# Notes on the linear $\pi$-calculus and LLP

Kohei Honda*

**Abstract.** We embed two versions of Laurent's polarised linear logic (LLP) in the linear $\pi$-calculus [5] and its variant and show there is a close correspondence between them in both dynamics and type structure. Among others the cut-elimination of LLP is derived from the strong normalisation of typed terms; while logical connectives and geometric structure of LLP offer deep insight on the hidden structure of the linear $\pi$-calculus.

## 1 Introduction

This note presents an embedding of Laurent's polarised linear logic (LLP) [4, 3] in a linear $\pi$-calculus introduced in [5]. First we consider a natural subsystem of LLP, which we embed in the original linear $\pi$-calculus [5], then move to the full LLP which is embedded in a calculus with a slightly different notion of linearity. In each case, there is a tight correspondence between cut-elimination of LLP proofs and the extended reduction of typed terms. We also discuss possible ways to obtain syntactic full completeness. The linear typing is a natural and simple way to extract a certain kind of interactive behaviour from the $\pi$-calculus, while LLP is a refinement of Linear Logic where the control of polarity leads to new insight on proof structures and their dynamics. The correspondence between them looks interesting from the following two viewpoints.

- The connection with LLP offers an analysis of the type structure of the linear $\pi$-calculus from a logical viewpoint. For example we obtain new insight on the linear type structures by relating their constructors to logical connectives of LLP; further the geometric structure of typed terms is elucidated through the connection.
- In the other direction, LLP's type structure and dynamics are placed in a broader setting of typed name passing interaction, which can be used for the analysis of LLP. For example the cut elimination of LLP is directly derivable from the strong normalisability of typed processes.

Structures of types arising from logics, especially those originating in proof theory, have played a significant rôle in the study of types for functional computation. The study of types from a practical viewpoint and the one motivated from the isomorphisms between types and logical formulae, although guided by different concerns, together uncovered the rich, if limited, domain of types for sequential computation. It is difficult to foresee the exact shape in which such a dialogue may be carried over to the emerging realm of interaction-based computing — computing where interaction, rather than function, is a pivotal element.

*Queen Mary and Westfield College, University of London, U.K.

But certainly foundational inquiries into types of computation in this new realm, regardless of the methods one may employ, are theoretically apt (since the setting is general) and practically motivating (since we need harness a wide class of behaviour). Further the investigation in the past decades led to, in many cases in fruitful connection to sequential notions of types, a couple of new threads of research and mathematical structures which centre on interaction: in theory of computing, the $\pi$-calculus; in logics, Linear Logic and various game semantics. Their close connection, at least at the level of bare mathematical structure, suggests, even though in a nascent form, a possible existence of a fertile scientific field. Here I present a brief report on my recent finding, which comes from my visit to LMS in Marseilles where I talked about the typed asynchronous $\pi$-calculus: its spirit and a recent basic result, as well as a possible connection to LLP. The note reflects discussions with members of LMS. The results, small as they are, may hopefully offer some material for the reader's thought on the future prospect of interaction between logics and theories of computing.

Section 2 reviews the linear $\pi$-calculus. Section 3 embeds a subsystem of LLP in the calculus. Section 4 relaxes the linear type structure, inducing two distinct notions of control, one linear and another weakly-linear. In Section 5 the latter is used for the embedding of LLP.

## 2 Linear $\pi$-calculus

### 2.1 Types

The set of *action modes* is given by $\{\downarrow, \uparrow, \mathbf{!}, \mathbf{?}\}$. $p_{\mathtt{I}}, q_{\mathtt{I}}$ range over $\{\downarrow, \mathbf{!}\}$, while $p_{\mathtt{O}}, q_{\mathtt{O}}$ range over $\{\uparrow, \mathbf{?}\}$. $p, q, \ldots$ range over the whole set. We define the self-inverse injection $\overline{p}$ by $\overline{\downarrow} = \uparrow$ and $\overline{\mathbf{!}} = \mathbf{?}$. Using these data, we first define the set of *directed channel types* by the following grammar.

$$\tau \ ::= \ \tau_{\mathtt{I}} \mid \tau_{\mathtt{O}} \qquad \tau_{\mathtt{I}} \ ::= \ (\vec{\tau}_{\mathtt{O}})^{p_{\mathtt{I}}} \mid [\&_{i\in I}\vec{\tau}_{\mathtt{O}i}]^{p_{\mathtt{I}}} \qquad \tau_{\mathtt{O}} \ ::= \ (\vec{\tau}_{\mathtt{I}})^{p_{\mathtt{O}}} \mid [\oplus_{i\in I}\vec{\tau}_{\mathtt{O}i}]^{p_{\mathtt{O}}}$$

Above $\vec{\tau}$ denotes a vector of directed channel types. The indexing set $I$ can be arbitrary but in the present study $I = \{1, 2\}$ is enough. $\tau_{\mathtt{I}}, \ldots$ are called *input types*, while $\tau_{\mathtt{O}}, \ldots$ are called *output types*. We define $\overline{\tau}$, the *dual* of $\tau$, by dualising the action modes and exchanging $\oplus$ and $\&$ of $\tau$. We also use a type of form $\langle\tau, \overline{\tau}\rangle$, called *pair channel type*, which is considered as an unordered pair.[1] $\alpha, \beta, \ldots$ range over pair types, while $\delta, \delta', \ldots$ range over the set of all channel types. $\mathsf{md}(\tau)$ is the outermost action mode of $\tau$. $\mathsf{md}(\alpha)$ is $\perp$ if the modes of its components are $\{\downarrow, \uparrow\}$, and is $\mathbf{!}$ if else.

We define the commutative partial operation $\odot$ over channel types by the following rules (we take the least partial commutative operation including them).

- With $\mathsf{md}(\tau) = \downarrow$, $\tau \odot \overline{\tau} = \langle\tau,\ \overline{\tau}\rangle$,
- With $\mathsf{md}(\tau) = \mathbf{!}$, $\tau \odot \overline{\tau} = \langle\tau,\ \overline{\tau}\rangle$, $\langle\tau,\ \overline{\tau}\rangle \odot \overline{\tau} = \langle\tau,\ \overline{\tau}\rangle$ and $\overline{\tau} \odot \overline{\tau} = \overline{\tau}$.

[1] The use of pair types is not essential from a semantic viewpoint. However they allow a finer control of dynamics, which we prefer.

The first rule says only a unique input and its dual can compose in a linear channel. The algebra of replicated types says a replicated type allows the composition of zero or more outputs and a unique input.

