we can compose  $P$  and  $Q$  as  $(\nu x)(P \mid Q) :: -:\mathbf{1}$  using the above proposition and the cut rule.

A key distinctive aspects of our intuitionistic interpretation is the natural reading it offers of session typed processes as systems that implement a provided session (or service) based on a set of required sessions (or services). Another essential aspect is the asymmetry it introduces in the treatment of the exponential  $!A$ , whose “dual” behavior, in the sense discussed above, is not available in the type structure (interestingly, like in traditional session types). Intuitively, while the type of a shared server located at name  $s$  and providing protocols of type  $A$  is  $!A$ , and expresses the capability to **receive** (at  $s$ ) an unbounded number of concurrent incoming service request messages, the dual behavior should express the capability to **send** (to  $s$ ) an unbounded number of concurrent incoming service request messages for a local endpoint satisfying the protocol  $\bar{A}$ .

A full duality in the context of the interpretation just sketched may be recovered by resorting to an interpretation in classical linear logic (CLL), which assigns a dual type to every session type, in particular to  $!A$ , by introducing the “why-not” exponential connective  $?A$ . We may then consider the following type structure

**Definition 5.2 (C-Types).** C-Types  $(A, B, C)$  are given by

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

The input session type  $A \multimap B$  is here modeled by  $\bar{A} \wp B$ , following the interpretation of the linear implication in classical linear logic. The interpretation of  $A \wp B$  poses no problems: it types a session that inputs a channel of type  $A$ , and continues as  $B$ . It should be clear that the issues of the apparent asymmetry of  $A \wp B$  can be explained as we did for  $A \otimes B$  in Section 3.6. Moreover, we may define a full duality on C-Types, which exactly corresponds to the negation operator of CLL  $(\cdot)^\perp$ .

$$\begin{array}{lcl} \bar{\mathbf{1}} & = & \perp \\ \overline{!A} & = & ?\bar{A} \\ \overline{A \otimes B} & = & \bar{A} \wp \bar{B} \\ \overline{A \oplus B} & = & \bar{A} \& \bar{B} \end{array} \quad \begin{array}{lcl} \bar{\perp} & = & \mathbf{1} \\ \overline{?A} & = & !\bar{A} \\ \overline{A \wp B} & = & \bar{A} \otimes \bar{B} \\ \overline{A \& B} & = & \bar{A} \oplus \bar{B} \end{array}$$

To be absolutely faithful to the classical interpretation, and without loss of expressiveness with respect to traditional session types, we split the “session termination” type **end** in two different types:  $\mathbf{1}$  and  $\perp$ , the units for  $A \otimes B$  and  $A \wp B$ . These may be understood as representing the session closure behaviors implemented by matching endpoints. Alternatively, we could postulate  $\bar{\mathbf{1}} = \mathbf{1}$  (cf.  $\overline{\mathbf{end}} = \mathbf{end}$ ) which would lead to a slight deviation from classical linear logic and validate the nullary version of the mix rule (Girard(1987)), making  $\vdash \cdot$  derivable. This, and a further discussion of the general mix rule is beyond the scope of this paper.

$$\begin{array}{c}
\frac{}{\mathbf{0} \vdash x:\mathbf{1}; \Theta} \text{(T1)} \quad \frac{P \vdash \Delta; \Theta}{P \vdash x:\perp, \Delta; \Theta} \text{(T}\perp\text{)} \\
\frac{P \vdash \Delta, y:A, x:B; \Theta}{x(y).P \vdash \Delta, x:A \wp B; \Theta} \text{(T}\wp\text{)} \\
\frac{P \vdash \Delta, y:A; \Theta \quad Q \vdash \Delta', x:B; \Theta}{(\nu y)x\langle y \rangle.(P \mid Q) \vdash \Delta, \Delta', x:A \otimes B; \Theta} \text{(T}\otimes\text{)} \\
\frac{P \vdash \Delta, x:\bar{A}; \Theta \quad Q \vdash \Delta', x:A; \Theta}{(\nu x)(P \mid Q) \vdash \Delta, \Delta'; \Theta} \text{(Tcut)} \quad \frac{P \vdash y:\bar{A}; \Theta \quad Q \vdash \Delta; u:A, \Theta}{(\nu u)(!u(y).P \mid Q) \vdash \Delta; \Theta} \text{(Tcut}^?\text{)} \\
\frac{P \vdash \Delta, y:A; u:A, \Theta}{(\nu y)u\langle y \rangle.P \vdash \Delta; u:A, \Theta} \text{(Tcopy)} \\
\frac{P \vdash \Delta; u:A, \Theta}{P\{x/u\} \vdash \Delta, x:?A; \Theta} \text{(T?)} \quad \frac{Q \vdash y:A; \Theta}{!x(y).Q \vdash x:!A; \Theta} \text{(T!)} \\
\frac{P \vdash \Delta, x:A; \Theta}{x.\text{inl}; P \vdash \Delta, x:A \oplus B; \Theta} \text{(T}\oplus_1\text{)} \quad \frac{P \vdash \Delta, x:B; \Theta}{\vdash x.\text{inr}; P \vdash \Delta, x:A \oplus B; \Theta} \text{(T}\oplus_2\text{)} \\
\frac{P \vdash \Delta, x:A; \Theta \quad Q \vdash \Delta, x:B; \Theta}{x.\text{case}(P, Q) \vdash \Delta, x:A \& B; \Theta} \text{(T}\&\text{)}
\end{array}$$

Fig. 8. The Type System  $\pi\text{CLL}$ .

C-Types may be assigned to  $\pi$ -calculus processes by following the key ideas of our interpretation, by a type system  $\pi\text{CLL}$  that corresponds this time (exactly) to Andreoli's dyadic system (Andreoli 1992). We present  $\pi\text{CLL}$  in Figure 8.

Typing judgments in  $\pi\text{CLL}$  have the (one sided) form  $P \vdash \Delta; \Theta$ , where  $P$  is a  $\pi$ -calculus process,  $\Delta$  is a linear context, and  $\Theta$  is a context with declarations  $u:A$  which may be read as  $u:?A$  in the original one-sided formulation of classical linear logic. A remarkable consequence of the classical discipline is the loss of the *locality property* (Merro and Sangiorgi 2004) on shared channels, considered in general the most sensible for distributed implementations of shared channels. In the  $\pi$ -calculus, locality enforces that processes may not receive messages on previously received channel names, that is, only the output capability of a communication channel may be delegated in a distributed communication. In a very informal sense, non-locality means that a receiver may create a local stub for receiving a message on a shared channel created elsewhere, possibly causing undesirable interference. In  $\pi\text{CLL}$ , a well-typed process may input a channel of type  $?A$ , as the type  $?A \multimap B$  now becomes expressible, thus breaking locality on shared channels, a behavior excluded by the intuitionistic discipline.

We give an example of such a process. Let

$$C \triangleq (\nu x)(x(y).!y(z).P_x \mid (\nu q)x\langle q \rangle.(Q_q \mid R_x))$$

where  $Q_q \triangleq ((\nu k_1)q\langle k_1 \rangle.Q_1 \mid (\nu k_2)q\langle k_2 \rangle.Q_2)$ . Notice that  $C$  contains the subsystem

$$S_x \triangleq x(y).!y(z).P_x$$

The process  $S_x$  cannot be typed in  $\pi\text{DILL}$ . We may interpret  $S_x$  as a generator of remotely located shared server.  $S_x$  starts off by receiving a fresh channel name  $n$  from

the sender client (on  $x$ ) and then instantiates a replicated (receiver) shared server of the form  $!n(z).P_x$ . We illustrate the type derivation for subprocess  $S_x$

- 1  $P_v \vdash z:A; v:G$  (assumption, for some  $P_v$ )
- 2  $!y(z).P_v \vdash y:!A; v:G$  (T! from 1)
- 3  $!y(z).P_x \vdash y:!A, x:?G; \cdot$  (T? from 2)
- 4  $x(y).!y(z).P_x \vdash x:!A\wp?G$  (T $\wp$  from 3, notice that  $!A\wp?G = ?\bar{A} \multimap ?G$ )

On the other hand the subprocess  $(\nu q)x\langle q \rangle.(Q_q \mid R_x)$  sends on  $x$  a fresh channel of type  $?\bar{A}$ , in which two (in this case) different requests will be issued to the remotely allocated server (see the definition of  $Q_q$  above). This non-local behavior of shared channel names is not expressible in  $\pi$ DILL; we consider quite remarkable how the intuitionistic discipline, which is logically motivated, enforces the locality discipline, a behavioral concept related to distributed communicating systems.

## 6. Further Discussion

We further compare our linear type system for (finite) session types with more familiar session type systems (Kobayashi *et al.* 1996; Honda *et al.* 1998; Gay and Hole 2005).

An immediate observation is that in our case types are freely generated, while traditionally there is a stratification of types in “session” and “standard types” (the later corresponding to our  $!A$  types, typing session initiation channels). In our interpretation, a session may either terminate (**1**), or become a replicated server ( $!A$ ), which reveals a more general and uniform type structure than the ones proposed in (Kobayashi *et al.* 1996; Honda *et al.* 1998; Gay and Hole 2005), which cannot express a type such as  $A \multimap !S$ , which describes a process that inputs a session of type  $A$  and then behaves as a replicated server of type  $!S$ .

Channel “polarities” are captured in our system by the left-right distinction of sequents, rather than by annotations on channels (cf.  $x^+, x^-$ ). Session and linear type systems (Kobayashi *et al.* 1996; Honda *et al.* 1998; Gay and Hole 2005) also include a typing rule for output of the form

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

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

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

The copycat  $id_A(y, z)$  plays the role of the “link” processes of (Sangiorgi 1996; Boreale 1998). Notice that in our case the definition of the “link” is obtained for free by the interpretation of identity axioms (Proposition 3.1). The two processes depicted above can be shown to be behaviorally equivalent, under an adequate notion of observational equivalence, along the lines of (Boreale 1998).

Concerning parallel composition of processes, usually two rules can be found, one rule

corresponding to the interconnection of two dual session endpoints (implemented by a name restriction rule), and other rule corresponding to independent parallel composition, also present in most linear type systems for mobile processes. In our case, the cut rule combines both principles, and the following rule is derivable:

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

A consequence of the logical nature of our composition principles *cut* and *cut*<sup>1</sup> is that our typing discipline intrinsically enforces global progress, unlike with traditional session type systems (Honda *et al.* 1998; Gay and Hole 2005), which do not ensure progress in general, as we achieve in this work. The session composition rule found in such systems does not take into account the causal dependency between different sessions, and validates the connection of two session endpoints just by requiring duality of their polarity. For example, the system of (Gay and Hole 2005) contains a type rule of the form

$$\frac{\Gamma, x^+ : S, x^- : S' \vdash P \quad S \perp S'}{\Gamma \vdash (\nu x : S)P}$$

where  $S \perp S'$  expresses the duality of the session types  $S$  and  $S'$ . This rule ensures that only dual protocols are connected to form a session, but of course does not prevent causality cycles from appearing in systems with multiple open sessions. Techniques to ensure progress in traditional session type systems, typically building on extraneous devices such as well-founded orderings on events, have been proposed by several authors (Kobayashi 1998; Dezani-Ciancaglini *et al.* 2008).

We find it important to identify systems of session types as the one we have identified, in which progress (deadlock-freedom) is an essential meta-theoretical property, as is the case of basic type systems for foundational programming languages, in particular, for typed  $\lambda$ -calculi. Moreover, we have not been able to find an example of a interesting session typed system not typable in our framework. The work reported in this paper started a research program in which structured concurrency communication is approached from several perspectives, always based on canonical logical principles. For example, in (Toninho *et al.* 2011) we have shown how to specify rich interface contracts with dependent types, while in (Pfenning *et al.* 2011) we have introduced proof-carrying code with digital certificates in our basic framework, always based on purely logical constructions. There are not many studies comparing the expressiveness of session type systems, and that also clearly seems a challenging research direction. An important instrument towards that goal is, we believe, a better understanding of observational equivalences under the session type discipline.

## 7. Related Work and Conclusion

We have established a tight correspondence between a session-based type discipline for the  $\pi$ -calculus and intuitionistic linear logic: typing rules correspond to dual intuitionistic linear sequent calculus proof rules, moreover process reduction may be simulated in a type preserving way by proof conversions and reductions, and *vice versa*. As a result,

we obtain the subject reduction property, from which session fidelity follows. Our basic typing discipline intrinsically ensures global progress, beyond the restricted “progress on a single session” property obtained in pure session type systems.

Other works have investigated  $\pi$ -calculus models of linear logic proofs. Bellin and Scott (Bellin and Scott 1994) establish a mapping from linear logic proofs to a variant of the  $\pi$ -calculus and some connections between proof reduction and  $\pi$ -calculus reduction. However, this mapping results in complex encodings, so that their system could hardly be considered a type assignment system for processes, which has been achieved in this work. Moreover, no relation between behavioral descriptions and logical propositions was identified, as put by the authors: “[our encodings] have less to do with logic than one might think, they are essentially only about the abstract pluggings in proof structures”.

A realizability interpretation for a linear logic augmented with temporal modalities (cf. Hennessy-Milner) was proposed in (Beffara 2006), also based on a  $\pi$ -calculus variant. A recent related development is (Honda and Laurent 2010), where a correspondence between (independently formulated) proof nets and an IO-typed  $\pi$ -calculus is established. In our case, the type system and the logic proof system are exactly the same, and we reveal a direct connection between pure linear logic propositions and behavioral types on  $\pi$ -calculus, that covers all (both shared and linear) features of finite session types. A development of session types as linear process types (in the sense of (Kobayashi *et al.* 1996)) is presented in (Giunti and Vasconcelos 2010), where linearity and sharing are expressed by special annotations, unrelated to a linear logic interpretation.

We have also analyzed the relation between our type discipline and (finite, deadlock-free) session types. It is important to notice that our interpretation does not require locality (Merro and Sangiorgi 2004) for linear session channels, under which only the output capability of names could be transmitted, which seems required in other works on linearity for  $\pi$ -calculi (e.g., (Yoshida *et al.* 2007)). On the other hand, our intuitionistic discipline enforces locality of shared channels, which, quite interestingly, seems to be the sensible choice for distributed implementations of sessions. Further related topics would be the accommodation of recursive types and logical relations (Caires 2007).

One important motivation for choosing a purely logical approach to typing is that it often suggests uniform and expressive generalizations. In ongoing work, we have also established an explicit relationship between session-based concurrency and typed functional computation where in both cases determinacy (no races) and progress (deadlock-freedom) are expected properties. In particular, we have investigated new encodings of  $\lambda$ -calculi into the  $\pi$ -calculus that arise from translations from DILL natural deduction into sequent calculus and the reduction strategies they induce (Toninho *et al.* 2012). We have also explored a dependent generalization of our system of simple linear types, which successfully captures many additional properties of communication behavior in a purely logical manner (Toninho *et al.* 2011). Furthermore, the combination of dependent session types, proof irrelevance and a notion of affirmation allows us to capture a high-level model of certified, distributed code (Pfenning *et al.* 2011).

## References

- S. Abramsky. Computational Interpretations of Linear Logic. *Theor. Comp. Sci.*, 111(1&2), 1993.
- J.-M. Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.
- A. Barber and G. Plotkin. Dual Intuitionistic Linear Logic. Technical Report LFCS-96-347, Univ. of Edinburgh, 1997.
- E. Beffara. A Concurrent Model for Linear Logic. *ENTCS*, 155:147–168, 2006.
- G. Bellin and P. Scott. On the  $\pi$ -Calculus and Linear Logic. *Theor. Comp. Sci.*, 135:11–65, 1994.
- E. Bonelli, A. Compagnoni, and E. L. Gunter. Correspondence Assertions for Process Synchronization in Concurrent Communications. *J. of Func. Prog.*, 15(2):219–247, 2005.
- M. Boreale. On the Expressiveness of Internal Mobility in Name-Passing Calculi. *Theoretical Computer Science*, 195(2):205–226, 1998.
- L. Caires. Logical Semantics of Types for Concurrency. In *International Conference on Algebra and Coalgebra in Computer Science*, CALCO’07, 16–35. Springer LNCS 4624, 2007.
- I. Cervesato and F. Pfenning. A linear logical framework. *Inf. & Comput.*, 179(1), 2002.
- 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.
- M. Dezani-Ciancaglini and U. de’ Liguoro. Sessions and Session Types: an Overview. In *6th Intl Workshop on Web Services and Formal Methods WS-FM’09*, Lecture Notes in Computer Science. Springer-Verlag, 2010.
- M. Dezani-Ciancaglini, U. de’ Liguoro, and N. Yoshida. On Progress for Structured Communications. In G. Barthe and C. Fournet, editors, *3th Symposium Trustworthy Global Computing, TGC 2007*, pages 257–275. Springer LNCS 4912, 2008.
- S. Gay and M. Hole. Subtyping for Session Types in the Pi Calculus. *Acta Informatica*, 42(2-3):191–225, 2005.
- J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
- J.-Y. Girard and Y. Lafont. Linear Logic and Lazy Computation. In H. Ehrig, R. A. Kowalski, G. Levi, and U. Montanari, editors, *Theory and Practice of Software Development, TAP-SOFT’87*, pages 52–66. Springer LNCS 250, 1987.
- M. Giunti and V. T. Vasconcelos. A Linear Account of Session Types in the Pi-Calculus. In *21st International Conference on Concurrency Theory, CONCUR’10*, pages 432–446. Springer LNCS 6269, 2010.
- K. Honda. Types for dyadic interaction. In *4th International Conference on Concurrency Theory, CONCUR’93*, pages 509–523. Springer LNCS 715, 1993.
- K. Honda and O. Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theor. Comp. Sci.*, 411:2223–2238, 2010.
- K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP’98*, pages 122–138. Springer LNCS 1381, 1998.
- J. M. E. Hyland and C.-H. L. Ong. Pi-Calculus, Dialogue Games and PCF. In *WG2.8 Conference on Functional Programming Languages*, pages 96–107, 1995.
- N. Kobayashi. A Partially Deadlock-Free Typed Process Calculus. *ACM Tr. Progr. Lang. Sys.*, 20(2):436–482, 1998.
- N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. In *23rd Symposium on Principles of Programming Languages, POPL’96*, pages 358–371. ACM, 1996.