Let $x, y, z, \ldots$, or sometimes $a, b, c, \ldots$, range over a fixed countable set of *names*. We can now introduce action types.

**Definition 1.** *An* action type, *denoted $A, B, \ldots$, is a finite directed graph with nodes of the form $x^\delta$, called* typed names, *such that: (1) no names occur twice, and (2) an edge has form $x^\delta \to y^{\delta'}$ such that either (1) that $\mathsf{md}(\delta) = \downarrow$ and $\mathsf{md}(\delta) = \uparrow$, (2) $\mathsf{md}(\delta) = \mathbf{!}$ and $\mathsf{md}(\delta') = \mathbf{?}$, or (3) $x^\delta = \{\mathbf{!}, \mathbf{?}\}$ and $\mathsf{md}(\delta') = \mathbf{?}$.*

We often indicate the existence of a sequence of edges from $x$ to $y$ in such a graph by writing $x \to^+ y$. $|A|$ denotes typed names in $A$. We often consider $|A|$ as a finite function from names to channel types. $A \backslash \vec{x}$ is the result of taking off nodes with names in $\vec{x}$ from $A$. We say $A$ and $B$ are *compatible*, denoted $A \asymp B$, when (1) $x^\delta \in |A|$ and $x^{\delta'} \in |B|$ implies $\delta \odot \delta'$ is defined, and (2) whenever $x \to^+ y$ in $A \uplus B$ ($\uplus$ is the graph union) we have $x \neq y$. When $A \asymp B$, we define $A \odot B$ by the following data:

- $|A \odot B|(x) = \delta$ iff either (1) $|A|(x) = \delta$ and $x \notin \mathsf{fn}(B)$, (2) the symmetric case of (1), or (3) $\delta = \delta_1 \odot \delta_2$ where $|A|(x) = \delta_1$ and $|B|(x) = \delta_2$.
- $x \to y$ in $A \odot B$ iff, in $A \uplus B$, $x \to^+ y$ $(n \leq 0)$ and for no $x'$ and $y'$ we have $x' \to x$ and $y \to y'$.

### 2.2 Linear Typing

Processes are given by the following syntax.

$$
\begin{aligned}
P \ ::= \ & x(\vec{y}).P \mid !x(\vec{y}).P \mid \overline{x}(\vec{y})P \mid x[\&_{i\in I}(\vec{y_i}).P_i] \mid !x[\&_{i\in I}(\vec{y_i}).P_i] \mid \overline{x}\mathtt{in}_i(\vec{y})P \mid \\
& P|Q \ \mid \ (\boldsymbol{\nu}\, x)P \ \mid \ \mathbf{0}
\end{aligned}
$$

The structural equality includes those which make the output prefix asynchronous, e.g. $(\overline{x}(\vec{y})P)|Q \equiv \overline{x}(\vec{y})(P|Q)$ when $\mathsf{fn}(P) \cap \{\vec{y}\} = \emptyset$. The dynamics for unary prefixes is given by the following two rules:

$$
\begin{aligned}
x(\vec{y}).P|\overline{x}(\vec{y})Q &\longrightarrow (\boldsymbol{\nu}\, \vec{y})(P|Q) \\
!x(\vec{y}).P|\overline{x}(\vec{y})Q &\longrightarrow !x(\vec{y}).P|(\boldsymbol{\nu}\, \vec{y})(P|Q)
\end{aligned}
$$

Branching prefixes have the following dynamics:

$$
\begin{aligned}
x[\&_i(\vec{y_i}).P_i]|\overline{x}\mathtt{in}_i(\vec{y})Q &\longrightarrow (\boldsymbol{\nu}\, \vec{y_i})(P_i|Q) \\
!x[\&_i(\vec{y_i}).P_i]|\overline{x}\mathtt{in}_i(\vec{y})Q &\longrightarrow !x[\&_i(\vec{y_i}).P_i]|(\boldsymbol{\nu}\, \vec{y_i})(P_i|Q)
\end{aligned}
$$

Note unselected branches are eliminated in the reduction of branching.

The typing rules are a sequential version of [5], cf.[1]. We use the *IO-modes* $\phi \in \{\mathtt{I}, \mathtt{O}\}$ with partial algebra $\mathtt{I} \odot \mathtt{O} = \mathtt{O}$ and $\mathtt{I} \odot \mathtt{I} = \mathtt{I}$. If $\phi_1 \odot \phi_2$ is defined then we write $\phi_1 \asymp \phi_2$. The sequent takes the form $\vdash_\phi P \triangleright A$. We first present the three composition rules together with two weakning rules (the latter does not affect dynamics but makes the typing neater). Below and henceforth $C/x_1^{\delta_i}..x_n^{\delta_n} = A$ denotes, simulteneously: (1) $C$ contains $x_i^{\delta_i}$ for each $i$, and (2) $A$ is the result of taking off those nodes from $C$. $\otimes$ is the union of graphs with disjoint free names.

$$
\text{(Zero)}\ \frac{-}{\vdash_{\mathtt{I}} \mathbf{0} \triangleright \emptyset}
\qquad
\text{(Par)}\ \frac{\vdash_{\phi_i} P_i \triangleright A_i \quad (i=1,2) \qquad A_1 \asymp A_2 \quad \phi_1 \asymp \phi_2}{\vdash_{\phi_1 \odot \phi_2} P_1|P_2 \triangleright A_1 \odot A_2}
\qquad
\text{(Res)}\ \frac{\vdash_{\phi} P \triangleright C \qquad C/x^{\alpha} = A}{\vdash_{\phi} (\boldsymbol{\nu}\, x)P \triangleright A}
$$

$$
\text{(Weak-}\bot,?\text{)}\ \frac{\vdash_{\phi} P \triangleright A^{\text{-}x} \qquad \mathsf{md}(\delta) \in \{\bot, ?\}}{\vdash_{\phi} P \triangleright A \otimes x^{\delta}}
\qquad
\text{(Weak-}\mathtt{O}\text{)}\ \frac{\vdash_{\mathtt{I}} P \triangleright A}{\vdash_{\mathtt{O}} P \triangleright A}
$$

Next we present prefix rules.