- M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *Math. Struct. in Comp. Sci.*, 14(5): 715-767, 2004.
- R. Milner. Functions as processes. *Math. Struct. in Comp. Sci.*, 2(2):119-141, 1992.
- J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Termination in session-based concurrency via linear logical relations. *22nd European Symposium on Programming, ESOP'12*, Tallinn, Estonia, Mar. 2012. To appear.
- F. Pfenning and L. Caires and B. Toninho. Proof-Carrying Code in a Session-Typed Process Calculus. *Certified Programs and Proofs, CPP'11*, 21-36, 2011.
- D. Sangiorgi. Pi-Calculus, Internal Mobility, and Agent Passing Calculi. *Theoretical Computer Science*, 167(1&2):235-274, 1996.
- D. Sangiorgi and D. Walker. *The  $\pi$ -calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
- B. Toninho and L. Caires and F. Pfenning. Dependent Session Types via Intuitionistic Linear Type Theory. *Principles and Practice of Declarative Programming, PPDP'11*, 161-172, 2011.
- B. Toninho and L. Caires and F. Pfenning. Functions as Session-Typed Processes. *15th International Conference on Foundations of Software Science and Computation Structures, FoS-SaCS'12*, Tallinn, Estonia, Mar. 2012. To appear.
- K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. Specifying properties of concurrent computations in CLF. In C.Schürmann, editor, *4th Intl. Workshop on Logical Frameworks and Meta-Languages (LFM'04)*, Cork, Ireland, July 2004. ENTCS, vol 199.
- N. Yoshida, K. Honda, and M. Berger. Linearity and Bisimulation. *J. Logic and Algebraic Programming*, 72(2):207-238, 2007.

## Appendix A. Proofs

### A.1. Inversion Lemmas

**Lemma A.1.** Let  $\Gamma; \Delta \vdash D \rightsquigarrow P :: x : C$ .

- 1 If  $P \xrightarrow{\alpha} Q$  and  $C = \mathbf{1}$  then  $s(\alpha) \neq x$ .
- 2 If  $P \xrightarrow{\alpha} Q$  and  $y : \mathbf{1} \in \Delta$  then  $s(\alpha) \neq y$ .
- 3 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = A \otimes B$  then  $\alpha = \overline{(\nu y)x\langle y \rangle}$ .
- 4 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \otimes B \in \Delta$  then  $\alpha = y\langle z \rangle$ .
- 5 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = A \multimap B$  then  $\alpha = x(y)$ .
- 6 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \multimap B \in \Delta$  then  $\alpha = \overline{(\nu z)y\langle z \rangle}$ .
- 7 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = A \& B$  then  $\alpha = x.\text{inl}$  or  $\alpha = x.\text{inr}$ .
- 8 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \& B \in \Delta$  then  $\alpha = \overline{y.\text{inl}}$  or  $\alpha = \overline{y.\text{inr}}$ .
- 9 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = A \oplus B$  then  $\alpha = x.\text{inl}$  or  $\alpha = x.\text{inr}$ .
- 10 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \oplus B \in \Delta$  then  $\alpha = y.\text{inl}$  or  $\alpha = y.\text{inr}$ .
- 11 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = !A$  then  $\alpha = x(y)$ .
- 12 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : !A$  or  $y \in \Gamma$  then  $\alpha = \overline{(\nu z)y\langle z \rangle}$ .

*Proof.* By induction on the structure of  $D$ .

- 1 If  $P \xrightarrow{\alpha} Q$  and  $C = \mathbf{1}$  then  $s(\alpha) \neq x$ .

**Case:copy**, all left rules except **1L** and **!L**

$$s(\alpha) \neq x$$

by the definition of the l.t.s.

**Case:1L or !L**

$s(\alpha) \neq x$  by i.h.

**Case:cut**  $D_1 (y. D_2)$

$D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$

**Subcase:**  $P_1 \xrightarrow{\alpha} Q_1$

$s(\alpha) \neq x$  trivial, since  $x \notin fn(P_1)$  by typing

**Subcase:**  $P_2 \xrightarrow{\alpha} Q_2$

$s(\alpha) \neq x$  by i.h. on  $D_2$

**Case:cut<sup>!</sup>**  $D_1 (u. D_2)$

$D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$

**Subcase:**  $P_1 \xrightarrow{\alpha} Q_1$

$s(\alpha) \neq x$  trivial, since  $x \notin fn(P_1)$  by typing

**Subcase:**  $P_2 \xrightarrow{\alpha} Q_2$

$s(\alpha) \neq x$  by i.h. on  $D_2$

**Case:**All other rules do not offer  $\xrightarrow{\alpha}$  or  $C \neq \mathbf{1}$

2 If  $P \xrightarrow{\alpha} Q$  and  $y : \mathbf{1} \in \Delta$  then  $s(\alpha) \neq y$ .

**Case:copy**, all left rules except **1L** and **!L**

$s(\alpha) \neq y$  by the definition of the l.t.s.

**Case:1L or !L**

$s(\alpha) \neq y$  by i.h.

**Case:** $\Gamma; \Delta_1, \Delta_2 \vdash \text{cut } D_1 (z. D_2) \rightsquigarrow P :: x : C$ , with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$

**Subcase:**  $y : \mathbf{1} \in \Delta_1$

$s(\alpha) \neq y$  by i.h. and  $y \notin fn(P_2)$

**Subcase:**  $y : \mathbf{1} \in \Delta_2$  with  $z \neq y$

$s(\alpha) \neq y$  by i.h. and  $y \notin fn(P_1)$

**Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$

**Subcase:**  $P_1 \xrightarrow{\alpha} Q_1$

$s(\alpha) \neq y$  trivial, since linear ctxt. is empty for  $D_1$  and  $y : \mathbf{1}$

**Subcase:**  $P_2 \xrightarrow{\alpha} Q_2$

$s(\alpha) \neq y$  by i.h.

3 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = A \otimes B$  then  $\alpha = \overline{(\nu y)x\langle y \rangle}$

**Case:1L or !L**

$\alpha = \overline{(\nu y)x\langle y \rangle}$  by i.h.

**Case:** $\otimes$ R

$\alpha = \overline{(\nu y)x\langle y \rangle}$  by the l.t.s

**Case:cut**  $D_1 (y. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$

$\alpha = \overline{(\nu y)x\langle y \rangle}$  by i.h. on  $D_2$  and  $x \notin fn(P_1)$

- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = \overline{(\nu y)x\langle y \rangle}$  by i.h. on  $D_2$
- Case:**All other rules do not have  $s(\alpha) = x$  and  $C = A \otimes B$
- 4 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \otimes B \in \Delta$  then  $\alpha = y(z)$
- Case:1L** or **!L**  
 $\alpha = y(z)$  by i.h.
- Case: $\otimes$ L**  
 $\alpha = y(z)$  by the l.t.s
- Case:** $\Gamma; \Delta_1, \Delta_2 \vdash \text{cut } D_1 (z. D_2) \rightsquigarrow P :: x : C$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$
- Subcase:**  $y \in \Delta_1$   
 $\alpha = y(z)$  by i.h. on  $D_1$  and  $y \notin \text{fn}(P_2)$
- Subcase:**  $y \in \Delta_2$   
 $\alpha = y(z)$  by i.h. on  $D_2$  and  $y \notin \text{fn}(P_1)$
- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = y(z)$  by i.h. on  $D_2$
- 5 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = A \multimap B$  then  $\alpha = x(y)$
- Case:1L** or **!L**  
 $\alpha = x(y)$  by i.h.
- Case: $\multimap$ R**  
 $\alpha = x(y)$  by the l.t.s
- Case:cut**  $D_1 (y. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = x(y)$  by i.h. on  $D_2$  and  $x \notin \text{fn}(P_1)$
- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = x(y)$  by i.h. on  $D_2$
- Case:**All other rules do not have  $s(\alpha) = x$  and  $C = A \multimap B$
- 6 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \multimap B \in \Delta$  then  $\alpha = \overline{(\nu z)y\langle z \rangle}$
- Case:1L** or **!L**  
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h.
- Case: $\multimap$ L**  
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by the l.t.s
- Case:** $\Gamma; \Delta_1, \Delta_2 \vdash \text{cut } D_1 (w. D_2) \rightsquigarrow P :: x : C$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$
- Subcase:**  $y \in \Delta_1$   
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h. on  $D_1$  and  $y \notin \text{fn}(P_2)$
- Subcase:**  $y \in \Delta_2$   
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h. on  $D_2$  and  $y \notin \text{fn}(P_1)$

- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h. on  $D_2$
- 7 If  $P \overset{\alpha}{\rightarrow} Q$  and  $s(\alpha) = x$  and  $C = A \& B$  then  $\alpha = x.\text{inl}$  or  $\alpha = x.\text{inr}$   
**Case:1L** or **!L**  
 $\alpha = x.\text{inl}$  or  $\alpha = x.\text{inr}$  by i.h.
- Case:&R**  
 $\alpha = x.\text{inl}$  or  $\alpha = x.\text{inr}$  by the l.t.s
- Case:cut**  $D_1 (y. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = x(y)$  by i.h. on  $D_2$  and  $x \notin \text{fn}(P_1)$
- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = x(y)$  by i.h. on  $D_2$
- Case:**All other rules do not have  $s(\alpha) = x$  and  $C = A \& B$
- 8 If  $P \overset{\alpha}{\rightarrow} Q$  and  $s(\alpha) = y$  and  $y : A \& B \in \Delta$  then  $\alpha = \overline{y.\text{inl}}$  or  $\alpha = \overline{y.\text{inr}}$   
**Case:1L** or **!L**  
 $\alpha = \overline{y.\text{inl}}$  or  $\alpha = \overline{y.\text{inr}}$  by i.h.
- Case:&L<sub>1</sub>**  
 $\alpha = \overline{y.\text{inl}}$  by the l.t.s
- Case:&L<sub>2</sub>**  
 $\alpha = \overline{y.\text{inr}}$  by the l.t.s
- Case:** $\Gamma; \Delta_1, \Delta_2 \vdash \text{cut } D_1 (w. D_2) \rightsquigarrow P :: x : C$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
**Subcase:**  $y \in \Delta_1$   
 $\alpha = \overline{y.\text{inl}}$  or  $\alpha = \overline{y.\text{inr}}$  by i.h. on  $D_1$  and  $y \notin \text{fn}(P_2)$   
**Subcase:**  $y \in \Delta_2$   
 $\alpha = \overline{y.\text{inl}}$  or  $\alpha = \overline{y.\text{inr}}$  by i.h. on  $D_2$  and  $y \notin \text{fn}(P_1)$
- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = \overline{y.\text{inl}}$  or  $\alpha = \overline{y.\text{inr}}$  by i.h. on  $D_2$
- 9 If  $P \overset{\alpha}{\rightarrow} Q$  and  $s(\alpha) = x$  and  $C = A \oplus B$  then  $\alpha = \overline{x.\text{inl}}$  or  $\alpha = \overline{x.\text{inr}}$   
**Case:1L** or **!L**  
 $\alpha = \overline{x.\text{inl}}$  or  $\alpha = \overline{x.\text{inr}}$  by i.h.
- Case: $\oplus R_1$**   
 $\alpha = \overline{x.\text{inl}}$  by the l.t.s
- Case: $\oplus R_2$**   
 $\alpha = \overline{x.\text{inr}}$  by the l.t.s
- Case:cut**  $D_1 (y. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = \overline{x.\text{inl}}$  or  $\alpha = \overline{x.\text{inr}}$  by i.h. on  $D_2$  and  $x \notin \text{fn}(P_1)$

- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = \overline{x.inl}$  or  $\alpha = \overline{x.inr}$  by i.h. on  $D_2$
- Case:**All other rules do not have  $s(\alpha) = x$  and  $C = A \oplus B$
- 10 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : A \oplus B \in \Delta$  then  $\alpha = y.inl$  or  $\alpha = y.inr$
- Case:1L** or **!L**  
 $\alpha = \overline{y.inl}$  or  $\alpha = \overline{y.inr}$  by i.h.
- Case: $\oplus$ L**  
 $\alpha = \overline{y.inl}$  or  $\alpha = \overline{y.inr}$  by the l.t.s
- Case:** $\Gamma; \Delta_1, \Delta_2 \vdash \text{cut } D_1 (w. D_2) \rightsquigarrow P :: x : C$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$
- Subcase:**  $y \in \Delta_1$   
 $\alpha = \overline{y.inl}$  or  $\alpha = \overline{y.inr}$  by i.h. on  $D_1$  and  $y \notin fn(P_2)$
- Subcase:**  $y \in \Delta_2$   
 $\alpha = \overline{y.inl}$  or  $\alpha = \overline{y.inr}$  by i.h. on  $D_2$  and  $y \notin fn(P_1)$
- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = \overline{y.inl}$  or  $\alpha = \overline{y.inr}$  by i.h. on  $D_2$
- 11 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = x$  and  $C = !A$  then  $\alpha = x(y)$
- Case:1L** or **!L**  
 $\alpha = x(y)$  by i.h.
- Case:!R**  
 $\alpha = x(y)$  by the l.t.s
- Case:cut**  $D_1 (z. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = x(y)$  by i.h. on  $D_2$  and  $x \notin fn(P_1)$
- Case:cut<sup>!</sup>**  $D_1 (u. D_2)$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$   
 $\alpha = x(y)$  by i.h. on  $D_2$
- Case:**All other rules do not have  $s(\alpha) = x$  and  $C = !A$
- 12 If  $P \xrightarrow{\alpha} Q$  and  $s(\alpha) = y$  and  $y : !A$  or  $y \in \Gamma$  then  $\alpha = \overline{(\nu z)y\langle z \rangle}$ .
- Case:1L** or **!L**  
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h.
- Case:copy**  
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by the l.t.s
- Case:** $\Gamma; \Delta_1, \Delta_2 \vdash \text{cut } D_1 (w. D_2) \rightsquigarrow P :: x : C$  with  $D_1 \rightsquigarrow P_1$  and  $D_2 \rightsquigarrow P_2$
- Subcase:**  $y : !A$  and  $y \in \Delta_1$   
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h. on  $D_1$  and  $y \notin fn(P_2)$
- Subcase:**  $y : !A$  and  $y \in \Delta_2$   
 $\alpha = \overline{(\nu z)y\langle z \rangle}$  by i.h. on  $D_2$  and  $y \notin fn(P_1)$
- Subcase:**  $y \in \Gamma$  and  $P \xrightarrow{\alpha} Q$  from  $P_1 \xrightarrow{\alpha} Q_1$

$\alpha = \overline{(\nu z)y\langle z \rangle}$	by i.h. on $D_1$
<b>Subcase:</b> $y \in \Gamma$ and $P \xrightarrow{\alpha} Q$ from $P_2 \xrightarrow{\alpha} Q_2$	
$\alpha = \overline{(\nu z)y\langle z \rangle}$	by i.h. on $D_2$
<b>Case:</b> $\text{cut}^! D_1 (u. D_2)$ with $D_1 \rightsquigarrow P_1$ and $D_2 \rightsquigarrow P_2$	
<b>Subcase:</b> $y : !A$	
$\alpha = \overline{(\nu z)y\langle z \rangle}$	by i.h. on $D_2$
<b>Subcase:</b> $y \in \Gamma$	
$\alpha = \overline{(\nu z)y\langle z \rangle}$	by i.h. on $D_2$

□

## A.2. Preservation Lemmas

**Lemma A.2.** Assume

- (a)  $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:C_1 \ \& \ C_2$  with  $P \xrightarrow{x.\text{inl}} P'$ ;
- (b)  $\Gamma; \Delta_2, x:C_1 \ \& \ C_2 \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{x.\text{inl}} Q'$ .

Then

- (c)  $\text{cut } D(x. E) \equiv \Rightarrow \equiv F$  for some  $F$ ;
- (d)  $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu x)(P' \mid Q')$ .

*Proof.* By simultaneous induction on  $D$  and  $E$ . That is, in each appeal to the induction hypothesis either  $D$  becomes smaller and  $E$  remains the same, or  $D$  remains the same and  $E$  becomes smaller.

The possible cases for  $D$  are  $\&R$ ,  $\mathbf{1L}$ ,  $!L$ ,  $\text{cut}$ , and  $\text{cut}^!$ . In all other cases  $P^x$  cannot offer  $x.\text{inl}$ . The possible cases for  $E$  are  $\&L_1$ ,  $\mathbf{1L}$ ,  $!L$ ,  $\text{cut}$ , and  $\text{cut}^!$ .