$$
\text{(In}^{\downarrow}\text{)}\ \frac{\vdash_{\mathtt{O}} P \triangleright C^{\text{-}x} \qquad C/\vec{y}^{\vec{\tau}} = \uparrow A \otimes ?B}{\vdash_{\mathtt{I}} x(\vec{y}).P \triangleright (x^{(\vec{\tau})^{\downarrow}} \to A) \otimes B}
\qquad
\text{(In}^{!}\text{)}\ \frac{\vdash_{\mathtt{O}} P \triangleright C^{\text{-}x} \qquad C/\vec{y}^{\vec{\tau}} = ?A}{\vdash_{\mathtt{I}} !\, x(\vec{y}).P \triangleright x^{(\vec{\tau})^{!}} \to A}
$$

$$
\text{(Out)}\ \frac{\vdash_{\mathtt{I}} P \triangleright C \qquad C/\vec{y}^{\vec{\tau}} = A \asymp x^{(\vec{\tau})^{p_{\mathrm{O}}}}}{\vdash_{\mathtt{O}} \overline{x}(\vec{y})P \triangleright A \odot x^{(\vec{\tau})^{p_{\mathrm{O}}}}}
$$

The rules for branching and selection are analogously given, see Appendix. It is easy to see the typing system is equivalent to the original one in [5] except the sequentiality constraint is added. In particular whenever $\vdash_{\phi} P \triangleright A$ in the above system we have $|A| \vdash P \triangleright \langle A \rangle$ in the original system in [5] where $\langle A \rangle$ is the graph made by replacing channel types in $A$ with their modes. From this we can immediately conclude, among others, that the typing is closed under injective renaming, $\equiv$ and the reduction relation (for subject reduction we additionally use (Weak-$\mathtt{O}$) for IO-modes).

### 2.3 Some properties of typed terms

We list a few properties of typed terms which are of interest in the context oft he present note. They are concerned with the extended reduction introduced in [5] and their normal forms.

First, the *extended reduction*, which eliminates all dual and restricted channels, is generated by the following three rules. In the rules we assume the standard variable convention for bound names.

$$
\begin{aligned}
x(\vec{y}).P|C[\overline{x}(\vec{y})Q] &\longrightarrow_l C[(\boldsymbol{\nu}\,\vec{y})(P|Q)] \\
!x(\vec{y}).P|C[\overline{x}(\vec{y})Q] &\longrightarrow_r !x(\vec{y}).P|C[(\boldsymbol{\nu}\,\vec{y})(P|Q)] \\
(\boldsymbol{\nu}\,x)!x(\vec{y}).P &\longrightarrow_g \mathbf{0}
\end{aligned}
$$

The rules for branching are similarly defined. We define $\rightarrow_e$ to be the compatible relation generated from the above rules and set $\Rightarrow_e \stackrel{\text{def}}{=} \equiv \cup \rightarrow_e^*$. We can easily check that if $\vdash_{\phi} P \triangleright A$ and $P \Rightarrow_e Q$ then $\vdash_{\phi} Q \triangleright A$ again. Further $\Rightarrow_e$ is Church-Rosser up to $\equiv$. Write $P \Downarrow_e$ if $P$ does not have an infinite extended reduction sequence. We now observe the following. For the proof see [5].

**Proposition 1.** (strong normalisability of $\Rightarrow_e$) $\vdash_{\phi} P \triangleright A$ *implies* $P \Downarrow_e$.

Typable normal forms w.r.t $\Rightarrow_e$ are *extended normal forms.* $\mathsf{ENF}$ denotes their collection. By the definition of $\rightarrow_e$, if $P \in \mathsf{ENF}$ then (1) no name occurs as both input and output in $P$ and (2) $P$ has no substantial hiding (i.e. $P$ does not have a subterm of form $(\boldsymbol{\nu}\, x)P'$ where $x$ occurs free in $P'$).

The remaining two propositions, whose proofs are elementary and are omitted, clarify the geometric properties of typed processes in extended normal forms. First let us say $P$ is *connected* when $P \equiv P_1|P_2$ implies $P_1 \equiv \mathbf{0}$ or $P_2 \equiv \mathbf{0}$. Also $x \in \mathsf{fn}(A)$ is *input in* $A$ if $\mathsf{md}(|A|(x)) \in \{\downarrow, !\}$.

**Proposition 2.** *Let $P \in \mathsf{ENF}$. Then $P$ is connected iff $\vdash_A P \triangleright \phi$ such that either: (1) $\phi = \mathtt{I}$ and there is at most one input name in $A$, or (2) $\phi = \mathtt{O}$ and there is no input name in $A$.*

Let $P \in \mathsf{ENF}$ be connected and $P \not\equiv \mathbf{0}$. By Proposition 2, $P$ has a unique active name. This is $P$'s *primary channel.* The internal structure of extended normal forms becomes clear by the following.

**Proposition 3.** *Let $P \in \mathsf{ENF}$ be connected with primary channel $x$ typed with a unary type. (1) If $x$ is input then either $P \equiv x(\vec{y}).P'$ or $P \equiv !x(\vec{y}).P'$ where, in each case, $P'$ is again connected. (2) If $x$ is output then $P \equiv \overline{x}(y_1..y_n)\Pi_{1\leq n\leq n}P_i$, where each $P_i$ is again connected with primary channel $y_i$. Similarly when $x$ is typed with a branching type.*

## 3 Embedding LLP (1)

### 3.1 A subsystem of $\mathsf{LLP}$

We first introduce the set of logical formulae of $\mathsf{LLP}$. Assume given atomic formulae, written $X, X', \ldots$. The sets of propositional formulae of $\mathsf{LLP}$ are given by the following grammar.

$$\begin{array}{lcl} \mathrm{N} & ::= & X \mid \mathrm{N}⅋\mathrm{M} \mid \mathrm{N}\&\mathrm{M} \mid ?\mathrm{P} \mid \uparrow\mathrm{P} \\ \mathrm{P} & ::= & X^{\perp} \mid \mathrm{P}\otimes\mathrm{Q} \mid \mathrm{P}\oplus\mathrm{Q} \mid !\mathrm{N} \mid \downarrow\mathrm{N} \end{array}$$

Note there are two kinds of formulae, *nagative* $(\mathrm{N}, \mathrm{M}, \ldots)$ and *positive* $(\mathrm{P}, \mathrm{Q}, \ldots)$. The set of negative and positive formulae are ranged over by $\mathrm{A}, \mathrm{B}, \ldots$. We omitted units since, at this point, the encoding of additive units looks elusive (they do have logical content, but computationally their nature is not straightforward).

We work with two logical systems which use these formulae. The first one, which we treat in this section, is a subsystem of $\mathsf{LLP}$ in which we restrict the rules for exponentials (by which it also becomes a subsystem of Linear Logic [2]). $\Gamma$ denotes a set of formulae, $\mathcal{N}$ a set of negative formulae, and "$\Gamma, \Delta$" the disjoint union of $\Gamma$ and $\Delta$. $?\Gamma$ indicates that all formulae in $\Gamma$ has form $?P$.

$$
\begin{array}{cc}
(\mathrm{Ax})\dfrac{-}{\vdash \mathrm{N}, \mathrm{N}^{\perp}} & (\mathrm{Cut})\dfrac{\vdash \Gamma, \mathrm{N} \quad \vdash \Delta, \mathrm{N}^{\perp}}{\vdash \Gamma, \Delta} \\[2ex]
(\parr)\dfrac{\vdash \mathcal{N}, \mathrm{N}, \mathrm{M}}{\vdash \mathcal{N}, \mathrm{N}\parr\mathrm{M}} & (\otimes)\dfrac{\vdash \Gamma, \mathrm{P} \quad \vdash \Delta, \mathrm{Q}}{\vdash \Gamma, \Delta, \mathrm{P}\otimes\mathrm{Q}} \\[2ex]
(\&)\dfrac{\vdash \mathcal{N}, \mathrm{N} \quad \vdash \mathcal{N}, \mathrm{M}}{\vdash \mathcal{N}, \mathrm{N}\&\mathrm{M}} & (\oplus_l)\dfrac{\vdash \Gamma, \mathrm{P}}{\vdash \Gamma, \mathrm{P}\oplus\mathrm{Q}} \quad (\oplus_r)\dfrac{\vdash \Gamma, \mathrm{Q}}{\vdash \Gamma, \mathrm{P}\oplus\mathrm{Q}} \\[2ex]
(?_d)\dfrac{\vdash \Gamma, \mathrm{P}}{\vdash \Gamma, ?\mathrm{P}} & (!)\dfrac{\vdash ?\Gamma, \mathrm{N}}{\vdash ?\Gamma, !\mathrm{N}} \\[2ex]
(?_w)\dfrac{\vdash \Gamma}{\vdash \Gamma, ?\mathrm{P}} & (?_c)\dfrac{\vdash \Gamma, ?\mathrm{P}, ?\mathrm{P}}{\vdash \Gamma, ?\mathrm{P}} \\[2ex]
(\uparrow)\dfrac{\vdash \Gamma, \mathrm{P}}{\vdash \Gamma, \uparrow\mathrm{P}} & (\downarrow)\dfrac{\vdash \mathcal{N}, \mathrm{N}}{\vdash \mathcal{N}, \downarrow\mathrm{N}}
\end{array}
$$

Note that whenever a sequent is provable in $\mathsf{LLP}^{-}$ then it is provable in $\mathsf{LLP}$.

### 3.2 Embedding

Formulae are mapped to linear types as follows. For simplicity we map all atomic formulae to a single type.[2]

$$
\begin{array}{rlrl}
X^{\circ} & \stackrel{\mathrm{def}}{=} ()^{\uparrow} & (X^{\perp\circ}) & \stackrel{\mathrm{def}}{=} ()^{\downarrow} \\
(\mathrm{N}\parr\mathrm{M})^{\circ} & \stackrel{\mathrm{def}}{=} ((\mathrm{N}^{\circ}\mathrm{M}^{\circ})^{\downarrow})^{\uparrow} & (\mathrm{P}\otimes\mathrm{Q})^{\circ} & \stackrel{\mathrm{def}}{=} ((\mathrm{P}^{\circ}\mathrm{Q}^{\circ})^{\uparrow})^{\downarrow} \\
(\mathrm{N}\&\mathrm{M})^{\circ} & \stackrel{\mathrm{def}}{=} ([\mathrm{N}^{\circ}\&\mathrm{M}^{\circ}]^{\downarrow})^{\uparrow} & (\mathrm{P}\oplus\mathrm{Q})^{\circ} & \stackrel{\mathrm{def}}{=} ([\mathrm{P}^{\circ}\oplus\mathrm{Q}^{\circ}]^{\uparrow})^{\downarrow} \\
(\mathrm{N}\&\mathrm{M})^{\circ} & \stackrel{\mathrm{def}}{=} ([\mathrm{N}^{\circ}\&\mathrm{M}^{\circ}]^{\downarrow})^{\uparrow} & (\mathrm{P}\oplus\mathrm{Q})^{\circ} & \stackrel{\mathrm{def}}{=} ([\mathrm{P}^{\circ}\oplus\mathrm{Q}^{\circ}]^{\uparrow})^{\downarrow} \\
(?\mathrm{P})^{\circ} & \stackrel{\mathrm{def}}{=} (\mathrm{P}^{\circ})^{?} & (!\mathrm{N})^{\circ} & \stackrel{\mathrm{def}}{=} (\mathrm{N}^{\circ})^{!} \\
(\uparrow\mathrm{P})^{\circ} & \stackrel{\mathrm{def}}{=} (\mathrm{P}^{\circ})^{\uparrow} & (\downarrow\mathrm{N})^{\circ} & \stackrel{\mathrm{def}}{=} (\mathrm{N}^{\circ})^{\downarrow}
\end{array}
$$

The mapping is completely syntactic, so that we do not obtain any additional equations. The interpretation on the one hand uncovers "hidden" logical connectives, $\otimes$ and $\parr$, in the linear $\pi$-calculus, and, on the other, suggests the "origins" of polarities as utilised in $\mathsf{LLP}$ and $\mathsf{LLP}^{-}$ from a process-theoretic viewpoint. These observations also pertain to the acyclicity in causality in these systems (this aspect is more clearly seen in the term annotation given below).

Next we interpret proofs as processes by annotating the above inference rules with typed terms. We use a sequent of form $P \vdash x : \mathrm{A}, y : \mathrm{B}, \ldots$. Accordingly

[2] A different treatment of atomic formulae is possible, though this affects neither the stated results nor proof methods.

symbols such as $\Gamma$ and $\mathcal{N}$ range over the sets of name-formula pairs.