**Case:**  $D = \&R D_1 D_2$  and  $E = \&L_1 x (x. E_1)$ .

$$\text{cut } D (x. E) = \text{cut } (\&R D_1 D_2) (\&L_1 x (x. E_1)) = F \quad \text{[a]}$$

$$F \rightsquigarrow (\nu x)(x.\text{case}(P_1^x, P_2^x) \mid x.\text{inl}; Q_1^z) = (\nu x)(P^x \mid Q^z) \quad \text{[b]}$$

$$F \Rightarrow \text{cut } D_1 (x. E_1) = F' \quad \text{By reduction (cut/\&R/\&L_1) [c]}$$

$$F' \rightsquigarrow (\nu x)(P_1^x \mid Q_1^z) \equiv R' \text{ with } P_1^x = P'^x \text{ and } Q_1^z = Q'^z \quad \text{[d]}$$

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

$$\begin{aligned} \text{cut } D (x. E) &= \text{cut } (\mathbf{1L} y D_1) (x. E) \\ &\equiv \mathbf{1L} y (\text{cut } D_1 (x. E)) = F \end{aligned} \quad \text{By rule (cut/\mathbf{1L}/-) [a]}$$

$$F \rightsquigarrow (\nu x)(P^x \mid Q^z) = R \quad \text{since } D_1 \rightsquigarrow P^x \text{ [b]}$$

$$\text{cut } D_1 (x. E) \equiv F_1 \text{ for some } F_1;$$

$$F_1 \rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z);$$

$$F_1 \Rightarrow F'_1;$$

$$F'_1 \rightsquigarrow R_1'^z \equiv (\nu x)(P'^x \mid Q'^z) \quad \text{By i.h. on } D_1, E$$

$$F = \mathbf{1L} y F_1 \Rightarrow \mathbf{1L} y F'_1 = F' \quad \text{By congruence [c]}$$

$$F' \rightsquigarrow R_1^z \equiv (\nu x)(P'^x \mid Q'^z) \quad \text{[d]}$$

**Case:**  $D = !L y (u. D_1)$  and  $E$  arbitrary.

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut} (!L y (u. D_1)) (x. E) \\
&\equiv !L y (u. \text{cut } D_1 (x. E)) = F && \text{By rule (cut/!L/-) [a]} \\
F &\rightsquigarrow (\nu x)(P^x | Q^z) = R && \text{since } D_1 \rightsquigarrow P^x\{u/y\} \text{ [b]} \\
\text{cut } D_1 (x. E) &\equiv F_1 \text{ for some } F_1; \\
F_1 &\rightsquigarrow R_1^z \equiv (\nu x)(P^x | Q^z); \\
F_1 &\Rightarrow F'_1; \\
F'_1 &\rightsquigarrow R_1'^z \equiv (\nu x)(P'^x | Q'^z) && \text{By i.h. on } D_1, E \\
F &= !L y (u. F_1) \Rightarrow !L y (u. F'_1) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow R_1'^z \equiv (\nu x)(P'^x | Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $D = \text{cut } D_1 (y. D_2)$  and  $E$  arbitrary.

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut} (\text{cut } D_1 (y. D_2)) (x. E) \\
&\equiv \text{cut } D_1 (y. \text{cut } D_2 (x. E)) = F && \text{By rule (cut/-/cut}_1\text{) [a]} \\
F &\rightsquigarrow (\nu y)(P_1^y | ((\nu x)(P_2^x | Q^z))) \equiv (\nu x)((\nu y)(P_1^y | P_2^x) | Q^z) = R && \text{[b]} \\
\text{cut } D_2 (x. E) &\equiv F_2 \text{ with} \\
F_2 &\rightsquigarrow R_2 \text{ and } R_2 \equiv (\nu x)(P_2^x | Q^z) \text{ and} \\
F_2 &\Rightarrow F'_2 \text{ with} \\
F'_2 &\rightsquigarrow R_2' \equiv (\nu x)(P_2'^x | Q'^z) && \text{By i.h. on } D_2, E \\
F &\Rightarrow \text{cut } D_1 (y. F'_2) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow (\nu y)(P_1^y | (\nu x)(P_2'^x | Q'^z)) \equiv (\nu x)((\nu y)(P_1^y | P_2'^x) | Q'^z) \\
&= (\nu x)(P'^x | Q'^z) && \text{[d]}
\end{aligned}$$

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

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut} (\text{cut}^! D_1 (u. D_2)) (x. E) \\
&\equiv \text{cut}^! D_1 (u. \text{cut } D_2 (x. E)) = F && \text{By rule (cut/cut}^!\text{/ -) [a]} \\
F &\rightsquigarrow (\nu u)((!u(y). P_1^y) | (\nu x)(P_2^x | Q^z)) \\
&\equiv (\nu x)((\nu u)((!u(y). P_1^y) | P_2^x) | Q^z) = R && \text{By struct. cong. [b]} \\
\text{cut } D_2 (x. E) &\equiv F_2 \text{ with} \\
F_2 &\rightsquigarrow R_2 \text{ and } R_2 \equiv (\nu x)(P_2^x | Q^z) \text{ and} \\
F_2 &\Rightarrow F'_2 \text{ with} \\
F'_2 &\rightsquigarrow R_2' \equiv (\nu x)(P_2'^x | Q'^z) && \text{By i.h. on } D_2, E \\
F &\Rightarrow \text{cut}^! D_1 (u. F'_2) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow (\nu u)((!u(y). P_1^y) | (\nu x)(P_2'^x | Q'^z)) \\
&\equiv (\nu x)((\nu u)((!u(y). P_1^y) | P_2'^x) | Q'^z) = (\nu x)(P'^x | Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $E = 1L y E_1$  and  $D$  arbitrary.

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut } D (x. 1L y E_1) \\
&\equiv 1L y (\text{cut } D (x. E_1)) = F && \text{By rule (cut/-/1L) [a]} \\
F &\rightsquigarrow (\nu x)(P^x | Q^z) = R && \text{[b]} \\
\text{cut } D (x. E_1) &\equiv F_1 \text{ for some } F_1;
\end{aligned}$$

$$\begin{aligned}
F_1 &\rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z); \\
F_1 &\Rightarrow F'_1; \\
F'_1 &\rightsquigarrow R'_1 \equiv (\nu x)(P'^x \mid Q'^z) && \text{By i.h. on } D, E_1 \\
F &= \mathbf{1L} \ y \ F_1 \Rightarrow \mathbf{1L} \ y \ F'_1 = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow R_1^z \equiv (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $E = \mathbf{!L} \ y \ (u. E_1)$  and  $D$  arbitrary.

$$\begin{aligned}
\text{cut } D \ (x. E) &= \text{cut } D \ (x. \mathbf{!L} \ y \ (u. E_1)) \\
&\equiv \mathbf{!L} \ y \ (u. \text{cut } D \ (x. E_1)) = F && \text{By rule (cut/-/!L) [a]} \\
F &\rightsquigarrow (\nu x)(P^x \mid Q^z) = R && \text{since } E_1 \rightsquigarrow Q^z\{u/y\} \text{ [b]} \\
\text{cut } D \ (x. E_1) &\equiv F_1 \text{ for some } F_1; \\
F_1 &\rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z); \\
F_1 &\Rightarrow F'_1; \\
F'_1 &\rightsquigarrow R'_1 \equiv (\nu x)(P'^x \mid Q'^z) && \text{By i.h. on } D, E_1 \\
F &= \mathbf{!L} \ y \ (u. F_1) \Rightarrow \mathbf{1L} \ y \ (u. F'_1) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow R_1^z \equiv (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $E = \text{cut } E_1 \ (y. E_2)$  with  $x \in FV(E_1)$  and  $D$  arbitrary.

$$\begin{aligned}
\text{cut } D \ (x. E) &= \text{cut } D \ (x. \text{cut } E_1 \ (y. E_2)) \\
&\equiv \text{cut} \ (\text{cut } D \ (x. E_1)) \ (y. E_2) = F && \text{By reduction (cut/-/cut}_1\text{) [a]} \\
F &\rightsquigarrow (\nu y)((\nu x)(P^x \mid Q_1^y \mid Q_2^z) \equiv (\nu x)(P^x \mid (\nu y)(Q_1^y \mid Q_2^z)) = R && \text{[b]} \\
\text{cut } D \ (x. E_1) &\equiv F_1 \text{ with} \\
F_1 &\rightsquigarrow R_1 \equiv (\nu x)(P^x \mid Q_1^y) \text{ and} \\
F_1 &\Rightarrow F'_1 \text{ with} \\
F'_1 &\rightsquigarrow R'_1 \equiv (\nu x)(P'^x \mid Q_1'^y) && \text{By i.h. on } D, E_1 \\
F &\Rightarrow \text{cut } F'_1 \ (y. E_2) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow (\nu y)((\nu x)(P'^x \mid Q_1'^y \mid Q_2^z) \equiv (\nu x)(P'^x \mid (\nu y)(Q_1'^y \mid Q_2^z)) \\
&= (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $E = \text{cut } E_1 \ (y. E_2)$  with  $x \in FV(E_2)$  and  $D$  arbitrary.

$$\begin{aligned}
\text{cut } D \ (x. E) &= \text{cut } D \ (x. \text{cut } E_1 \ (y. E_2)) \\
&\equiv \text{cut } E_1 \ (y. \text{cut } D \ (x. E_2)) = F && \text{By reduction (cut/-/cut}_2\text{) [a]} \\
F &\rightsquigarrow (\nu y)(Q_1^y \mid (\nu x)(P^x \mid Q_2^z)) \equiv (\nu x)(P^x \mid (\nu y)(Q_1^y \mid Q_2^z)) = R && \text{[b]} \\
\text{cut } D \ (x. E_2) &\equiv F_2 \text{ with} \\
F_2 &\rightsquigarrow R_2 \equiv (\nu x)(P^x \mid Q_2^z) \text{ and} \\
F_2 &\Rightarrow F'_2 \text{ with} \\
F'_2 &\rightsquigarrow R'_2 \equiv (\nu x)(P'^x \mid Q_2'^z) && \text{By i.h. on } D, E_2 \\
F &\Rightarrow \text{cut } E_1 \ (y. F'_2) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow (\nu y)(Q_1^y \mid (\nu x)(P'^x \mid Q_2'^z)) \equiv (\nu x)(P'^x \mid (\nu y)(Q_1^y \mid Q_2'^z)) \\
&= (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $E = \text{cut}^! E_1 (u. E_2)$  with  $x \in FV(E_1)$  and  $D$  arbitrary. This case is impossible because  $E_1$  contains no free linear variables.

**Case:**  $E = \text{cut}^! E_1 (u. E_2)$  with  $x \in FV(E_2)$  and  $D$  arbitrary.

$$\begin{aligned} \text{cut } D (x. E) &= \text{cut } D (x. \text{cut}^! E_1 (u. E_2)) \\ &\equiv \text{cut}^! E_1 (u. \text{cut } D (x. E_2)) = F \end{aligned} \quad \text{By reduction (cut/-/cut}^!\text{) [a]}$$

$$\begin{aligned} F &\rightsquigarrow (\nu u)((!u(y).Q_1^y) \mid (\nu x)(P^x \mid Q_2^z)) \equiv \\ &(\nu x)(P^x \mid (\nu u)((!u(y).Q_1^y) \mid Q_2^z)) = R \end{aligned} \quad \text{[b]}$$

$$\begin{aligned} \text{cut } D (x. E_2) &\equiv F_2 \text{ with} \\ F_2 &\rightsquigarrow R_2 \equiv (\nu x)(P^x \mid Q_2^z) \text{ and} \\ F_2 &\Rightarrow F'_2 \text{ with} \end{aligned}$$

$$F'_2 \rightsquigarrow R'_2 \equiv (\nu x)(P'^x \mid Q'^z) \quad \text{By i.h. on } D, E_2$$

$$F \Rightarrow \text{cut}^! E_1 (u. F'_2) = F' \quad \text{By congruence [c]}$$

$$\begin{aligned} F' &\rightsquigarrow (\nu u)((!u(y).Q_1^y) \mid (\nu x)(P'^x \mid Q'^z)) \\ &\equiv (\nu x)(P'^x \mid (\nu u)((!u(y).Q_1^y) \mid Q'^z)) = (\nu x)(P'^x \mid Q'^z) \end{aligned} \quad \text{[d]}$$

□

**Lemma A.3.** Assume

- (a)  $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:C_1 \ \& \ C_2$  with  $P \xrightarrow{x.\text{inr}} P'$ ;
- (b)  $\Gamma; \Delta_2, x:C_1 \ \& \ C_2 \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{x.\text{inl}} Q'$ .

Then

- (c)  $\text{cut } D(x. E) \equiv \Rightarrow \equiv F$  for some  $F$ ;
- (d)  $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu x)(P' \mid Q')$ .

*Proof.* Completely symmetric to the previous lemma.

□

**Lemma A.4.** Assume

- (a)  $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:C_1 \ \oplus \ C_2$  with  $P \xrightarrow{x.\text{inl}} P'$ ;
- (b)  $\Gamma; \Delta_2, x:C_1 \ \oplus \ C_2 \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{x.\text{inl}} Q'$ .

Then

- (c)  $\text{cut } D(x. E) \equiv \Rightarrow \equiv F$  for some  $F$ ;
- (d)  $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu x)(P' \mid Q')$ .

*Proof.*

By simultaneous induction on  $D$  and  $E$ . That is, in each appeal to the induction hypothesis either  $D$  becomes smaller and  $E$  remains the same, or  $D$  remains the same and  $E$  becomes smaller.

The possible cases for  $D$  are  $\oplus R_1$ ,  $\mathbf{1L}$ ,  $\mathbf{!L}$ ,  $\text{cut}$ , and  $\text{cut}^!$ . In all other cases  $P^x$  cannot offer  $x.\text{inl}$ . The possible cases for  $E$  are  $\oplus L$ ,  $\mathbf{1L}$ ,  $\mathbf{!L}$ ,  $\text{cut}$ , and  $\text{cut}^!$ .

**Case:**  $D = \oplus R_1 D_1$  and  $E = \oplus L x (x. E_1) (x. E_2)$ .

$$\text{cut } D (x. E) = \text{cut } (\oplus R_1 D_1) (\oplus L x (x. E_1) (x. E_2)) = F \quad \text{[a]}$$

$$F \rightsquigarrow (\nu x)(x.\text{inl}; P_1^x \mid x.\text{case}(Q_1^z, Q_2^z)) = (\nu x)(P^x \mid Q^z) \quad \text{[b]}$$

$$\begin{aligned}
F &\Rightarrow \text{cut } D_1 (x. E_1) = F' && \text{By reduction (cut}/\oplus R_1/\oplus L) \text{ [c]} \\
F' &\rightsquigarrow (\nu x)(P_1^x \mid Q_1^z) \equiv R' \text{ with } P_1^x = P'^x \text{ and } Q_1^z = Q'^z && \text{[d]}
\end{aligned}$$

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

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut } (\mathbf{1L} \ y \ D_1) (x. E) \\
&\equiv \mathbf{1L} \ y \ (\text{cut } D_1 (x. E)) = F && \text{By rule (cut}/\mathbf{1L}/-) \text{ [a]} \\
F &\rightsquigarrow (\nu x)(P^x \mid Q^z) = R && \text{since } D_1 \rightsquigarrow P^x \text{ [b]} \\
\text{cut } D_1 (x. E) &\equiv F_1 \text{ for some } F_1; \\
F_1 &\rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z); \\
F_1 &\Rightarrow F'_1; \\
F'_1 &\rightsquigarrow R'^z_1 \equiv (\nu x)(P'^x \mid Q'^z) && \text{By i.h. on } D_1, E \\
F &= \mathbf{1L} \ y \ F_1 \Rightarrow \mathbf{1L} \ y \ F'_1 = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow R'^z_1 \equiv (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $D = \mathbf{!L} \ y \ (u. D_1)$  and  $E$  arbitrary.

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut } (\mathbf{!L} \ y \ (u. D_1)) (x. E) \\
&\equiv \mathbf{!L} \ y \ (u. \text{cut } D_1 (x. E)) = F && \text{By rule (cut}/\mathbf{!L}/-) \text{ [a]} \\
F &\rightsquigarrow (\nu x)(P^x \mid Q^z) = R && \text{since } D_1 \rightsquigarrow P^x\{u/y\} \text{ [b]} \\
\text{cut } D_1 (x. E) &\equiv F_1 \text{ for some } F_1; \\
F_1 &\rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z); \\
F_1 &\Rightarrow F'_1; \\
F'_1 &\rightsquigarrow R'^z_1 \equiv (\nu x)(P'^x \mid Q'^z) && \text{By i.h. on } D_1, E \\
F &= \mathbf{!L} \ y \ (u. F_1) \Rightarrow \mathbf{!L} \ y \ (u. F'_1) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow R'^z_1 \equiv (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

**Case:**  $D = \text{cut } D_1 (y. D_2)$  and  $E$  arbitrary.

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut } (\text{cut } D_1 (y. D_2)) (x. E) \\
&\equiv \text{cut } D_1 (y. \text{cut } D_2 (x. E)) = F && \text{By rule (cut}/-/ \text{cut}_1) \text{ [a]} \\
F &\rightsquigarrow (\nu y)(P_1^y \mid ((\nu x)(P_2^x \mid Q^z))) \equiv (\nu x)((\nu y)(P_1^y \mid P_2^x) \mid Q^z) = R && \text{[b]} \\
\text{cut } D_2 (x. E) &\equiv F_2 \text{ with} \\
F_2 &\rightsquigarrow R_2 \text{ and } R_2 \equiv (\nu x)(P_2^x \mid Q^z) \text{ and} \\
F_2 &\Rightarrow F'_2 \text{ with} \\
F'_2 &\rightsquigarrow R'^z_2 \equiv (\nu x)(P_2'^x \mid Q'^z) && \text{By i.h. on } D_2, E \\
F &\Rightarrow \text{cut } D_1 (y. F'_2) = F' && \text{By congruence [c]} \\
F' &\rightsquigarrow (\nu y)(P_1^y \mid (\nu x)(P_2'^x \mid Q'^z)) \equiv (\nu x)((\nu y)(P_1^y \mid P_2'^x) \mid Q'^z) \\
&= (\nu x)(P'^x \mid Q'^z) && \text{[d]}
\end{aligned}$$