$$
(\mathrm{Ax})\frac{-}{[x \to y] \vdash x:\mathrm{N}, y:\mathrm{N}^{\perp}} \qquad (\mathrm{Cut})\frac{P \vdash \Gamma, x:\mathrm{N} \quad Q \vdash \Delta, x:N^{\perp}}{(\boldsymbol{\nu}\, x)(P|Q) \vdash \Gamma, \Delta}
$$

$$
(\invamp)\frac{P \vdash \mathcal{N}, x:\mathrm{N}, y:\mathrm{M}}{\overline{z}(u)u(xy).P \vdash \mathcal{N}, z:\mathrm{N}\invamp\mathrm{M}} \qquad (\otimes)\frac{P \vdash \Gamma, x:\mathrm{P} \quad Q \vdash \Delta, y:\mathrm{Q}}{z(u).\overline{u}(xy)(P|Q) \vdash \Gamma, \Delta, z:\mathrm{P}\otimes\mathrm{Q}}
$$

$$
(\&)\frac{P_i \vdash \mathcal{N}, x_i:\mathrm{N}_i \quad (i=1,2)}{\overline{z}(u)u[\&_i(x_i).P_i \vdash \mathcal{N}, z:\mathrm{N}_1\&\mathrm{N}_2} \qquad (\oplus_l)\frac{P \vdash \Gamma, x:\mathrm{P}}{z(u).\overline{u}\mathtt{in}_1(x)P \vdash \Gamma, x:\mathrm{P}\oplus\mathrm{Q}}
$$

$$
(?_d)\frac{P \vdash \Gamma, x:\mathrm{P}}{\overline{z}(x)P \vdash \Gamma, z:?\mathrm{P}} \qquad (!)\frac{P \vdash ?\Gamma, x:\mathrm{N}}{!z(x).P \vdash ?\Gamma, z:!\mathrm{N}}
$$

$$
(?_w)\frac{P \vdash \Gamma}{P \vdash \Gamma, ?\mathrm{P}} \qquad (?_c)\frac{P \vdash \Gamma, x:?\mathrm{P}, y:?\mathrm{P}}{P\{z/xy\} \vdash \Gamma, z:?\mathrm{P}}
$$

$$
(\uparrow)\frac{P \vdash \Gamma, x:\mathrm{P}}{\overline{y}(x)P \vdash \Gamma, y:\uparrow\mathrm{P}} \qquad (\downarrow)\frac{P \vdash \mathcal{N}, x:\mathrm{N}}{y(x).P \vdash \mathcal{N}, y:\downarrow\mathrm{N}}
$$

Immediately $P \vdash \Gamma$ iff $\vdash \Gamma$ (with $\Gamma$ in the r.h.s. forgetting names). Note terms are indeed processes in Section 2, though it is not immediate they are typable or not. In fact they are typable by the interpretation of the corresponding formula (as it needs be!), as the following development shows.

First we need to translate $\Gamma$ (as name-formula pairs) into an action type. For this purpose let us remember, from [4], that $\vdash \Gamma$ in $\mathsf{LLP}$ (hence in $\mathsf{LLP}^{-}$) implies $\Gamma$ includes at most one positive formula. Thus if $\Gamma, \mathrm{P}$ is derivable then it can be written as $\mathcal{N}, \mathrm{P}$. Note also formulae in $\mathcal{N}$ are divided into those with mode ? and those without. Thus we reach the desired translation.

$$
\begin{aligned}
(\_)^{\circ} &\stackrel{\mathrm{def}}{=} \_ \\
(x:\mathrm{N},\ \mathcal{N})^{\circ} &\stackrel{\mathrm{def}}{=} x^{\mathrm{N}^{\circ}},\ \mathcal{N}^{\circ} \\
(x:\mathrm{P},\ \mathcal{N},\ ?\Gamma)^{\circ} &\stackrel{\mathrm{def}}{=} \begin{cases} (x^{\mathrm{P}^{\circ}} \to \mathcal{N}^{\circ}),\ \Gamma^{\circ} & (\mathrm{P}\ \text{linear}) \\ (y^{\mathrm{P}^{\circ}} \to \Gamma^{\circ}),\ \mathcal{N}^{\circ} & (\mathrm{P}\ \text{exponential}) \end{cases}
\end{aligned}
$$

Given the interpretation it is mostly mechanical to check the typability of terms. Only (Cut) and ($?_c$) are of some interest.

- **(Cut).** We use the following observation: *If* $\vdash\ !\mathrm{N}, \Gamma$ *in* $\mathsf{LLP}^{-}$ *then* $\Gamma$ *only contains* ?*-formulae* (note, for example, ($\otimes$) needs to use negative formulae so ! cannot survive this rule without changing its polarity). First assume N is a ?-formula. By induction hypothesis and by the observation above we have $\vdash_{\mathtt{I}} Q \triangleright x^{(\mathrm{N}^{\perp})^{\circ}} \to \Delta^{\circ}$ with $?\Delta$. If $\phi = \mathtt{O}$ the typability is immediate. If $\phi = \mathtt{I}$ then we have either $\vdash_{\mathtt{I}} P \triangleright (y^{\mathrm{P}^{\circ}} \to {\Gamma_1}^{\circ}) \otimes {\Gamma_2}^{\circ} \otimes x^{\mathrm{N}^{\circ}}$ (with $\Gamma = y:\mathrm{P}, \Gamma_1, ?\Gamma_2$) or $\vdash_{\mathtt{I}} P \triangleright y^{\mathrm{P}^{\circ}} \to ({\Gamma'}^{\circ} \otimes x^{\mathrm{N}^{\circ}})$ (with $\Gamma = y:\mathrm{P}, ?\Gamma'$). Thus either $\vdash_{\mathtt{I}} (\boldsymbol{\nu}\, x)(P|Q) \triangleright (y^{\mathrm{P}^{\circ}} \to \Gamma_1^{\circ}) \otimes {\Gamma_2}^{\circ} \otimes \Delta^{\circ}$ (the former case) or $\vdash_{\mathtt{I}} (\boldsymbol{\nu}\, x)(P|Q) \triangleright y^{\mathrm{P}^{\circ}} \to ({\Gamma'}^{\circ} \otimes \Delta^{\circ})$ (the latter), as required.
- **($?_c$).** This is from the general property of the linear $\pi$-calculus: *whenever* $\vdash_{\phi} P \triangleright A$ *with* $|A|(x) = |A|(y)$ *such that its mode is* **?**, *we have* $\vdash_{\phi} P\{z/xy\} \triangleright A\{z/xy\}$ *with* $z$ *fresh.* The proof is by mechanical induction.

We thus know, writing $P \vdash_{\mathsf{LLP}^-} \Gamma$ if $P \vdash \Gamma$ is derivable:

**Theorem 1.** (soundness) $P \vdash_{\mathsf{LLP}^-} \Gamma$ *implies* $\vdash_{io(\Gamma)} P \triangleright \Gamma^\circ$.

By Proposition 1 the term $P$ of $P \vdash \Gamma$ is strongly normalising with respect to $\Rightarrow_e$. Since a single cut-elimination step over proofs (in terms of natural proof nets for $\mathsf{LLP}^-$) precisely corresponds to two extended reduction (except for axioms which only need one), this also means the cut-elimination procedure of $\mathsf{LLP}^-$-proofs is strongly normalising. Note $\Rightarrow_e$-normal forms do not contain mutually dual names. We thus obtain the following. The result is also a consequence of cut elimination in Linear Logic [2], as well as that of $\mathsf{LLP}$ itself [4].

**Corollary 1.** (cut elimination) *The cut rule is redundant in* $\mathsf{LLP}^-$. *In fact the cut elimination procedure always terminates with a unique normal form.*

### 3.3 Remarks on full embedding

Propositions 2 and 3 together with the correspondence in type structures indicates close connection between $\mathsf{LLP}^-$ and the linear $\pi$-calculus. However it is a different matter whether the inhabitants of $\mathsf{LLP}^-$-formulae precisely coincide with their proofs or not. Direct syntactic correspondence do not work, even up to basic conversion rules on typed terms of the kind studied in [5]. We suspect the situation remains the same even semantically: this is to be expected because, in the present interpretation, the type structure of $\mathsf{LLP}^-$ is decomposed into a finer structure and, in spite of this, we cannot use the "linearity constraint" on channel types which played a crucial rôle for a similar result in [5].

Since our interest lies in understanding the relationship between the linear $\pi$-calculus and Linear Logic, it is interesting to ask how we can bring these two paradigms closer. On the one hand, one may consider a different version of the linear $\pi$-calculus in which, for example, we allow an input type to carry input types (so we have a typed term of form $x(y).y(z).P$). This is far from impossible but leads to some complications, for example w.r.t. sequentiality. On the other, one may consider bringing $\mathsf{LLP}^-$ nearer to the present linear typing. Let us discuss one simple way to do so.

We only change tensors and sums, for which we introduce refined connectives, say $\otimes$', $⅋$', $\oplus$' and &'. In these connectives, we do not change the polarity of carried types (e.g. P$\otimes$'Q for tensor) but the resulting polarity is now rotated (e.g. P$\otimes$'Q is now negative). The typing rules do not change. The motivation is to keep geometric nature of the original connectives (the disjointness of $\otimes$', the connectedness of $⅋$', ...) while allowing their representation as composite connectives. For example the original $\otimes$ can be mapped as $(\mathrm{P}\otimes\mathrm{Q})^\circ \stackrel{\text{def}}{=} \downarrow(\mathrm{P}^\circ\otimes'\mathrm{Q}^\circ)$. The embedding into the linear $\pi$-calculus is now evident so that a single cut elimination now corresponds to a single extended reduction. We can check basic algebraic properties of the (composite) connectives listed in [4] do hold in this interpretation. Further, using Proposition 3 and focussing on the type (formula) of its primary channel, we can easily show typable terms are precisely the proofs of the logic, up to a minor syntactic conversion (this conversion is only due to axiom links: if we use a different representation for them we have exact syntactic fullness up to $\equiv$). For reference the outline is discussed in Appendix B. Thus we obtain a closest possible correspondence between a logical system and the linear $\pi$-calculus. It is yet to see whether such a formalism can play a useful rôle in the study of type structure for interactive computing.

# 4 Linearity and control

## 4.1 Relaxing linearity for control

In the following we discuss a relaxation of linear type structure motivated by representation of the notion of control, as found in "call-cc" in functional calculi. As in pure linear types, their "control" version is simple and intuitive. I note that the following structure is currently being studied by my colleagues as part of our investigation into basic type structures of the $\pi$-calculus.

But what is control? It says that a procedure may return not only at its "principal return point" but also at its "emergency return point". Since more than one of the latter may exist, this means we need to relax linearity. For this purpose we introduce a pair of "weakly linear" modes, $\downarrow'$ and $\uparrow'$, which are mutually dual. One can consider the original $\uparrow$ represents the "principal return point", while $\uparrow'$ indicates a "secondary return point". By duality, an input channel with $\downarrow$-mode can expect to receive an output of $\uparrow$ or $\uparrow'$-mode. However a channel of $\downarrow'$-mode may or may not receive its dual output (in this sense it is close to affine types [1], even though it differs in that we expect computation as a whole surely terminates).

These observations lead to the following algebra, where we write $w(\tau)$ for the weakened version of strongly linear $\tau$.

- With $\mathsf{md}(\tau) = \downarrow$, $\tau \odot \overline{\tau} = \langle \tau,\ \overline{\tau} \rangle$, $\quad \tau \odot \overline{w(\tau)} = \tau$ and $\overline{\tau} \odot \overline{w(\tau)} = \overline{\tau}$.
- With $\mathsf{md}(\tau) = \downarrow'$, $\tau \odot \overline{\tau} = \langle \tau,\ \overline{\tau} \rangle$, $\quad \overline{\tau} \odot \overline{\tau} = \overline{\tau}$, $\langle \tau,\ \overline{\tau} \rangle \odot \overline{\tau} = \langle \tau,\ \overline{\tau} \rangle \odot \overline{\tau}$.

The typing rule for ($\mathrm{In}^{\downarrow'}$) is precisely as in the original ($\mathrm{In}^{\downarrow}$), replacing $\uparrow A$ with $\uparrow' A$. On the other hand, we relax ($\mathrm{In}^{\downarrow}$) so that $?B$ becomes $?\uparrow' B$ (i.e. a linear input suppresses a weakly linear output), and similarly relax ($\mathrm{In}^{!}$) by replacing $?A$ with $?\uparrow' A$ (so that one can return from the midst of interaction). Finally (Weak) allows a name with $\uparrow'$-mode to get weakened.

The typing is still linear since there is a unique input for both strongly and weakly linear channels: thus even though multiple output is possible at a weakly linear channel, only one interaction can ever take place.

## 4.2 Some words on strong and weak linearity in control

The type structure presented above gives a clean articulation of two distinct notions of control in literature, which may be explained by the following two versions of the key abstraction rule in the $\lambda\mu$-calculus.

$$(\mu_1)\frac{\Gamma \vdash M \rhd \alpha\,;\,\Delta, a\!:\!\alpha}{\Gamma \vdash (\mu a\!:\!\alpha)M \rhd \alpha\,;\,\Delta} \qquad (\mu_2)\frac{\Gamma \vdash M \rhd \bot;\,\Delta, a\!:\!\alpha}{\Gamma \vdash (\mu a\!:\!\alpha)M \rhd \alpha\,;\,\Delta}$$

The rule $(\mu_1)$ says that an expression itself has type $\alpha$ ("the principal return"), but it will also have one or more secondary returns of the same type. This precisely corresponds to control with strongly linearity, for example $(\vec{\tau}\rho)^!$ with $\mathsf{md}(\rho) = \uparrow$. The notion also corresponds to call-cc (in programming) and weak bracketing (in games). A significant point of the present approach is that the representation via weak linearity represent the notion of control without relying on a particular evaluation strategy, just as it is so in the case of pure linearity.