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

$$\begin{aligned}
\text{cut } D (x. E) &= \text{cut } (\text{cut}^! D_1 (u. D_2)) (x. E) \\
&\equiv \text{cut}^! D_1 (u. \text{cut } D_2 (x. E)) = F && \text{By rule (cut}/\text{cut}^!/-) \text{ [a]} \\
F &\rightsquigarrow (\nu u)((\mathbf{!}u(y).P_1^y) \mid (\nu x)(P_2^x \mid Q^z)) \\
&\equiv (\nu x)((\nu u)((\mathbf{!}u(y).P_1^y) \mid P_2^x) \mid Q^z) = R && \text{By struct. cong. [b]}
\end{aligned}$$

$$\begin{aligned}
& \text{cut } D_2 (x. E) \equiv F_2 \text{ with} \\
& F_2 \rightsquigarrow R_2 \text{ and } R_2 \equiv (\nu x)(P_2^x \mid Q^z) \text{ and} \\
& F_2 \Rightarrow F'_2 \text{ with} \\
& F'_2 \rightsquigarrow R'_2 \equiv (\nu x)(P_2^{x'} \mid Q'^z) \qquad \text{By i.h. on } D_2, E \\
& F \Rightarrow \text{cut}^1 D_1 (u. F'_2) = F' \qquad \text{By congruence [c]} \\
& F' \rightsquigarrow (\nu u)((!u(y).P_1^y) \mid (\nu x)(P_2^{x'} \mid Q'^z)) \\
& \equiv (\nu x)((\nu u)((!u(y).P_1^y) \mid P_2^{x'}) \mid Q'^z) = (\nu x)(P'^x \mid Q'^z) \qquad \text{[d]}
\end{aligned}$$

**Case:**  $E = \mathbf{1L} y E_1$  and  $D$  arbitrary.

$$\begin{aligned}
& \text{cut } D (x. E) = \text{cut } D (x. \mathbf{1L} y E_1) \\
& \equiv \mathbf{1L} y (\text{cut } D (x. E_1)) = F \qquad \text{By rule (cut/-/1L) [a]} \\
& F \rightsquigarrow (\nu x)(P^x \mid Q^z) = R \qquad \text{[b]} \\
& \text{cut } D (x. E_1) \equiv F_1 \text{ for some } F_1; \\
& F_1 \rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z); \\
& F_1 \Rightarrow F'_1; \\
& F'_1 \rightsquigarrow R'_1 \equiv (\nu x)(P'^x \mid Q'^z) \qquad \text{By i.h. on } D, E_1 \\
& F = \mathbf{1L} y F_1 \Rightarrow \mathbf{1L} y F'_1 = F' \qquad \text{By congruence [c]} \\
& F' \rightsquigarrow R'_1^z \equiv (\nu x)(P'^x \mid Q'^z) \qquad \text{[d]}
\end{aligned}$$

**Case:**  $E = !L y (u. E_1)$  and  $D$  arbitrary.

$$\begin{aligned}
& \text{cut } D (x. E) = \text{cut } D (x. !L y (u. E_1)) \\
& \equiv !L y (u. \text{cut } D (x. E_1)) = F \qquad \text{By rule (cut/-/!L) [a]} \\
& F \rightsquigarrow (\nu x)(P^x \mid Q^z) = R \qquad \text{since } E_1 \rightsquigarrow Q^z\{u/y\} \text{ [b]} \\
& \text{cut } D (x. E_1) \equiv F_1 \text{ for some } F_1; \\
& F_1 \rightsquigarrow R_1^z \equiv (\nu x)(P^x \mid Q^z); \\
& F_1 \Rightarrow F'_1; \\
& F'_1 \rightsquigarrow R'_1 \equiv (\nu x)(P'^x \mid Q'^z) \qquad \text{By i.h. on } D, E_1 \\
& F = !L y (u. F_1) \Rightarrow \mathbf{1L} y (u. F'_1) = F' \qquad \text{By congruence [c]} \\
& F' \rightsquigarrow R'_1^z \equiv (\nu x)(P'^x \mid Q'^z) \qquad \text{[d]}
\end{aligned}$$

**Case:**  $E = \text{cut } E_1 (y. E_2)$  with  $x \in FV(E_1)$  and  $D$  arbitrary.

$$\begin{aligned}
& \text{cut } D (x. E) = \text{cut } D (x. \text{cut } E_1 (y. E_2)) \\
& \equiv \text{cut} (\text{cut } D (x. E_1)) (y. E_2) = F \qquad \text{By reduction (cut/-/cut}_1\text{) [a]} \\
& F \rightsquigarrow (\nu y)((\nu x)(P^x \mid Q_1^y) \mid Q_2^z) \equiv (\nu x)(P^x \mid (\nu y)(Q_1^y \mid Q_2^z)) = R \qquad \text{[b]} \\
& \text{cut } D (x. E_1) \equiv F_1 \text{ with} \\
& F_1 \rightsquigarrow R_1 \equiv (\nu x)(P^x \mid Q_1^y) \text{ and} \\
& F_1 \Rightarrow F'_1 \text{ with} \\
& F'_1 \rightsquigarrow R'_1 \equiv (\nu x)(P'^x \mid Q_1^{y'}) \qquad \text{By i.h. on } D, E_1 \\
& F \Rightarrow \text{cut } F'_1 (y. E_2) = F' \qquad \text{By congruence [c]} \\
& F' \rightsquigarrow (\nu y)((\nu x)(P'^x \mid Q_1^{y'}) \mid Q_2^z) \equiv (\nu x)(P'^x \mid (\nu y)(Q_1^{y'} \mid Q_2^z)) \\
& = (\nu x)(P'^x \mid Q'^z) \qquad \text{[d]}
\end{aligned}$$

**Case:**  $E = \text{cut } E_1 (y. E_2)$  with  $x \in FV(E_2)$  and  $D$  arbitrary.

$$\begin{aligned} \text{cut } D (x. E) &= \text{cut } D (x. \text{cut } E_1 (y. E_2)) \\ &\equiv \text{cut } E_1 (y. \text{cut } D (x. E_2)) = F \end{aligned} \quad \text{By reduction (cut/-/cut}_2\text{)} \text{ [a]}$$

$$F \rightsquigarrow (\nu y)(Q_1^y \mid (\nu x)(P^x \mid Q_2^z)) \equiv (\nu x)(P^x \mid (\nu y)(Q_1^y \mid Q_2^z)) = R \quad \text{[b]}$$

$$\text{cut } D (x. E_2) \equiv F_2 \text{ with}$$

$$F_2 \rightsquigarrow R_2 \equiv (\nu x)(P^x \mid Q_2^z) \text{ and}$$

$$F_2 \Rightarrow F'_2 \text{ with}$$

$$F'_2 \rightsquigarrow R'_2 \equiv (\nu x)(P'^x \mid Q_2'^z) \quad \text{By i.h. on } D, E_2$$

$$F \Rightarrow \text{cut } E_1 (y. F'_2) = F' \quad \text{By congruence [c]}$$

$$\begin{aligned} F' \rightsquigarrow (\nu y)(Q_1^y \mid (\nu x)(P'^x \mid Q_2'^z)) &\equiv (\nu x)(P'^x \mid (\nu y)(Q_1^y \mid Q_2'^z)) \\ &= (\nu x)(P'^x \mid Q'^z) \end{aligned} \quad \text{[d]}$$

**Case:**  $E = \text{cut}^! E_1 (u. E_2)$  with  $x \in FV(E_1)$  and  $D$  arbitrary. This case is impossible because  $E_1$  contains no free linear variables.

**Case:**  $E = \text{cut}^! E_1 (u. E_2)$  with  $x \in FV(E_2)$  and  $D$  arbitrary.

$$\begin{aligned} \text{cut } D (x. E) &= \text{cut } D (x. \text{cut}^! E_1 (u. E_2)) \\ &\equiv \text{cut}^! E_1 (u. \text{cut } D (x. E_2)) = F \end{aligned} \quad \text{By reduction (cut/-/cut}^!\text{)} \text{ [a]}$$

$$\begin{aligned} F \rightsquigarrow (\nu u)((!u(y).Q_1^y \mid (\nu x)(P^x \mid Q_2^z)) &\equiv \\ (\nu x)(P^x \mid (\nu u)((!u(y).Q_1^y \mid Q_2^z)) &= R \end{aligned} \quad \text{[b]}$$

$$\text{cut } D (x. E_2) \equiv F_2 \text{ with}$$

$$F_2 \rightsquigarrow R_2 \equiv (\nu x)(P^x \mid Q_2^z) \text{ and}$$

$$F_2 \Rightarrow F'_2 \text{ with}$$

$$F'_2 \rightsquigarrow R'_2 \equiv (\nu x)(P'^x \mid Q_2'^z) \quad \text{By i.h. on } D, E_2$$

$$F \Rightarrow \text{cut}^! E_1 (u. F'_2) = F' \quad \text{By congruence [c]}$$

$$\begin{aligned} F' \rightsquigarrow (\nu u)((!u(y).Q_1^y \mid (\nu x)(P'^x \mid Q_2'^z)) &\equiv \\ \equiv (\nu x)(P'^x \mid (\nu u)((!u(y).Q_1^y \mid Q_2'^z)) &= (\nu x)(P'^x \mid Q'^z) \end{aligned} \quad \text{[d]}$$

□

**Lemma A.5.** Assume

$$(a) \Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:C_1 \oplus C_2 \text{ with } P \xrightarrow{x.\text{inr}} P';$$

$$(b) \Gamma; \Delta_2, x:C_1 \oplus C_2 \vdash E \rightsquigarrow Q :: z:C \text{ with } Q \xrightarrow{x.\text{inr}} Q'.$$

Then

$$(c) \text{cut } D (x. E) \equiv \equiv F \text{ for some } F;$$

$$(d) \Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C \text{ for some } R \equiv (\nu x)(P' \mid Q').$$

*Proof.* Symmetric to the previous lemma. □

**Lemma A.6.** Assume

$$(a) \Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:C_1 \otimes C_2 \text{ with } P \xrightarrow{(\nu y)x(y)} P';$$

$$(b) \Gamma; \Delta_2, x:C_1 \otimes C_2 \vdash E \rightsquigarrow Q :: z:C \text{ with } Q \xrightarrow{x(y)} Q'.$$

Then

- (c)  $\text{cut } D(x.E) \equiv \Rightarrow \equiv F$  for some  $F$ ;  
(d)  $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu x)(P' \mid Q')$ .

*Proof.*

By simultaneous induction on  $D$  and  $E$ . The possible cases for  $D$  are  $\otimes R$ ,  $!L$ ,  $!L$ ,  $\text{cut}$  and  $\text{cut}^!$ . The possible cases for  $E$  are  $\otimes L$ ,  $!L$ ,  $!L$ ,  $\text{cut}$ , and  $\text{cut}^!$ .

**Case:**  $D = \otimes R D_1 D_2$  and  $E = \otimes L x (y.x.E')$ .

$$\begin{aligned} D_1 &\rightsquigarrow P_1^y \text{ and } D_2 \rightsquigarrow P_2^x \text{ where} \\ P^x &= (\nu y)(x\langle y \rangle . (P_1^y \mid P_2^x)) \xrightarrow{(\nu y)x\langle y \rangle} (\nu y)(P_1^y \mid P_2^x) = P'^x && \text{By inversion} \\ E' &\rightsquigarrow Q'^z \text{ where } Q^z = x(y).Q'^z \xrightarrow{x\langle y \rangle} Q'^z && \text{By inversion} \\ \text{cut } D(x.E) &= \text{cut}(\otimes R D_1 D_2)(\otimes L x (y.x.E')) \\ \Rightarrow \text{cut } D_1(y.\text{cut } D_2(x.E')) &= F && \text{By rule (cut}/\otimes R/\otimes L) \end{aligned}$$

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

**Case:**  $D = !L n D'$  and  $E$  arbitrary.

$$\begin{aligned} D' &\rightsquigarrow P && \text{By inversion} \\ \text{cut } D(x.E) &= \text{cut}(!L n D')(x.E) \\ &\equiv !L n (\text{cut } D'(x.E)) && \text{By rule (cut}/!L/-) \\ \Rightarrow !L n F' &= F \text{ for some } F' \rightsquigarrow (\nu x)(P' \mid Q') && \text{By i.h. on } D' \text{ and } E \\ F &\rightsquigarrow (\nu x)(P' \mid Q') \end{aligned}$$

**Case:**  $D = !L n (u.D')$  and  $E$  arbitrary.

$$\begin{aligned} D' &\rightsquigarrow P && \text{By inversion} \\ \text{cut } D(x.E) &= \text{cut}(!L n (u.D'))(x.E) \\ &\equiv !L n (u.\text{cut } D'(x.E)) && \text{By rule (cut}/!L/-) \\ \Rightarrow !L n (u.F') &= F \text{ for some } F' \rightsquigarrow (\nu x)(P' \mid Q') && \text{By i.h. on } D' \text{ and } E \\ F &\rightsquigarrow (\nu x)(P' \mid Q') \end{aligned}$$

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

$$\begin{aligned} D_1 &\rightsquigarrow P_1^n \text{ and } D_2 \rightsquigarrow P_2^x \text{ and } P_2^x \xrightarrow{(\nu y)x\langle y \rangle} P_2'^x \text{ where} \\ P &= (\nu n)(P_1^n \mid P_2^x) \xrightarrow{(\nu y)x\langle y \rangle} (\nu n)(P_1^n \mid P_2'^x) = P'^x && \text{By inversion} \\ \text{cut } D(x.E) &= \text{cut}(\text{cut } D_1 (n.D_2))(x.E) \\ &\equiv \text{cut } D_1 (n.\text{cut } D_2 (x.E)) && \text{By rule (cut}/\text{cut}/-) \\ \Rightarrow \text{cut } D_1 (n.F_2) &= F \text{ where } F_2 \rightsquigarrow (\nu x)(P_2^x \mid Q'^z) && \text{By i.h. on } D_2 \text{ and } E \\ F &\rightsquigarrow (\nu n)(P_1^n \mid (\nu x)(P_2^x \mid Q'^z)) \\ &\equiv (\nu x)((\nu n)(P_1^n \mid P_2^x) \mid Q'^z) \\ &= (\nu x)(P'^x \mid Q'^z) \end{aligned}$$

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

$$\begin{aligned} D_1 &\rightsquigarrow (!u(n).P_1^n) \text{ and } D_2 \rightsquigarrow P_2^x \text{ and } P_2^x \xrightarrow{(\nu y)x\langle y \rangle} P_2'^x \text{ where} \\ P &= (\nu u)((!u(n).P_1^n) \mid P_2^x) \xrightarrow{(\nu y)x\langle y \rangle} (\nu u)((!u(n).P_1^n) \mid P_2'^x) = P'^x && \text{By inversion} \end{aligned}$$

$$\begin{aligned}
& \text{cut } D(x.E) = \text{cut}(\text{cut}^! D_1(u.D_2))(x.E) \\
& \equiv \text{cut}^! D_1(u.\text{cut } D_2(x.E)) && \text{By rule (cut/cut}^!/-) \\
& \Rightarrow \text{cut } D_1(u.F_2) = F \text{ where } F_2 \rightsquigarrow (\nu x)(P_2^{!x} \mid Q^{!z}) && \text{By i.h. on } D_2 \text{ and } E \\
& F \rightsquigarrow (\nu u)((!u(n).P_1^n) \mid (\nu x)(P_2^{!x} \mid Q^{!z})) \\
& \equiv (\nu x)((\nu u)((!u(n).P_1^n) \mid P_2^{!x}) \mid Q^{!z}) \\
& = (\nu x)(P^{!x} \mid Q^{!z})
\end{aligned}$$

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

$$\begin{aligned}
& E' \rightsquigarrow Q^z && \text{By inversion} \\
& \text{cut } D(x.E) = \text{cut } D(x.\mathbf{1L} \ n \ E') \\
& \equiv \mathbf{1L} \ n \ (\text{cut } D(x.E')) && \text{By rule (cut/-/1L)} \\
& \Rightarrow \mathbf{1L} \ n \ F' = F \text{ for some } F' \rightsquigarrow (\nu x)(P^{!x} \mid Q^{!z}) && \text{By i.h. on } D \text{ and } E' \\
& F \rightsquigarrow (\nu x)(P^{!x} \mid Q^{!z})
\end{aligned}$$

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

$$\begin{aligned}
& E' \rightsquigarrow Q^z && \text{By inversion} \\
& \text{cut } D(x.E) = \text{cut } D(x.\mathbf{!L} \ n \ (u.E')) \\
& \equiv \mathbf{!L} \ n \ (u.\text{cut } D(x.E')) && \text{By rule (cut/-/!L)} \\
& \Rightarrow \mathbf{!L} \ n \ (u.F') = F \text{ for some } F' \rightsquigarrow (\nu x)(P^{!x} \mid Q^{!z}) && \text{By i.h. on } D \text{ and } E' \\
& F \rightsquigarrow (\nu x)(P^{!x} \mid Q^{!z})
\end{aligned}$$

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

$$\begin{aligned}
& E_1 \rightsquigarrow Q_1^n \text{ and } E_2 \rightsquigarrow Q_2^z \text{ for } Q_1^n \xrightarrow{x(y)} Q_1'^n \text{ where} \\
& Q = (\nu n)(Q_1^n \mid Q_2^z) \xrightarrow{x(y)} (\nu n)(Q_1'^n \mid Q_2^z) && \text{By inversion} \\
& \text{cut } D(x.E) = \text{cut } D(x.\text{cut } E_1(n.E_2)) \\
& = \text{cut}(\text{cut } D(x.E_1))(n.E_2) && \text{By rule (cut/cut/-) and } x \notin FV(E_2) \\
& \Rightarrow \text{cut } F_1(n.E_2) = F \text{ for some } F_1 \rightsquigarrow (\nu x)(P^{!x} \mid Q_1'^n) && \text{By i.h. on } D \text{ and } E_1 \\
& F \rightsquigarrow (\nu n)((\nu x)(P^{!x} \mid Q_1'^n) \mid Q_2^z) \\
& \equiv (\nu x)(P^{!x} \mid (\nu n)(Q_1'^n \mid Q_2^z)) \\
& = (\nu x)(P^{!x} \mid Q^{!z})
\end{aligned}$$