In $(\mu_2)$, we allow the case when the expression does not have the main return point, which naturally, corresponds to a weak linear control, for example when we put $\mathsf{md}(\rho) = \uparrow'$ above. This idea may look a little unnatural if we consider call-cc as the basic primitive for control. But it is simpler and has a correspondence with type structures in classical logic.[3]. linear control disappear in affine types, cf.[1].

### 4.3 Properties of Typed Terms

Terms of the linear $\pi$-calculus with control satisfy all properties mentioned in Section 2.3. In particular typable terms are strongly normalising with respect to both usual reduction and extended reduction. Let us define $\to_e$ by the previous rules (with linear reduction restricted to strongly linear channels) together with the following rule:

$$x(\vec{y}).P|C[\overline{x}(\vec{y})Q] \;\to_e\; x(\vec{y}).P|C[(\boldsymbol{\nu}\,\vec{y})(P|Q)]$$

where $x$ is a weakly linear channel (note we mention types in the rule). Perhaps surprisingly, this rule is type-correct. We can then prove, writing $\vdash^{c}_{\phi} P \rhd A$ for a typed term in the new system and again writing $\Downarrow_e$ for convergence with respect to $\to_e$,

**Proposition 4.** *If* $\vdash^{c}_{\phi} P \rhd A$ *then* $P \Downarrow_e$.

For the proof we first observe that the above rule for extended reduction already means that we can reduce weak linearity to replicated linearity. Using this we can reduce the strong normalisability in the present system to that of the original linear $\pi$-calculus, hence as required. Very simple!

Another observation on typed terms concerns the distinction between strongly linear control and weakly linear one. The following proposition says that strongly linearity does preserve a fundamental property of linear channels, cf.[5]. Below $\stackrel{l}{\longrightarrow}$ and $\stackrel{l}{\Longrightarrow}$ denote standard labelled transition for the $\pi$-terms.

**Proposition 5.** *Let* $\vdash^{c}_{\mathrm{I}} P \rhd x^{\tau} \otimes \vec{z}^{\vec{\rho}}$ *where* $\tau = ((\vec{\sigma})^{\uparrow})^!$ *and* $\mathsf{md}(\rho_i) = \uparrow'$ *with each* $\rho_i$ *unary. Then* $P \stackrel{x(y)}{\longrightarrow} P'$ *such that either* $P' \stackrel{\overline{z_i}(\vec{w})}{\Longrightarrow}$ *or* $P' \stackrel{\overline{y}(\vec{v})}{\Longrightarrow}$.

There is an obvious branching version. Thus if you ask a question to a replicated agent and expects an answer at a linear channel, then either you can receive something at that channel, or, if there are free control channels, you will get something at one of them. In particular if you have no free control channels, you are guaranteed to receive an output at a strongly linear channel, even though this may be a control (weakly linear) output. The proof precisely follows the one given in [5], using the strong normalisability w.r.t. the normal reduction (which directly follows from Proposition 4).

# 5 Embedding LLP (2)

## 5.1 LLP: the full system

The full LLP, introduced and studied by Laurent [4], is a result of not only restricting Linear Logic by the use of polarity (as in LLP⁻) but also relaxes the rules of exponentials by allowing any negative formulae to be weakened and contracted. Accordingly we have a more liberal rule for "of course". Concretely, LLP arises from LLP⁻ by:

(1) replacing ?P in $(?_w)$ and $(?_c)$ with N; and:
(2) replacing ?Γ in (!) with $\mathcal{N}$.

The construction immediately suggests its link with weak linearity. We shall see this is indeed so in the next subsection.

## 5.2 Embedding LLP in the linear $\pi$-calculus with control

The interpretation of the full LLP-formulae in the linear-control channel types is given as follows.

$$
\begin{array}{rclrcl}
X^\bullet & \stackrel{\mathrm{def}}{=} & ()^{\uparrow'} & (X^{\perp^\bullet}) & \stackrel{\mathrm{def}}{=} & ()^{\downarrow'} \\
(\mathrm{N}⅋\mathrm{M})^\bullet & \stackrel{\mathrm{def}}{=} & ((\mathrm{N}^\bullet\mathrm{M}^\bullet)^{\downarrow})^{\uparrow'} & (\mathrm{P}\otimes\mathrm{Q})^\bullet & \stackrel{\mathrm{def}}{=} & ((\mathrm{P}^\bullet\mathrm{Q}^\bullet)^{\uparrow})^{\downarrow'} \\
(\mathrm{N}\&\mathrm{M})^\bullet & \stackrel{\mathrm{def}}{=} & ([\mathrm{N}^\bullet\&\mathrm{M}^\bullet]^{\downarrow})^{\uparrow'} & (\mathrm{P}\oplus\mathrm{Q})^\bullet & \stackrel{\mathrm{def}}{=} & ([\mathrm{P}^\bullet\oplus\mathrm{Q}^\bullet]^{\uparrow})^{\downarrow'} \\
(\mathrm{N}\&\mathrm{M})^\bullet & \stackrel{\mathrm{def}}{=} & ([\mathrm{N}^\bullet\&\mathrm{M}^\bullet]^{\downarrow'})^{\uparrow'} & (\mathrm{P}\oplus\mathrm{Q})^\bullet & \stackrel{\mathrm{def}}{=} & ([\mathrm{P}^\bullet\oplus\mathrm{Q}^\bullet]^{\uparrow'})^{\downarrow'} \\
(?\mathrm{P})^\bullet & \stackrel{\mathrm{def}}{=} & (\mathrm{P}^\bullet)^{?} & (!\mathrm{N})^\bullet & \stackrel{\mathrm{def}}{=} & (\mathrm{N}^\bullet)^{!} \\
(\uparrow\mathrm{P})^\bullet & \stackrel{\mathrm{def}}{=} & (\mathrm{P}^\bullet)^{\uparrow'} & (\downarrow\mathrm{N})^\bullet & \stackrel{\mathrm{def}}{=} & (\mathrm{N}^\bullet)^{\downarrow'}
\end{array}
$$

We keep the intermediate strong linearity, since this makes the embedding more faithful (in fact it is better to keep it as "purely linear" whose algebra does not combine weak linearity). Note all outermost non-exponential types are weakly linear in the interpretation.

The term assignment is precisely the same, except for the replacement of the symbols mentioned in Section 5.1. We write the sequent of this term calculus $P \vdash_{\mathsf{LLP}} \Gamma$. The embedding $\Gamma^\bullet$ remains precisely the same as $\Gamma^\circ$, except we use $\mathrm{A}^\bullet$ instead of $\mathrm{A}^\circ$. By the previous result and by inspecting the typing rules for linear control, we immediately observe:

**Theorem 2.** (soundness) $P \vdash_{\mathsf{LLP}} \Gamma$ *implies* $\vdash^{c}_{io(\Gamma)} P \triangleright \Gamma^{\bullet}$.

By Proposition 4 we conclude:

**Corollary 2.** (cut elimination in LLP) *The cut rule is redundant in* LLP*; the cut elimination procedure always terminates with a unique normal form.*

Again the full completeness does not hold for the same reason as we discussed in Section 3, for which we have the same methods to close the gap. In both settings, we find significant connections between two paradigms as well as differences, both technically and in orientation. These differences and similarities together suggest a few interesting topics for further research.

London, June 11-18, 2001.

## References


1. Berger, M. Honda, K. and N. Yoshida. *Sequentiality and the π-Calculus.* TLCA'01, 2001. The full version to appear as a QMW technical report, Queen Mary and Westfield.
2. Girard, J.-Y., *Linear Logic.* TCS, Vol. 50, 1–102, 1987.
3. Laurent, O. *Polarised Proof-nets and λμ-calculus.* TCS, to appear.
4. Laurent, O. *Polarised Games for Classical Logic.* Typescript, 2001.
5. Yoshida, N., Berger, M. and Honda, K., *Strong Normalisation in the π-Calculus*, LINCS'01, 2001. A full version appeared as a MCS technical report, Leicester University.


## Appendix A: Typing rules for branching

(Bra$^{\downarrow}$)
$$\frac{\vdash_{\mathtt{O}} P_i \triangleright C_i^{-x} \qquad C_i/\vec{y}_i^{\,\vec{\tau}_i} = \uparrow A \otimes ?B}{\vdash_{\mathtt{I}} x[\&_i(\vec{y}_i).P_i] \triangleright (x^{[\&_i \vec{\tau}_i]} \xrightarrow{\downarrow} A) \otimes B}$$

(Bra$^{!}$)
$$\frac{\vdash_{\mathtt{O}} P_i \triangleright C_i^{-x} \qquad C_i/\vec{y}_i^{\,\vec{\tau}_i} = ?A}{\vdash_{\mathtt{I}} !\, x[\&(\vec{y}_i).P] \triangleright x^{[\&_i \vec{\tau}_i]} \xrightarrow{!} A}$$

(Sel)
$$\frac{\vdash_{\mathtt{I}} P \triangleright C \qquad C/\vec{y}^{\,\vec{\tau}_i} = A \asymp x^{[\oplus(\vec{\tau})^{p_{\mathrm{O}}}}}{\vdash_{\mathtt{O}} \overline{x}(\vec{y})P \triangleright A \odot x^{(\vec{\tau})^{p_{\mathrm{O}}}}}$$

## Appendix B: Full completeness for LLP and LLP$^-$ with rotation

We briefly discuss how full completeness is obtained for the "rotating" variant of LLP$^-$discussed in Section 3.3. By precisely the same method we can prove the full completeness for the embedding of LLP with rotation in the linear $\pi$-calculus with control, which we omit.

The proof is by the length of the derivation of $\Gamma^{\circ}$ where $\Gamma$ is a LLP$^-$-derivable. For the base case, we have axioms for atomic formulae. The tightest possible correspondence is obtained if we consider a forwarder $[x \triangleright y]$ as a constant, with action type $x^X \to y^{\overline{X}}$ and the reduction rule $[x \triangleright y]|[y \triangleright z] \longrightarrow [x \triangleright z]$ (in this case we interpret $X$ as $X$ itself: if we use the encoding as given in Section 3.1, we need to use a permutation of prefixes as discussed in [5] when we deal with an input prefix of form $x.P$ in the induction step).

For induction step, We only discuss ⅋', ⊗' and ?. Other cases are similar. Suppose $\vdash_{io(\Gamma)} P \triangleright \Gamma^\circ$ with the length of derivation more than one. First if the last rule applied adds an input at $x$ with $\Gamma(x) = \mathrm{N}_1 ⅋' \mathrm{N}_2$, then $P \stackrel{\text{def}}{=} \overline{x}(y_1 y_2)Q$. By Proposition 3 we have $\vdash_{\mathtt{0}} Q \triangleright (\Gamma/x, \mathrm{N}_1, \mathrm{N}_2)^\circ$. By induction hypothesis we have $Q \vdash \Gamma/x, \mathrm{N}_1, \mathrm{N}_2$, from which we know $P \vdash \Gamma$, as required.

Next assume the last applied rule adds an output at $x$ and $\Gamma(x) = \mathrm{Q}_1 \otimes' \mathrm{Q}_2$. Then $P \equiv \overline{x}(y_1 y_2)P'$. By Proposition 3, we know $P' \equiv Q_1 | Q_2$ where $Q_1$ and $Q_2$ are both connected. Thus we know, for some $\Delta_{1,2}$ such that $\Delta_1 \odot \Delta_2 = (\Gamma/x), \mathrm{Q}_1, \mathrm{Q}_2$, we have $\vdash_{io(\Delta_i)} Q_i \triangleright \Delta_i^\circ$ $(i = 1, 2)$. We then use induction as above.

Finally suppose $\vdash_{io(\Gamma)} P \triangleright \Gamma^\circ$ where the last applied rule is (Out) which adds a ?-output, so that we can necessarily write $\Gamma(x) = ?\mathrm{P}$. Then $P \stackrel{\text{def}}{=} \overline{x}(y)Q$. If $x$ does not occur in $Q$ we simply use induction. If $x$ does occur in $Q$, let $P' \stackrel{\text{def}}{=} \overline{x}(y)Q'$ with $Q' \stackrel{\text{def}}{=} Q\{z/x\}$ with $z$ fresh. By Proposition 3, we know $\vdash_{io(\Delta)} Q' \triangleright \Delta^\circ$ where $\Delta = \Gamma\{z/x\} \otimes y : \mathrm{Q}$. Since $Q'$ can be typed with the same length as $Q$, we have $Q' \vdash \Delta'$. We can now apply $(?_d)$ to have $P'$ and then apply $(?_c)$ to obtain $P \vdash \Gamma$, as required.