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

$$\begin{aligned}
& E_1 \rightsquigarrow Q_1^n \text{ and } E_2 \rightsquigarrow Q_2^z \text{ for } Q_2^z \xrightarrow{x(y)} Q_2'^z \text{ where} \\
& Q = (\nu n)(Q_1^n \mid Q_2^z) \xrightarrow{x(y)} (\nu n)(Q_1^n \mid Q_2'^z) && \text{By inversion} \\
& \text{cut } D(x.E) = \text{cut } D(x.\text{cut } E_1(n.E_2)) \\
& = \text{cut } E_1(n.\text{cut } D(x.E_2)) && \text{By rule (cut/-/cut) and } x \notin FV(E_1) \\
& \Rightarrow \text{cut } E_1(n.F_2) = F \text{ for some } F_2 \rightsquigarrow (\nu x)(P^{!x} \mid Q_2'^z) && \text{By i.h. on } D \text{ and } E_2 \\
& F \rightsquigarrow (\nu n)(Q_1^n \mid (\nu x)(P^{!x} \mid Q_2'^z)) \\
& \equiv (\nu x)(P^{!x} \mid (\nu n)(Q_1^n \mid Q_2'^z)) \\
& = (\nu x)(P^{!x} \mid Q^{!z})
\end{aligned}$$

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

$$\begin{aligned}
& x \in FV(E_2) && \text{By inversion} \\
& E_1 \rightsquigarrow (!u(n).Q_1^n) \text{ and } E_2 \rightsquigarrow Q_2^z \text{ for } Q_2^z \xrightarrow{x(y)} Q_2'^z \text{ where}
\end{aligned}$$

$$\begin{aligned}
Q &= (\nu u)((!u(n).Q_1^n \mid Q_2^z) \xrightarrow{x(y)} (\nu u)((!u(n).Q_1^n \mid Q_2^z)) && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } D(x.\text{cut}^! E_1(u.E_2)) \\
&= \text{cut}^! E_1(u.\text{cut } D(x.E_2)) && \text{By rule (cut/-/cut}^!) \\
&\Rightarrow \text{cut}^! E_1(u.F_2) = F \text{ for some } F_2 \rightsquigarrow (\nu x)(P^{!x} \mid Q_2^z) && \text{By i.h. on } D \text{ and } E_2 \\
F &\rightsquigarrow (\nu u)((!u(n).Q_1^n \mid (\nu x)(P^{!x} \mid Q_2^z)) \\
&\equiv (\nu x)(P^{!x} \mid (\nu u)((!u(n).Q_1^n \mid Q_2^z)) \\
&= (\nu x)(P^{!x} \mid Q^{!z})
\end{aligned}$$

□

**Lemma A.7.** Assume

- (a)  $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:C_1 \multimap C_2$  with  $P \xrightarrow{x(y)} P'$ ;  
(b)  $\Gamma; \Delta_2, x:C_1 \multimap C_2 \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{(\nu y)x(y)} Q'$ .

Then

- (c)  $\text{cut } D(x.E) \equiv \equiv F$  for some  $F$ ;  
(d)  $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu x)(\nu y)(P' \mid Q')$ .

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

**Case:**  $D = \multimap R(x.D_1)$  and  $E = \multimap L x E_1(x.E_2)$ .

$$\begin{aligned}
E_1 &\rightsquigarrow Q_1^y \text{ and } E_2 \rightsquigarrow Q_2^z \text{ where} \\
Q^z &= (\nu y)(x(y).(Q_1^y \mid Q_2^z)) \xrightarrow{(\nu y)x(y)} (\nu y)(Q_1^y \mid Q_2^z) = Q'^x && \text{By inversion} \\
D_1 &\rightsquigarrow P^{!x} \text{ where } P^x = x(y).P^{!x} \xrightarrow{x(y)} P^{!x} && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } (\multimap R(x.D_1)) (\multimap L x E_1(x.E_2)) \\
&\Rightarrow \text{cut } E_1(y.\text{cut } D_1(x.E_2)) = F && \text{By rule (cut}/\multimap R/\multimap L) \\
F &\rightsquigarrow (\nu y)(Q_1^y \mid (\nu x)(P_1^x \mid Q_2^z)) \equiv (\nu x)((\nu y)P_1^x \mid (Q_1^y \mid Q_2^z)) = (\nu x)(P^{!x} \mid Q'^z)
\end{aligned}$$

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

$$\begin{aligned}
D' &\rightsquigarrow P && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } (\mathbf{1L} n D')(x.E) \\
&\equiv \mathbf{1L} n (\text{cut } D'(x.E)) && \text{By rule (cut}/\mathbf{1L}/-) \\
&\Rightarrow \mathbf{1L} n F' = F \text{ for some } F' \rightsquigarrow (\nu x)(P' \mid Q') && \text{By i.h. on } D' \text{ and } E \\
F &\rightsquigarrow (\nu x)(P' \mid Q')
\end{aligned}$$

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

$$\begin{aligned}
D' &\rightsquigarrow P && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } (\mathbf{!L} n(u.D'))(x.E) \\
&\equiv \mathbf{!L} n(u.\text{cut } D'(x.E)) && \text{By rule (cut}/\mathbf{!L}/-) \\
&\Rightarrow \mathbf{!L} n(u.F') = F \text{ for some } F' \rightsquigarrow (\nu x)(P' \mid Q') && \text{By i.h. on } D' \text{ and } E \\
F &\rightsquigarrow (\nu x)(P' \mid Q')
\end{aligned}$$

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

$$\begin{aligned}
D_1 &\rightsquigarrow P_1^n \text{ and } D_2 \rightsquigarrow P_2^x \text{ and } P_2^x \xrightarrow{(\nu y)x(y)} P_2^{x'} \text{ where} \\
P &= (\nu n)(P_1^n | P_2^x) \xrightarrow{(\nu y)x(y)} (\nu n)(P_1^n | P_2^{x'}) = P^{x'} && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut}(\text{cut } D_1(n.D_2))(x.E) \\
&\equiv \text{cut } D_1(n.\text{cut } D_2(x.E)) && \text{By rule (cut/cut/-)} \\
&\Rightarrow \text{cut } D_1(n.F_2) = F \text{ where } F_2 \rightsquigarrow (\nu x)(P_2^{x'} | Q^{z'}) && \text{By i.h. on } D_2 \text{ and } E \\
F &\rightsquigarrow (\nu n)(P_1^n | (\nu x)(P_2^{x'} | Q^{z'})) \\
&\equiv (\nu x)((\nu n)(P_1^n | P_2^{x'}) | Q^{z'}) \\
&= (\nu x)(P^{x'} | Q^{z'})
\end{aligned}$$

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

$$\begin{aligned}
D_1 &\rightsquigarrow (!u(n).P_1^n) \text{ and } D_2 \rightsquigarrow P_2^x \text{ and } P_2^x \xrightarrow{(\nu y)x(y)} P_2^{x'} \text{ where} \\
P &= (\nu u)((!u(n).P_1^n) | P_2^x) \xrightarrow{(\nu y)x(y)} (\nu u)((!u(n).P_1^n) | P_2^{x'}) = P^{x'} && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut}^!(\text{cut}^! D_1(u.D_2))(x.E) \\
&\equiv \text{cut}^! D_1(u.\text{cut}^! D_2(x.E)) && \text{By rule (cut/cut^!/-)} \\
&\Rightarrow \text{cut}^! D_1(u.F_2) = F \text{ where } F_2 \rightsquigarrow (\nu x)(P_2^{x'} | Q^{z'}) && \text{By i.h. on } D_2 \text{ and } E \\
F &\rightsquigarrow (\nu u)((!u(n).P_1^n) | (\nu x)(P_2^{x'} | Q^{z'})) \\
&\equiv (\nu x)((\nu u)((!u(n).P_1^n) | P_2^{x'}) | Q^{z'}) \\
&= (\nu x)(P^{x'} | Q^{z'})
\end{aligned}$$

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

$$\begin{aligned}
E' &\rightsquigarrow Q^z && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } D(x.\mathbf{1L} n E') \\
&\equiv \mathbf{1L} n (\text{cut } D(x.E')) && \text{By rule (cut/-/1L)} \\
&\Rightarrow \mathbf{1L} n F' = F \text{ for some } F' \rightsquigarrow (\nu x)(P^{x'} | Q^{z'}) && \text{By i.h. on } D \text{ and } E' \\
F &\rightsquigarrow (\nu x)(P^{x'} | Q^{z'})
\end{aligned}$$

**Case:**  $E = !L n (u.E')$  and  $D$  arbitrary.

$$\begin{aligned}
E' &\rightsquigarrow Q^z && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } D(x.!L n (u.E')) \\
&\equiv !L n (u.\text{cut } D(x.E')) && \text{By rule (cut/-/!L)} \\
&\Rightarrow !L n (u.F') = F \text{ for some } F' \rightsquigarrow (\nu x)(P^{x'} | Q^{z'}) && \text{By i.h. on } D \text{ and } E' \\
F &\rightsquigarrow (\nu x)(P^{x'} | Q^{z'})
\end{aligned}$$

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

$$\begin{aligned}
E_1 &\rightsquigarrow Q_1^n \text{ and } E_2 \rightsquigarrow Q_2^z \text{ for } Q_1^n \xrightarrow{x(y)} Q_1^{n'} \text{ where} \\
Q &= (\nu n)(Q_1^n | Q_2^z) \xrightarrow{x(y)} (\nu n)(Q_1^{n'} | Q_2^z) && \text{By inversion} \\
\text{cut } D(x.E) &= \text{cut } D(x.\text{cut } E_1(n.E_2)) \\
&= \text{cut}(\text{cut } D(x.E_1))(n.E_2) && \text{By rule (cut/cut/-) and } x \notin FV(E_2) \\
&\Rightarrow \text{cut } F_1(n.E_2) = F \text{ for some } F_1 \rightsquigarrow (\nu x)(P^{x'} | Q_1^{n'}) && \text{By i.h. on } D \text{ and } E_1 \\
F &\rightsquigarrow (\nu n)((\nu x)(P^{x'} | Q_1^{n'}) | Q_2^z) \\
&\equiv (\nu x)(P^{x'} | (\nu n)(Q_1^{n'} | Q_2^z)) \\
&= (\nu x)(P^{x'} | Q^{z'})
\end{aligned}$$

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

$$\begin{aligned}
& E_1 \rightsquigarrow Q_1^n \text{ and } E_2 \rightsquigarrow Q_2^z \text{ for } Q_2^z \xrightarrow{x(y)} Q_2'^z \text{ where} \\
& Q = (\nu n)(Q_1^n \mid Q_2^z) \xrightarrow{x(y)} (\nu n)(Q_1^n \mid Q_2'^z) && \text{By inversion} \\
& \text{cut } D(x. E) = \text{cut } D(x. \text{cut } E_1 (n. E_2)) \\
& = \text{cut } E_1 (n. \text{cut } D(x. E_2)) && \text{By rule (cut/-/cut) and } x \notin FV(E_1) \\
& \Rightarrow \text{cut } E_1 (n. F_2) = F \text{ for some } F_2 \rightsquigarrow (\nu x)(P'^x \mid Q_2'^z) && \text{By i.h. on } D \text{ and } E_2 \\
& F \rightsquigarrow (\nu n)(Q_1^n \mid (\nu x)(P'^x \mid Q_2'^z)) \\
& \equiv (\nu x)(P'^x \mid (\nu n)(Q_1^n \mid Q_2'^z)) \\
& = (\nu x)(P'^x \mid Q'^z)
\end{aligned}$$

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

$$\begin{aligned}
& x \in FV(E_2) && \text{By inversion} \\
& E_1 \rightsquigarrow (!u(n).Q_1^n) \text{ and } E_2 \rightsquigarrow Q_2^z \text{ for } Q_2^z \xrightarrow{x(y)} Q_2'^z \text{ where} \\
& Q = (\nu u)((!u(n).Q_1^n) \mid Q_2^z) \xrightarrow{x(y)} (\nu u)((!u(n).Q_1^n) \mid Q_2'^z) && \text{By inversion} \\
& \text{cut } D(x. E) = \text{cut } D(x. \text{cut}^! E_1 (u. E_2)) \\
& = \text{cut}^! E_1 (u. \text{cut } D(x. E_2)) && \text{By rule (cut/-/cut}^!) \\
& \Rightarrow \text{cut}^! E_1 (u. F_2) = F \text{ for some } F_2 \rightsquigarrow (\nu x)(P'^x \mid Q_2'^z) && \text{By i.h. on } D \text{ and } E_2 \\
& F \rightsquigarrow (\nu u)((!u(n).Q_1^n) \mid (\nu x)(P'^x \mid Q_2'^z)) \\
& \equiv (\nu x)(P'^x \mid (\nu u)((!u(n).Q_1^n) \mid Q_2'^z)) \\
& = (\nu x)(P'^x \mid Q'^z)
\end{aligned}$$

□

**Lemma A.8.** Assume

- (a)  $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:A$  with  $P \xrightarrow{x(y)} P'$ ;
- (b)  $\Gamma; \Delta_2, x:!A \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{(\nu y)x(y)} Q'$ .

Then

- (c)  $\text{cut } D(x. E) \equiv \Rightarrow \equiv F$  for some  $F$ ;
- (d)  $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu x)(\nu y)(P' \mid Q')$ .

*Proof.* By simultaneous induction on the structure of  $D, E$ . There are only five possible cases for  $D$ :  $!R D'_1$ ,  $\mathbf{1L} n D'_1$ ,  $!L n (u. D_1)$ ,  $\text{cut } D'_1 (n. D'_1)$ , and  $\text{cut}^! D'_1 (u. D'_1)$ . In all other cases  $P$  cannot offer  $x(y)$ , which follows by analysis of the typed extraction rules and the definition of  $\xrightarrow{\alpha}$ . There are only four possible cases for  $E$ :  $!L n (u. E'_2)$ ,  $\mathbf{1L} n E'_2$ ,  $\text{cut } E'_2 (n. E'_2)$ , and  $\text{cut}^! E'_2 (u. E'_2)$ . In all other cases  $Q$  cannot offer  $(\nu y)x(y)$ , which follows by analysis of the typed extraction rules and the definition of  $\xrightarrow{\alpha}$ .

**Case:**  $D = !R D'_1$ ,  $E = !L x (u. E'_2)$ .

$$\begin{aligned}
& \Delta_1 = (\cdot) \\
& \Gamma; \cdot \vdash D'_1 \rightsquigarrow R_1 :: u:A \text{ where } P = !x(u).R_1 \\
& \Gamma, u : A; \Delta_2 \vdash E' \rightsquigarrow Q_1 :: z:C \text{ where } Q = (\nu y)x(y).Q_1 && \text{by inversion} \\
& \text{cut } D(x. E) = && \text{this case} \\
& \text{cut } (!R D'_1) (x. !L x (u. E'_2))
\end{aligned}$$

$\equiv \text{cut}^! D'_1 (u. E'_2)$  by (cut/!R/!L)  
 $\text{cut}^! D'_1 (u. E'_2) \equiv \text{cut}^! D'_1 (u. E^*)$  for some  $E^*$   
 $\Gamma, u : A; \Delta_2 \vdash E^* \rightsquigarrow R' :: z:C$  with  $R' \equiv (\nu y)(R_1\{y/u\} \mid Q'_1)$  by Lemma !Red  
 Pick  $F = \text{cut}^! D'_1 (u. E^*)$  [satisfying (c)]  
 $\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C$  by cut  
 with  $R = (\nu x)(!x(u).R_1 \mid (\nu y)(R_1\{y/u\} \mid Q'_1))$   
 $\equiv (\nu y)(\nu x)(!x(u).P_1 \mid R_1\{y/u\} \mid Q'_1)$   
 $\equiv (\nu y)(\nu x)(P'_1 \mid Q'_1)$  since  $P'_1 \equiv !x(u).R_1 \mid R_1\{y/u\}$  [satisfying (d)]

**Case:**  $D$  arbitrary,  $E = !L y (u. E'_2)$ .

$\Delta_2 = (\Delta^*, y : !B)$   
 $\Gamma, u : B; \Delta^*, x : !A \vdash E'_2 \rightsquigarrow Q_1 :: z:C$  by inversion  
 $\text{cut } D (x. E'_2) \equiv E^*$  for some  $E^*$   
 $\Gamma, u : B; \Delta^* \vdash E^* \rightsquigarrow R' :: z:C$  with  $R' \equiv (\nu y)(\nu x)(P'_1 \mid Q'_1)$  by i.h.  
 $\text{cut } D (x. E) = \text{cut } D (x. !L y (u. E'_2))$   
 $\equiv !L y (u. \text{cut } D (x. E'_2))$  by (cut/-/!L)  
 $\equiv !L y (u. E^*)$  by congruence  
 Pick  $F = !L y (u. E^*)$  [satisfying (c)]  
 $\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C$  by !L  
 with  $R = R'$  [satisfying (d)]

**Case:**  $D = !L y (u. D'_1)$ ,  $E$  arbitrary.

$\Delta_1 = (\Delta^*, y : !B)$   
 $\Gamma, u : B; \Delta^* \vdash D'_1 \rightsquigarrow P_1 :: x:!A$  by inversion  
 $\text{cut } D'_1 (x. E) \equiv D^*$  for some  $D^*$   
 $\Gamma, u : B; \Delta^* \vdash D^* \rightsquigarrow R' :: z:C$  with  $R' \equiv (\nu y)(\nu x)(P'_1 \mid Q'_1)$  by i.h.  
 $\text{cut } D (x. E) = \text{cut } (!L y (u. D'_1)) (x. E)$   
 $\equiv !L y (u. \text{cut } D'_1 (x. E))$  by (cut/!L/-)  
 $\equiv !L y (u. D^*)$  by congruence  
 Pick  $F = !L y (u. D^*)$  [satisfying (c)]  
 $\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C$  by !L  
 with  $R = R'$  [satisfying (d)]

**Case:**  $D$  arbitrary,  $E = \mathbf{1L} n E'_2$ .

$\Delta_2 = (\Delta^*, n : \mathbf{1})$   
 $\Gamma, x : A; \Delta^* \vdash E'_2 \rightsquigarrow Q_1 :: z:C$  by inversion  
 $\text{cut } D (x. E'_2) \equiv E^*$  for some  $E^*$   
 $\Gamma; \Delta^* \vdash E^* \rightsquigarrow R' :: z:C$  with  $R' \equiv (\nu y)(\nu x)(P'_1 \mid Q'_1)$  by i.h.  
 $\text{cut } D (x. E) = \text{cut } D (x. \mathbf{1L} n E'_2)$   
 $\equiv \mathbf{1L} n (\text{cut } D (x. E'_2))$  by (cut/-/!L)  
 $\equiv \mathbf{1L} n E^*$  by congruence  
 Pick  $F = \mathbf{1L} n E^*$  [satisfying (c)]  
 $\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C$  by !L  
 with  $R = R'$  [satisfying (d)]

**Case:**  $D = \mathbf{1L} \ n \ D'_1$ ,  $E$  arbitrary.

$$\begin{array}{l}
\Delta_1 = (\Delta^*, n : \mathbf{1}) \\
\Gamma; \Delta^* \vdash D'_1 \rightsquigarrow P_1 :: x:A \qquad \text{by inversion} \\
\text{cut } D'_1 (x.E) \equiv \Rightarrow \equiv D^* \text{ for some } D^* \\
\Gamma; \Delta^* \vdash D^* \rightsquigarrow R' :: z:C \text{ with } R' \equiv (\nu y)(\nu x)(P'_1 \mid Q'_1) \qquad \text{by i.h.} \\
\text{cut } D (x.E) = \text{cut } (\mathbf{1L} \ n. D'_1) (x.E) \\
\equiv \mathbf{1L} \ n. (\text{cut } D'_1 (x.E)) \qquad \text{by (cut/1L/-)} \\
\equiv \Rightarrow \equiv \mathbf{1L} \ n \ D^* \qquad \text{by congruence} \\
\text{Pick } F = \mathbf{1L} \ n \ D^* \qquad \text{[satisfying (c)]} \\
\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C \qquad \text{by } \mathbf{1L} \\
\text{with } R = R' \qquad \text{[satisfying (d)]}
\end{array}$$

**Case:**  $D$  arbitrary,  $E = \text{cut } E'_2 (n. E''_2)$ .

$$\begin{array}{l}
\Delta_2 = (\Delta'_2, \Delta''_2) \\
Q_1 = (\nu n)(R_1 \mid R_2) \\
\Gamma, x : A; \Delta'_2 \vdash E'_2 \rightsquigarrow R_1 :: n:B \\
\Gamma, x : A; \Delta''_2, n : B \vdash E''_2 \rightsquigarrow R_2 :: z:C \qquad \text{by inversion} \\
\text{Subcase: } R_1 \xrightarrow{(\nu y)x(y)} R'_1 \text{ and } Q'_1 = (\nu n)(R'_1 \mid R_2) \\
\text{cut } D (x.E'_2) \equiv \Rightarrow \equiv D^* \text{ for some } D^* \\
\Gamma; \Delta'_2 \vdash D^* \rightsquigarrow R' :: n:B \text{ with } R' \equiv (\nu y)(\nu x)(P'_1 \mid R'_1) \qquad \text{by i.h.} \\
\text{cut } D (x.E) = \text{cut } D (x.\text{cut } E'_2 (n. E''_2)) \\
\equiv \text{cut } (\text{cut } D (x.E'_2)) (n. E''_2) \qquad \text{by (cut/ - /cut}_1\text{)} \\
\equiv \Rightarrow \equiv \text{cut } D^* (n. E''_2) \qquad \text{by congruence} \\
\text{Pick } F = \text{cut } D^* (n. E''_2) \qquad \text{[satisfying (c)]} \\
\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C \qquad \text{by cut} \\
\text{with } R = (\nu n)(R' \mid R_2) \\
\equiv (\nu n)((\nu y)(\nu x)(P'_1 \mid R'_1) \mid R_2) \\
\equiv (\nu n)(\nu y)(\nu x)(P'_1 \mid R'_1 \mid R_2) \\
\equiv (\nu y)(\nu x)(P'_1 \mid (\nu n)(R'_1 \mid R_2)) \\
\equiv (\nu y)(\nu x)(P'_1 \mid Q'_1) \qquad \text{[satisfying (d)]} \\
\text{Subcase: } R_2 \xrightarrow{(\nu y)x(y)} R'_2 \text{ and } Q'_1 = (\nu n)(R_1 \mid R'_2) \\
\text{cut } D (x.E''_2) \equiv \Rightarrow \equiv D^* \text{ for some } D^* \\
\Gamma; \Delta''_2, n : B \vdash D^* \rightsquigarrow R' :: z:C \text{ with } R' \equiv (\nu y)(\nu x)(P'_1 \mid R'_2) \qquad \text{by i.h.} \\
\text{cut } D_1 (x.D_2) = \text{cut } D (x.\text{cut } E'_2 (n. E''_2)) \\
\equiv \text{cut } E'_2 (n.\text{cut } D (x.E''_2)) \qquad \text{by (cut/ - /cut}_2\text{)} \\
\equiv \Rightarrow \equiv \text{cut } E'_2 (n. D^*) \qquad \text{by congruence} \\
\text{Pick } F = \text{cut } E'_2 (n. D^*) \qquad \text{[satisfying (c)]} \\
\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C \qquad \text{by cut} \\
\text{with } R = (\nu n)(R_1 \mid R') \\
\equiv (\nu n)(R_1 \mid (\nu y)(\nu x)(P'_1 \mid R'_2)) \\
\equiv (\nu y)(\nu x)(\nu n)(R_1 \mid P'_1 \mid R'_2) \\
\equiv (\nu y)(\nu x)(\nu n)(P'_1 \mid R_1 \mid R'_2) \\
\equiv (\nu y)(\nu x)(P'_1 \mid (\nu n)(R_1 \mid R'_2))
\end{array}$$

$$\equiv (\nu y)(\nu x)(P'_1 \mid Q'_1) \quad [\text{satisfying (d)}]$$

**Case:**  $D = \text{cut } D'_1 (n. D''_1)$ ,  $E$  arbitrary.

$$\begin{aligned} \Delta_1 &= (\Delta'_1, \Delta''_1) \\ P_1 &= (\nu n)(R_1 \mid R_2) \\ \Gamma; \Delta'_1 \vdash D'_1 \rightsquigarrow R_1 &:: n:B \\ \Gamma; \Delta''_1, n:B \vdash D''_1 \rightsquigarrow R_2 &:: x:!A && \text{by inversion} \\ R_2 \xrightarrow{x(y)} R'_2 \text{ and } P'_1 &= (\nu n)(R_1 \mid R'_2) \\ \text{cut } D''_1 (x. E) &\equiv \equiv D^* \text{ for some } D^* \\ \Gamma; \Delta''_1, n:B, \Delta_2 \vdash D^* \rightsquigarrow R' &:: z:C \text{ with } R' \equiv (\nu y)(\nu x)(R'_2 \mid Q'_1) && \text{by i.h.} \\ \text{cut } D (x. E) &= \text{cut} (\text{cut } D'_1 (n. D''_1)) (x. E) \\ &\equiv \text{cut } D'_1 (n. \text{cut } D''_1 (x. E)) && \text{by (cut/ - /cut}_1) \\ &\equiv \equiv \text{cut } D'_1 (n. D^*) && \text{by congruence} \\ \text{Pick } F &= \text{cut } D'_1 (n. D^*) && [\text{satisfying (c)}] \\ \Gamma; \Delta \vdash F \rightsquigarrow R &:: z:C && \text{by cut} \\ \text{with } R &= (\nu n)(R_1 \mid R') \\ &\equiv (\nu n)(R_1 \mid (\nu y)(\nu x)(R'_2 \mid Q'_1)) \\ &\equiv (\nu y)(\nu x)(\nu n)(R_1 \mid R'_2 \mid Q'_1) \\ &\equiv (\nu y)(\nu x)(\nu n)(R_1 \mid R'_2 \mid Q_1) \\ &\equiv (\nu y)(\nu x)(P'_1 \mid Q'_1) && [\text{satisfying (d)}] \end{aligned}$$

**Case:**  $D$  arbitrary,  $E = \text{cut}^! E'_2 (u. E''_2)$ .

$$\begin{aligned} Q_1 &= (\nu u)(!u(w).R_1 \mid R_2) \\ \Gamma; \cdot \vdash E'_2 \rightsquigarrow R_1 &:: w:B \\ \Gamma, u : B; \Delta_2, x : !A \vdash E''_2 \rightsquigarrow R_2 &:: z:C && \text{by inversion} \\ R_2 \xrightarrow{(\nu y)x(y)} R'_2 \text{ and } Q'_1 &= (\nu u)(!u(w).R_1 \mid R'_2) \\ \text{cut } D (x. E''_2) &\equiv \equiv D^* \text{ for some } D^* \\ \Gamma, u : B; \Delta_2 \vdash D^* \rightsquigarrow R' &:: z:C \text{ with } R' \equiv (\nu y)(\nu x)(P'_1 \mid R'_2) && \text{by i.h.} \\ \text{cut } D (x. E) &= \text{cut } D (x. \text{cut}^! E'_2 (u. E''_2)) \\ &\equiv \text{cut}^! E'_2 (u. \text{cut } D (x. E''_2)) && \text{by (cut/ - /cut}^!) \\ &\equiv \equiv \text{cut}^! E'_2 (u. D^*) && \text{by congruence} \\ \text{Pick } F &= \text{cut}^! E'_2 (u. D^*) && [\text{satisfying (c)}] \\ \Gamma; \Delta \vdash F \rightsquigarrow R &:: z:C && \text{by cut}^! \\ \text{with } R &= (\nu u)(!u(w).R_1 \mid R') \\ &\equiv (\nu u)(!u(w).R_1 \mid (\nu y)(\nu x)(P'_1 \mid R'_2)) \\ &\equiv (\nu y)(\nu x)(\nu u)(!u(w).R_1 \mid P'_1 \mid R'_2) \\ &\equiv (\nu y)(\nu x)(\nu u)(P'_1 \mid !u(w).R_1 \mid R'_2) \\ &\equiv (\nu y)(\nu x)(P'_1 \mid (\nu u)(!u(w).R_1 \mid R'_2)) \\ &\equiv (\nu y)(\nu x)(P'_1 \mid Q'_1) && [\text{satisfying (d)}] \end{aligned}$$

**Case:**  $D_1 = \text{cut}^! D'_1 (u. D''_1)$ ,  $E$  arbitrary.

$$P_1 = (\nu u)(!u(w).R_1 \mid R_2)$$

$$\begin{array}{l}
\Gamma \vdash D'_1 \rightsquigarrow R_1 :: w:B \\
\Gamma, u : B; \Delta_1 \vdash D''_1 \rightsquigarrow R_2 :: x:!A \quad \text{by inversion} \\
R_2 \xrightarrow{x(y)} R'_2 \text{ and } P'_1 = (\nu u)(!u(w).R_1 \mid R'_2) \\
\text{cut } D''_1(x.E) \equiv \equiv D^* \text{ for some } D^* \\
\Gamma, u : B; \Delta \vdash D^* \rightsquigarrow R' :: z:C \text{ with } R' \equiv (\nu y)(\nu x)(R'_2 \mid Q'_1) \quad \text{by i.h.} \\
\text{cut } D_1(x.E) = \text{cut}^! D'_1(u.D''_1)(x.E) \\
\equiv \text{cut}^! D'_1(u.\text{cut } D''_1(x.E)) \quad \text{by (cut/cut}^!/-) \\
\equiv \equiv \text{cut}^! D'_1(u.D^*) \quad \text{by congruence} \\
\text{Pick } F = \text{cut}^! D'_1(u.D^*) \quad \text{[satisfying (c)]} \\
\Gamma; \Delta \vdash F \rightsquigarrow R :: z:C \quad \text{by cut}^! \\
\text{with } R = (\nu u)(!u(w).R_1 \mid (\nu y)(\nu x)(R'_2 \mid Q'_1)) \\
\equiv (\nu y)(\nu x)(\nu u)(!u(w).R_1 \mid R'_2 \mid Q'_1) \\
\equiv (\nu y)(\nu x)((\nu u)(!u(w).R_1 \mid R'_2) \mid Q'_1) \\
\equiv (\nu y)(\nu x)(P'_1 \mid Q'_1) \quad \text{[satisfying (d)]}
\end{array}$$

□

**Lemma A.9.** Assume

- (a)  $\Gamma; \cdot \vdash D \rightsquigarrow P :: u:A$  and  
(b)  $\Gamma, u:A; \Delta \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{(\nu y)u\langle y \rangle} Q'$ .

Then

- (c)  $\text{cut}^! D_1(u.D_2) \equiv \equiv \text{cut}^! D_1(u.F)$  for some  $F$ ;  
(d)  $\Gamma, u:A; \Delta \vdash F \rightsquigarrow R :: z:C$  with  $R \equiv (\nu y)(P\{y/u\} \mid Q')$ .

*Proof.* By induction on the structure of  $E$ . There are only five possible cases for  $E$ :  $\mathbf{1L} \ n \ E'_2$ ,  $\text{copy } u \ (x.E'_2)$ ,  $\mathbf{!L} \ n \ (u.E'_2)$ ,  $\text{cut } E'_2 \ (n.E''_2)$ , and  $\text{cut}^! E'_2 \ (v.E''_2)$ . In all other cases  $Q_1$  cannot offer  $(\nu y)u\langle y \rangle$ , which follows by analysis of the typed extraction rules and the definition of  $\xrightarrow{\alpha}$ .

**Case:**  $E = \text{copy } u \ (x.E'_2)$ .

$$\begin{array}{l}
Q_1 \equiv (\nu y)u\langle y \rangle.Q'_1 \\
\Gamma, u : A; \Delta, y : A \vdash E'_2 \rightsquigarrow Q'_1 :: z : C \quad \text{by inversion} \\
\text{cut}^! D(u.(\text{copy } u \ (y.E'_2))) \\
\Rightarrow \text{cut } D(y.\text{cut}^! D(u.E'_2)) \quad \text{by (cut}^!/-/\text{copy)} \\
\equiv \text{cut}^! D(u.\text{cut } D(y.E'_2)) \quad \text{by (cut/-/cut}^!) \\
\text{Pick } F = \text{cut } D(y.E'_2) \quad \text{[satisfying (c)]} \\
\Gamma, u : A; \Delta \vdash F \rightsquigarrow R :: z:C \text{ with } R \equiv (\nu y)(P_1\{y/u\} \mid Q'_1) \quad \text{[satisfying (d)]}
\end{array}$$

**Case:**  $E = \mathbf{1L} \ n \ E'_2$ .

$$\begin{array}{l}
\Delta = (\Delta^*, n : \mathbf{1}) \\
\Gamma, u:A; \Delta^* \vdash E'_2 \rightsquigarrow Q_1 :: z:C \text{ with } Q_1 \xrightarrow{(\nu y)u\langle y \rangle} Q'_1 \quad \text{By inversion} \\
\text{cut}^! D(u.E'_2) \equiv \equiv \text{cut}^! D(u.D^*) \text{ for some } D^*
\end{array}$$

$\Gamma; \Delta^* \vdash D^* \rightsquigarrow R' :: z:C$  with  $R' \equiv (\nu y)(P_1\{y/u\} \mid Q'_1)$  By i.h.

$\text{cut}^! D(u. \mathbf{1L} n E'_2)$   
 $\equiv \mathbf{1L} n (\text{cut}^! D(u. E'_2))$  by (cut<sup>!</sup>/ - / $\mathbf{1L}$ )  
 $\equiv \Rightarrow \equiv \mathbf{1L} n (\text{cut}^! D(u. D^*))$  By congruence  
 $\equiv \text{cut}^! D(u. \mathbf{1L} n D^*)$  by (cut<sup>!</sup>/ - / $\mathbf{1L}$ )  
 Pick  $F = \mathbf{1L} n D^*$  [satisfying (c)]  
 Pick  $R = R'$   
 $\Gamma, u : A; \Delta \vdash F \rightsquigarrow R :: z:C$  with  $R \equiv (\nu y)(P_1\{y/u\} \mid Q'_1)$  By  $\mathbf{1L}$  [satisfying (d)]

**Case:**  $E = !\mathbf{L} x (v. E'_2)$ .

$\Delta = (\Delta^*, x : !B)$   
 $\Gamma, v:B, u:A; \Delta^* \vdash E'_2 \rightsquigarrow Q_1 :: z:C$  with  $Q_1 \xrightarrow{(\nu y)u\langle y \rangle} Q'_1$  By inversion

$\text{cut}^! D(u. E'_2) \equiv \Rightarrow \equiv \text{cut}^! D(u. D^*)$  for some  $D^*$   
 $\Gamma, v:B; \Delta^* \vdash D^* \rightsquigarrow R' :: z:C$  with  $R' \equiv (\nu y)(P_1\{y/u\} \mid Q'_1)$  By i.h.

$\text{cut}^! D(u. (!\mathbf{L} x (v. E'_2)))$   
 $\equiv !\mathbf{L} x (v. \text{cut}^! D(u. E'_2))$  by (cut<sup>!</sup>/ - /! $\mathbf{L}$ )  
 $\equiv \Rightarrow \equiv !\mathbf{L} x (v. \text{cut}^! D(u. D^*))$  By congruence  
 $\equiv \text{cut}^! D(u. !\mathbf{L} x (v. D^*))$  by (cut<sup>!</sup>/ - /! $\mathbf{L}$ )  
 Pick  $F = !\mathbf{L} x (v. D^*)$  [satisfying (a)]  
 Pick  $R = R'$   
 $\Gamma, u : A; \Delta \vdash F \rightsquigarrow R :: z:C$  with  $R \equiv (\nu y)(P_1\{y/u\} \mid Q'_1)$  By ! $\mathbf{L}$  [satisfying (b)]

**Case:**  $E = \text{cut} E'_2 (n. E''_2)$ .

$\Delta = (\Delta_1, \Delta_2)$   
 $\Gamma, x : A; \Delta_1 \vdash E'_2 \rightsquigarrow R_1 :: n:B$   
 $\Gamma, x : A; \Delta_2, n:B \vdash E''_2 \rightsquigarrow R_2 :: z:C$  By inversion  
 $Q_1 = (\nu n)(R_1 \mid R_2)$

Either  $R_1 \xrightarrow{(\nu y)x\langle y \rangle} R'_1$  and  $Q'_1 \equiv (\nu n)(R'_1 \mid R_2)$ , or  
 $R_2 \xrightarrow{(\nu y)x\langle y \rangle} R'_2$  and  $Q'_1 \equiv (\nu n)(R_1 \mid R'_2)$

**Subcase:**  $R_1 \xrightarrow{(\nu y)x\langle y \rangle} R'_1$   
 $\text{cut}^! D(u. E'_2) \equiv \Rightarrow \equiv \text{cut}^! D(u. D^*)$  for some  $D^*$   
 $\Gamma, u : A; \Delta_1 \vdash D^* \rightsquigarrow S :: n:B$   
 with  $S \equiv (\nu y)(P_1\{y/u\} \mid R'_1)$  By i.h.

$\text{cut}^! D(u. \text{cut} E'_2 (n. E''_2))$   
 $\equiv \text{cut} (\text{cut}^! D(u. E'_2)) (n. \text{cut}^! D(u. E''_2))$  by (cut<sup>!</sup>/ - /cut)  
 $\equiv \text{cut} (\text{cut}^! D(u. D^*)) (n. \text{cut}^! D(u. E''_2))$  By congruence  
 $\equiv \text{cut}^! D(u. \text{cut} D^* (n. E''_2))$  by (cut<sup>!</sup>/ - /cut)

Pick  $D = \text{cut } D^* (n. D''_2)$  [satisfying (c)]  
 Pick  $R = (\nu n)(S \mid R_2)$   
 $\Gamma, u : A; \Delta \vdash D \rightsquigarrow R :: z:C$  By cut  
 with  $R = (\nu n)((\nu y)(P_1\{y/u\} \mid R'_1) \mid R_2)$   
 $\equiv (\nu y)(P_1\{y/u\} \mid (\nu n)(R'_1 \mid R_2))$   
 $\equiv (\nu y)(P_1\{y/u\} \mid Q'_1)$  [satisfying (d)]  
**Subcase:**  $R_2 \xrightarrow{(\nu y)x(y)} R'_2$   
 $\text{cut}^! D (u. E''_2) \equiv \equiv \text{cut}^! D (u. D^*)$  for some  $D^*$   
 $\Gamma, u : A; \Delta_2, n : B \vdash D^* \rightsquigarrow S :: z:C$   
 with  $S \equiv (\nu y)(P_1\{y/u\} \mid R'_2)$  By i.h.

$\text{cut}^! D (u. \text{cut } E'_2 (n. E''_2))$   
 $\equiv \text{cut} (\text{cut}^! D (u. E''_2)) (n. \text{cut}^! D (u. E''_2))$  By (cut<sup>!</sup>/ – /cut)  
 $\equiv \text{cut} (\text{cut}^! D (u. E''_2)) (n. \text{cut}^! D (u. D^*))$  By congruence  
 $\equiv \text{cut}^! D (u. \text{cut } E'_2 (n. D^*))$  by (cut<sup>!</sup>/ – /cut)  
 Pick  $F = \text{cut } E'_2 (n. D^*)$  [satisfying (c)]  
 Pick  $R = (\nu n)(R_1 \mid S)$   
 $\Gamma, u : A; \Delta \vdash D \rightsquigarrow R :: z:C$  By cut  
 with  $R = (\nu n)(R_1 \mid (\nu y)(P_1\{y/u\} \mid R'_2))$   
 $\equiv (\nu y)(P_1\{y/u\} \mid (\nu n)(R_1 \mid R'_2))$   
 $\equiv (\nu y)(P_1\{y/u\} \mid Q'_1)$  [satisfying (d)]

**Case:**  $E = \text{cut}^! E'_2 (v. E''_2)$ .

$\Gamma, u : A; \vdash E'_2 \rightsquigarrow R_1 :: w:B$   
 $\Gamma, v : B, u : A; \Delta_2, \vdash E''_2 \rightsquigarrow R_2 :: z:C$  By inversion  
 $Q_1 = (\nu v)(!v(w).R_1 \mid R_2)$   
 $R_2 \xrightarrow{(\nu y)x(y)} R'_2$  and  $Q'_1 \equiv (\nu v)(!v(w).R_1 \mid R'_2)$   
 $\text{cut}^! D (v. E''_2) \equiv \equiv \text{cut}^! D (v. D^*)$  for some  $D^*$   
 $\Gamma, v : B, u : A; \Delta_2 \vdash D^* \rightsquigarrow S :: z:C$   
 with  $S \equiv (\nu y)(P_1\{y/u\} \mid R'_2)$  By i.h.  
 $\text{cut}^! D (u. \text{cut}^! E'_2 (v. E''_2))$   
 $\equiv \text{cut}^! (\text{cut}^! D (u. E''_2)) (v. \text{cut}^! D (u. E''_2))$  by (cut<sup>!</sup>/ – /cut<sup>!</sup>)  
 $\equiv \text{cut}^! (\text{cut}^! D (u. E''_2)) (v. \text{cut}^! D (u. D^*))$  By congruence  
 $\equiv \text{cut}^! D (u. \text{cut}^! E'_2 (v. D^*))$  by (cut<sup>!</sup>/ – /cut<sup>!</sup>)  
 Pick  $F = \text{cut}^! E'_2 (v. D^*)$  [satisfying (c)]  
 $\Gamma, u : A; \Delta \vdash F \rightsquigarrow R :: z:C$  By cut  
 with  $R = (\nu v)(!v(w).R_1 \mid S)$   
 $\equiv (\nu v)(!v(w).R_1 \mid (\nu y)(P_1\{y/u\} \mid R'_2))$   
 $\equiv (\nu y)(\nu v)(!v(w).R_1 \mid P_1\{y/u\} \mid R'_2)$   
 $\equiv (\nu y)(\nu v)(P_1\{y/u\} \mid !v(w).R_1 \mid R'_2)$   
 $\equiv (\nu y)(P_1\{y/u\} \mid (\nu v)(!v(w).R_1 \mid R'_2))$

$$\equiv (\nu y)(P_1\{y/u\} \mid Q'_1) \quad \text{[satisfying (d)]}$$

□

**Lemma A.10.** Assume

(a)  $\Gamma; \cdot \vdash D \rightsquigarrow P :: u:A$  and

(b)  $\Gamma, u:A; \Delta_2 \vdash E \rightsquigarrow Q :: z:C$  with  $Q \xrightarrow{(\nu y)u(y)} Q'$ .

Then

(c)  $\text{cut}^! D (u. E) \equiv \equiv F$  for some  $F$  and

(d)  $\Gamma; \Delta \vdash F \rightsquigarrow R :: z : C$  for some  $R \equiv (\nu u)(!u(x).P \mid (\nu y)(P\{y/u\} \mid Q'))$ .

*Proof.* Directly from Lemma A.9 and the typed extraction rule for  $\text{cut}^!$ . □

**Theorem A.1.** Assume  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:A$  and  $P \rightarrow Q$ .

Then there is  $E$  such that  $D \equiv \equiv E$  and  $\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$

*Proof.* By induction on the structure of  $D$ . The possible cases for  $D$  are **1L**, **!L**,  $\text{cut}^!$  and  $\text{cut}^!$ . In all other cases  $P$  cannot offer  $\tau$ .

**Case:**  $D = \mathbf{1L} \ n \ D'$ .

$$\Delta = (\Delta^*, n : \mathbf{1})$$

$$\Gamma; \Delta^* \vdash D' \rightsquigarrow P :: z:A$$

by inversion

$$\Gamma; \Delta^* \vdash E' \rightsquigarrow Q :: z:A \text{ for some } E' \text{ with } D' \equiv \equiv E'$$

by i.h.

$$\text{Pick } E = \mathbf{1L} \ n \ E'.$$

$$D \equiv \equiv E$$

by congruence

$$\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$$

by **1L**

**Case:**  $D = \mathbf{!L} \ x \ (u. D')$ .

$$\Delta = (\Delta^*, x : \mathbf{!}B)$$

$$\Gamma, u : B; \Delta^* \vdash D' \rightsquigarrow P :: z:A$$

by inversion

$$\Gamma, u : B; \Delta^* \vdash E' \rightsquigarrow Q :: z:A \text{ for some } E' \text{ with } D' \equiv \equiv E'$$

by i.h.

$$\text{Pick } E = \mathbf{!L} \ x \ (u. E').$$

$$D \equiv \equiv E$$

by congruence

$$\Gamma : \Delta \vdash E \rightsquigarrow Q :: z:A$$

by **1L**

**Case:**  $D = \text{cut}^! D_1 (u. D_2)$ .

$$P \equiv (\nu u)(!u(w).P_1 \mid P_2)$$

$$\Gamma; \vdash D_1 \rightsquigarrow P_1 :: u:C$$

$$\Gamma, u : C; \Delta \vdash D_2 \rightsquigarrow P_2 :: z:A$$

by inversion

From  $P \rightarrow Q$  either

$$(1) P_2 \rightarrow Q_2 \text{ and } Q = (\nu u)(!u(w).P_1 \mid Q_2)$$

$$(2) P_2 \xrightarrow{(\nu y)u(y)} Q_2 \text{ and } Q = (\nu u)(!u(w).P_1 \mid (\nu y)(P_1\{y/u\} \mid Q_2))$$

**Subcase (1):**

$$\Gamma, u : C; \Delta \vdash D \rightsquigarrow Q_2 :: z:A \text{ for some } E' \text{ with } D_2 \equiv \equiv E'$$

by i.h.

$$\text{cut}^! D_1 (u. D_2) \equiv \equiv \text{cut}^! D_1 (u. E')$$

$$\text{Pick } E = \text{cut}^! D_1 (u. E')$$

$\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$  by cut<sup>!</sup>

**Subcase (2):**

cut<sup>!</sup>  $D_1 (u. D_2) \equiv \equiv E$  for some  $E$

$\Gamma; \Delta \vdash E \rightsquigarrow R :: z:A$  with  $R \equiv Q$  by Corollary A.10

**Case:**  $D = \text{cut } D_1 (x. D_2)$ .

$P \equiv (\nu x)(P_1 \mid P_2)$

$\Delta = (\Delta_1, \Delta_2)$

$\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow P_1 :: x:C$

$\Gamma; \Delta_2, x : C \vdash D_2 \rightsquigarrow P_1 :: z:A$

by inversion

Since  $P \rightarrow Q$  there are four subcases:

(1)  $P_1 \rightarrow Q_1$  and  $Q = (\nu x)(Q_1 \mid P_2)$

(2)  $P_2 \rightarrow Q_2$  and  $Q = (\nu x)(P_1 \mid Q_2)$

(3)  $P_1 \xrightarrow{\alpha} Q_1$  and  $P_2 \xrightarrow{\bar{\alpha}} Q_2$

(4)  $P_1 \xrightarrow{\bar{\alpha}} Q_1$  and  $P_2 \xrightarrow{\alpha} Q_2$

**Subcase (1):**  $P_1 \rightarrow Q_1$

$D_1 \equiv \equiv E_1$  for some  $E_1$

$\Gamma; \Delta_1 \vdash E_1 \rightsquigarrow Q_1 :: x:C$

by i.h.

$D = \text{cut } D_1 (x. D_2)$

$\equiv \equiv \text{cut } E_1 (x. D_2)$

by congruence

Pick  $E = \text{cut } E_1 (x. D_2)$

$\Gamma; \Delta \vdash E \rightsquigarrow Q :: z:A$

**Subcase (2):**  $P_2 \rightarrow Q_2$

Symmetric to Subcase (1).

**Subcase (3):**  $P_1 \xrightarrow{\alpha} Q_1$  and  $P_2 \xrightarrow{\bar{\alpha}} Q_2$

**Subsubcase:**  $C = \mathbf{1}$

not possible

**Subsubcase:**  $C = C_1 \& C_2$

$\alpha = x.\text{inl}$  or  $\alpha = x.\text{inr}$

By Lemma A.1

cut  $D_1 (x. D_2) \equiv \equiv D$  for some  $D$

$\Gamma; \Delta \vdash D \rightsquigarrow R :: z : C$

with  $R \equiv (\nu x)(Q_1 \mid Q_2) = Q$

by Lemmas A.2 and A.3

**Subsubcase:**  $C = C_1 \oplus C_2$

not possible

**Subsubcase:**  $C = C_1 \otimes C_2$

not possible

**Subsubcase:**  $C = C_1 \multimap C_2$

$\alpha = x(y)$  and  $\bar{\alpha} = (\nu y)x\langle y \rangle$

By Lemma A.1

cut  $D_1 (x. D_2) \equiv \equiv D$  for some  $D$

$\Gamma; \Delta \vdash D \rightsquigarrow Q :: z : C$

with  $R \equiv (\nu x)(\nu y)(Q_1 \mid Q_2) = Q$

by Lemma A.7

**Subsubcase:**  $C = !C_1$

$\alpha = x(y)$  and  $\bar{\alpha} = \overline{(\nu y)x\langle y \rangle}$

By Lemma A.1

cut  $D_1 (x. D_2) \equiv \Rightarrow \equiv D$  for some  $D$

$\Gamma; \Delta \vdash D \rightsquigarrow Q :: z : C$

with  $R \equiv (\nu x)(\nu y)(Q_1 \mid Q_2) = Q$

by Lemma A.8

**Subcase (4):**  $P_1 \xrightarrow{\bar{\alpha}} Q_1$  and  $P_2 \xrightarrow{\alpha} Q_2$

**Subsubcase:**  $C = \mathbf{1}$

not possible

**Subsubcase:**  $C = C_1 \& C_2$

not possible

**Subsubcase:**  $C = C_1 \oplus C_2$

$\bar{\alpha} = \overline{x.\text{inl}}$  or  $\bar{\alpha} = \overline{x.\text{inr}}$

By Lemma A.1

cut  $D_1 (x. D_2) \equiv \Rightarrow \equiv D$  for some  $D$

$\Gamma; \Delta \vdash D \rightsquigarrow R :: z : C$

with  $R \equiv (\nu x)(Q_1 \mid Q_2) = Q$

by Lemmas A.4 and A.5

**Subsubcase:**  $C = C_1 \otimes C_2$

$\bar{\alpha} = \overline{(\nu y)x\langle y \rangle}$  and  $\alpha = x(y)$

By Lemma A.1

cut  $D_1 (x. D_2) \equiv \Rightarrow \equiv D$  for some  $D$

$\Gamma; \Delta \vdash D \rightsquigarrow Q :: z : C$

with  $R \equiv (\nu x)(\nu y)(Q_1 \mid Q_2) = Q$

**Subsubcase:**  $C = C_1 \multimap C_2$

not possible

**Subsubcase:**  $C = !C_1$

not possible

□

### A.3. Progress Lemmas

**Lemma A.11.** Assume  $\Gamma, \Delta \vdash D \rightsquigarrow P :: z : C$  and not  $\text{live}(P)$ ; then

- 1  $C = \mathbf{1}$  or  $C = !C'$  for some  $C'$ .
- 2  $(x_i : A_i) \in \Delta$  implies  $A_i = \mathbf{1}$  or there is  $B_j$  with  $A_i = !B_j$ ;
- 3  $C = !C'$  implies  $P \equiv (\nu \bar{x})(!z(y).R \mid R')$ .

*Proof.* By structural induction on the structure of  $D$ . The only possible cases for  $D$  are  $\mathbf{1R}$ ,  $\mathbf{1L}$   $D'$ ,  $!L$   $x (u. D')$ , cut  $D' (x. D'')$ ,  $\mathbf{R!}$   $D'$ , and cut<sup>!</sup>  $D' (x. D'')$ , which follows by analysis of the typing rules.

**Case:**  $D = \text{cut } D' (x. D'')$

$\Delta = (\Delta_1, D_2)$

$P \equiv (\nu x)(P_1 \mid P_2)$

$\Gamma; \Delta_1 \vdash D' \rightsquigarrow P_1 :: x:A$

$\Gamma; \Delta_2, x : A \vdash D'' \rightsquigarrow P_2 :: z:C$

not  $\text{live}(P_1)$  and not  $\text{live}(P_2)$

by inversion  
Since not  $\text{live}(P_2)$

$C = \mathbf{1}$  or  $C = !C'$  for some  $C'$  [satisfying (1)]  
 $(x_i : A_i) \in (\Delta_2, x : A)$  implies  $A_i = \mathbf{1}$  or there is  $B_j$  with  $A_i = !B_j$   
 $C = !C'$  implies  $P_2 \equiv (\nu\bar{n})(!z(y).P'_2 \mid P''_2)$  by i.h.  
 $A = \mathbf{1}$  or  $A = !A'$  for some  $A'$   
 $(x_i : A_i) \in \Delta_1$  implies  $A_i = \mathbf{1}$  or there is  $B_j$  with  $A_i = !B_j$   
 $A = !A'$  implies  $P_1 \equiv (\nu\bar{m})(!x(y).P'_1 \mid P''_1)$  by i.h.  
 $(x_i : A_i) \in (\Delta_1, \Delta_2) = \Delta$  implies  $A_i = \mathbf{1}$   
or there is  $B_j$  with  $A_i = !B_j$  [satisfying (2)]  
 $C = !C'$  implies  $P \equiv (\nu x)(P_1 \mid P_2)$   
 $\equiv (\nu x)((\nu\bar{m})(!x(y).P'_1 \mid P''_1) \mid (\nu\bar{n})(!z(y).P'_2 \mid P''_2))$   
 $\equiv (\nu x)(\nu\bar{m})(\nu\bar{n})(!x(y).P'_1 \mid P''_1 \mid !z(y).P'_2 \mid P''_2)$   
 $\equiv (\nu x)(\nu\bar{m})(\nu\bar{n})(z(y).R \mid R')$   
with  $R = P'_2$  and  $R' = !x(y).P'_1 \mid P''_1 \mid !z(y).P'_2 \mid P''_2$  [satisfying (3)]

**Case:**  $D = \text{cut}^! D' (u. D'')$

$P \equiv (\nu x)(!x(y).P_1 \mid P_2)$   
 $\Gamma; \vdash D' \rightsquigarrow P_1 :: y:A$   
 $\Gamma, u : A; \Delta \vdash D'' \rightsquigarrow P_2 :: z:C$  by inversion  
not *live*( $P_2$ ) Since not *live*( $P_2$ )  
 $C = \mathbf{1}$  or  $C = !C'$  for some  $C'$  [satisfying (1)]  
 $(x_i : A_i) \in \Delta$  implies  $A_i = \mathbf{1}$  or there is  $B_j$  with  $A_i = !B_j$  [satisfying (2)]  
 $C = !C'$  implies  $P_2 \equiv (\nu\bar{n})(!z(y).P'_2 \mid P''_2)$  by i.h.  
 $C = !C'$  implies  $P \equiv (\nu x)(!x(y).P_1 \mid P_2)$   
 $\equiv (\nu x)(!x(y).P_1 \mid (\nu\bar{n})(!z(y).P'_2 \mid P''_2))$   
 $\equiv (\nu x)(\nu\bar{n})(!x(y).P_1 \mid !z(y).P'_2 \mid P''_2)$   
 $\equiv (\nu x)(\nu\bar{n})(z(y).R \mid R')$   
with  $R = P'_2$  and  $R' = !x(y).P_1 \mid P_2$  [satisfying (3)]

□

**Lemma A.12.** Let  $\Gamma; \Delta \vdash D \rightsquigarrow P :: z : C$ . If *live*( $P$ ) then there is  $Q$  such that one of the following holds:

- (a)  $P \rightarrow Q$ ,
- (b)  $P \xrightarrow{\alpha} Q$  for some  $\alpha$  where  $s(\alpha) \in z, \Gamma, \Delta$  and  $s(\alpha) \in \Gamma, \Delta$  if  $C = !A$ .

*Proof.* By induction on the structure of  $D$ . All cases are possible for  $D$  except  $\mathbf{1R}$  and  $\mathbf{!R} D'$ , which follows by analysis of the typing rules.

**Case:**  $D = \mathbf{1L} n D'$

$\Delta = (\Delta^*, n : \mathbf{1})$   
 $\Gamma; \Delta^* \vdash D' \rightsquigarrow P :: z:C$  by inversion  
There is  $Q$  such that either  $P \rightarrow Q$ , or  $P \xrightarrow{\alpha} Q$   
for some  $\alpha$  with  $s(\alpha) \in z, \Gamma, \Delta^*$  and  $s(\alpha) \in \Gamma, \Delta^*$  if  $C = !A$ . by i.h.

There is  $Q$  such that either  $P \rightarrow Q$ , or  $P \xrightarrow{\alpha} Q$   
for some  $\alpha$  with  $s(\alpha) \in z, \Gamma, \Delta$  and  $s(\alpha) \in \Gamma, \Delta$  if  $C = !A$ .

**Case:**  $D = !L n (u. D')$

$\Delta = (n : !A, \Delta^*)$

$\Gamma, u : A; \Delta^* \vdash D' \rightsquigarrow P :: z:C$

by inversion

There is  $Q$  such that either  $P \rightarrow Q$ , or  $P \xrightarrow{\alpha} Q$

for some  $\alpha$  with  $s(\alpha) \in z, \Gamma, u, \Delta^*$  and  $s(\alpha) \in \Gamma, u, \Delta^*$  if  $C = !A$ .

by i.h.

There is  $Q$  such that either  $P \rightarrow Q$ , or  $P \xrightarrow{\alpha} Q$

for some  $\alpha$  with  $s(\alpha) \in z, \Gamma, \Delta$  and  $s(\alpha) \in \Gamma, \Delta$  if  $C = !A$ .

**Case:**  $D = \otimes R D_1 D_2$

$\Delta = (\Delta_1, \Delta_2), C = C_1 \otimes C_2$ .

$\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow Q :: y:C_1$

$\Gamma; \Delta_2 \vdash D'_2 \rightsquigarrow R :: z:C_2$

$P \equiv (\nu y)z\langle y \rangle.(Q \mid R)$

by inversion

$P \xrightarrow{(\nu y)z\langle y \rangle} Q$  with  $z \in z, \Gamma, \Delta$  and  $C \neq !A$ .

**Case:**  $D = \otimes L (y. D_1)$

$\Delta = (D^*, x : C_1 \otimes C_2)$

$\Gamma; \Delta^*, y : C_1, x : C_2 \vdash D_1 \rightsquigarrow Q :: z:C$

$P \equiv x(y).Q$

by inversion

$P \xrightarrow{x(y)} Q$  with  $x \in \Gamma, \Delta$

**Case:**  $D = \multimap R D_1$

$C = C_1 \multimap C_2$ .

$\Gamma; \Delta, y : C_1 \vdash D_1 \rightsquigarrow Q :: z:C_2$

$P \equiv z(y).Q$

by inversion

$P \xrightarrow{z(y)} Q$  with  $z \in z, \Gamma, \Delta$  and  $C \neq !A$ .

**Case:**  $D = \multimap L D_1 D_2$

$\Delta = (\Delta_1, \Delta_2, x : C_1 \multimap C_2)$

$\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow Q :: y:C_1$

$\Gamma; \Delta_2, x : C_2 \vdash D'_2 \rightsquigarrow R :: z:C_2$

$P \equiv (\nu y)x\langle y \rangle.(Q \mid R)$

by inversion

$P \xrightarrow{(\nu y)x\langle y \rangle} Q$  with  $x \in \Gamma, \Delta$

**Case:**  $D = \text{cut } D_1 (x. D_2)$

$\Delta = (\Delta_1, \Delta_2)$

$\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow P_1 :: x:A$

$\Gamma; \Delta_2, x : A \vdash D_2 \rightsquigarrow P_2 :: z:C$

$P \equiv (\nu x)(P_1 \mid P_2)$

by inversion

$\text{live}(P_1)$  or  $\text{live}(P_2)$

since  $\text{live}(P)$

Case (1):  $\text{live}(P_1)$  and  $\text{live}(P_2)$ .

There is  $P'_1$  such that either  $P_1 \rightarrow P'_1$ , or  $P_1 \xrightarrow{\alpha} P'_1$

for some  $\alpha_1$  with  $s(\alpha_1) \in x, \Gamma, \Delta_1$  and  $s(\alpha_1) \in \Gamma, \Delta_1$  if  $A = !B_1$ .

There is  $P'_2$  such that either  $P_2 \rightarrow P'_2$ , or  $P_2 \xrightarrow{\alpha_2} P'_2$

for some  $\alpha_2$  with  $s(\alpha_2) \in x, \Gamma, \Delta_2, z$  and  $s(\alpha_2) \in x, \Gamma, \Delta_2$  if  $C = !B_2$ . by i.h.

Subcase (0.1):  $P_1 \rightarrow P'_1$  or  $P_2 \rightarrow P'_2$

$P \rightarrow Q$  [satisfying (a)]

Subcase (1.1):  $s(\alpha_1) \neq x$

$P \xrightarrow{\alpha_1} Q \equiv (\nu x)(P'_1 \mid P_2)$  with  $\alpha_1 \in \Gamma, \Delta$  [satisfying (b)]

Subcase (1.2):  $s(\alpha_2) \neq x$

$P \xrightarrow{\alpha_2} Q \equiv (\nu x)(P_1 \mid P'_2)$  with  $\alpha_2 \in z, \Gamma, \Delta$  [satisfying (b)]

Subcase (1.3):  $s(\alpha_1) = s(\alpha_2) = x$

$\alpha_2 = \bar{\alpha}_1$  By Lemma A.1

$P \rightarrow Q$  with  $Q \equiv (\nu x)(\nu y)(P'_1 \mid P'_2)$  or  $Q \equiv (\nu x)(P'_1 \mid P'_2)$  [satisfying (a)]

Case (2): not  $live(P_1)$  and  $live(P_2)$

There is  $P'_2$  such that either  $P_2 \rightarrow P'_2$ , or  $P_2 \xrightarrow{\alpha_2} P'_2$

for some  $\alpha_2$  with  $s(\alpha_2) \in x, \Gamma, \Delta_2, z$  and  $s(\alpha_2) \in x, \Gamma, \Delta_2$  if  $C = !B_2$  by i.h.

Subcase (2.1):  $P_2 \rightarrow P'_2$

$P \rightarrow Q$  with  $Q \equiv (\nu x)(P_1 \mid P'_2)$  [satisfying (a)]

Subcase (2.2):  $P_2 \xrightarrow{\alpha_2} P'_2$

Subcase (2.2.1):  $s(\alpha_2) \neq x$

$P \xrightarrow{\alpha_2} Q$  with  $Q \equiv (\nu x)(P_1 \mid P'_2)$  [satisfying (b)]

Subcase (2.2.2):  $s(\alpha_2) = x$

$A \neq \mathbf{1}$  By Lemma A.1

$P_1 \equiv (\nu \bar{y})(!x(w).R'_1 \mid R''_1)$

$A = !B$  By Lemma A.11

$P_1 \xrightarrow{x(y)} P_1$

$\alpha_2 = (\nu y)x(w)$  By Lemma A.1

$P \rightarrow Q$  with  $Q \equiv (\nu x)(\nu y)(P_1 \mid P'_2)$  [satisfying (a)]

Case (3):  $live(P_1)$  and not  $live(P_2)$

There is  $P'_1$  such that either  $P_1 \rightarrow P'_1$ , or  $P_1 \xrightarrow{\alpha_1} P'_1$

for some  $\alpha_1$  with  $s(\alpha_1) \in \Gamma, \Delta_1, x$  and  $s(\alpha_1) \in \Gamma, \Delta_1$  if  $C = !B_2$  by i.h.

Subcase (3.1):  $P_1 \rightarrow P'_1$

$P \rightarrow Q$  with  $Q \equiv (\nu x)(P'_1 \mid P_2)$  [satisfying (a)]

Subcase (3.1):  $P_1 \xrightarrow{\alpha_1} P'_1$

for some  $\alpha_1$  with  $s(\alpha_1) \in \Gamma, \Delta_1, x$  and  $s(\alpha_1) \in \Gamma, \Delta_1$  if  $A = !B$  for some  $B$

Subcase (3.1.1)  $s(\alpha_1) = x$

$A = \mathbf{1}$  or  $A = !B$  for some  $B$  By Lemma A.11

Subcase (3.1.1.1)  $A = \mathbf{1}$

Impossible, since  $P_1 \xrightarrow{\alpha_1} P'_1$  and  $s(\alpha_1) = x$  By Lemma A.1

Subcase (3.1.1.2)  $A = !B$  for some  $B$

Impossible, since  $s(\alpha_1) = x$  contradicts  $x \in \Gamma, \Delta_1$ .

**Case:**  $D = \text{cut}^! D_1 (u. D_2)$

$\Gamma; \vdash D_1 \rightsquigarrow P_1 :: y:A$

$\Gamma, u : A; \Delta \vdash D_2 \rightsquigarrow P_2 :: z:C$

$P \equiv (\nu u)(!u(y).P_1 \mid P_2)$

$\text{live}(P_2)$

by inversion  
since  $\text{live}(P)$

There is  $P'_2$  such that either  $P_2 \rightarrow P'_2$ , or  $P_2 \overset{\alpha_2}{\rightarrow} P'_2$

for some  $\alpha_2$  with  $s(\alpha_2) \in u, \Gamma, \Delta, z$  and  $s(\alpha_2) \in u, \Gamma, \Delta$  if  $C = !B$ .

Subcase (1):  $P_2 \rightarrow P'_2$

$P \rightarrow Q$  with  $Q \equiv (\nu u)(P_1 \mid P'_2)$

[satisfying (a)]

Subcase (2):  $P_2 \overset{\alpha_2}{\rightarrow} P'_2$

Subcase (2.1):  $s(\alpha_2) \neq u$

$P \overset{\alpha_2}{\rightarrow} Q$  with  $Q \equiv (\nu u)(P_1 \mid P'_2)$

where  $s(\alpha_2) \in \Gamma, \Delta, z$  and  $s(\alpha_1) \in \Gamma, \Delta$  if  $C = !B$

[satisfying (b)]

Subcase (2.2):  $s(\alpha_2) = u$

$P_2 \xrightarrow{(\nu y)u(y)} P'_2$

By Lemma A.1

$!u(y).P_1 \xrightarrow{u(y)} (P_1 \mid !u(y).P_1)$

$P \rightarrow Q$  with  $Q \equiv (\nu x)(\nu y)(P_1 \mid !u(y).P_1 \mid P'_2)$

[satisfying (a)]

□

**Theorem A.2.** Let  $\cdot; \cdot \vdash D \rightsquigarrow P :: x : \mathbf{1}$ . If  $\text{live}(P)$  then there is  $Q$  such that  $P \rightarrow Q$ .

*Proof.* (a)  $P \rightarrow Q$ , or (b)  $P \overset{\alpha}{\rightarrow} Q$  for some  $\alpha$  where  $s(\alpha) = x$ , by Lemma A.12. But  $P \overset{\alpha}{\rightarrow} Q$  with  $s(\alpha) = x$  is not possible since  $x : \mathbf{1}$ , by Lemma A.1. So  $P \rightarrow Q$ . □

#### A.4. Intuitionistic Duality

**Proposition A.3.** Let  $A$  be a  $!$ -free type. Then  $\Gamma; \Delta \vdash P :: x:A$  implies  $\Gamma; \Delta, x:\bar{A} \vdash P :: -:\mathbf{1}$ .

*Proof.* We make the proof term explicit, writing  $\Gamma; \Delta \vdash D \rightsquigarrow P :: x:A$ , and proceed by induction on the structure of  $D$ , constructing  $\Gamma; \Delta, x:\bar{A} \vdash P :: -:\mathbf{1}$  in each case (eliding a  $D'$ ). We show a few representative cases.

**Case:**  $D = \mathbf{1R}$

$\Delta = \cdot$  and  $P = \mathbf{0}$

$\Gamma; x:\mathbf{1} \vdash \mathbf{0} :: -:\mathbf{1}$

by inversion  
by  $\mathbf{1R}$  and  $\mathbf{1L}$

**Case:**  $D = \otimes R D_1 D_2$

$\Delta = (\Delta_1, \Delta_2)$ ,  $P = (\nu y)x\langle y \rangle. (P_1 \mid P_2)$ ,  $A = A_1 \otimes A_2$ ,

$\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow P_1 :: y:A_1$

$\Gamma; \Delta_2 \vdash D_2 \rightsquigarrow P_2 :: x:A_2$

by inversion

$\Gamma; \Delta_2, x:\bar{A}_2 \vdash P_2 :: -:\mathbf{1}$

by i.h. on  $D_2$

$\Gamma; \Delta_1, \Delta_2, x:A_1 \multimap \bar{A}_2 \vdash (\nu y)x\langle y \rangle. (P_1 \mid P_2) :: -:\mathbf{1}$

by rule  $\multimap L$

**Case:**  $D = \text{cut } D_1 (y. D_2)$

$\Delta = (\Delta_1, \Delta_2), P = (\nu y)(P_1 \mid P_2),$   
 $\Gamma; \Delta_1 \vdash D_1 \rightsquigarrow P_1 :: y:B$   
 $\Gamma; \Delta_2, y:B \vdash D_2 \rightsquigarrow P_2 :: x:A$   
 $\Gamma; \Delta_2, y:B, x:\bar{A} \vdash P_2 :: -:\mathbf{1}$   
 $\Gamma; \Delta_1, \Delta_2, x:\bar{A} \vdash (\nu y)(P_1 \mid P_2) :: -:\mathbf{1}$

by inversion  
 by i.h. on  $D_2$   
 by rule cut

**Case:**  $D = \text{cut}^! D_1 (u. D_2)$

$P = (\nu u)(!u(z).P_1 \mid P_2),$   
 $\Gamma; \vdash D_1 \rightsquigarrow P_1 :: z:B$   
 $\Gamma, u:B; \Delta \vdash D_2 \rightsquigarrow P_2 :: x:A$   
 $\Gamma, u:B; \Delta, x:\bar{A} \vdash P_2 :: -:\mathbf{1}$   
 $\Gamma; \Delta, x:\bar{A} \vdash (\nu u)(!u(z).P_1 \mid P_2) :: -:\mathbf{1}$

by inversion  
 by i.h. on  $D_2$   
 by rule cut<sup>!</sup>

**Case:**  $D = !R$ . This case is impossible, since  $A$  was assumed to be !-free.

□