Functions as proofs as processes

Emmanuel Beffara

Functions as proofs as processes

Emmanuel Beffara

Institut de Mathématiques de Luminy
UMR6206, Université Aix-Marseille II & CNRS

January 5, 2007

Abstract. This paper presents a logical approach to the translation of functional calculi into concurrent process calculi. The starting point is a type system for the π-calculus closely related to linear logic. Decompositions of intuitionistic and classical logics into this system provide type-preserving translations of the λ- and $\lambda\mu$-calculus, both for call-by-name and call-by-value evaluation strategies. Previously known encodings of the λ-calculus are shown to correspond to particular cases of this logical embedding. The realisability interpretation of types in the π-calculus provides systematic soundness arguments for these translations and allows for the definition of type-safe extensions of functional calculi.

1 Introduction

The π-calculus was introduced in the late 1980's as a core model of concurrent computation, in the same way as the λ-calculus is a core model of functional computation. Soon afterwards, Milner showed in the seminal paper *Functions as processes* [14] that the λ-calculus could be precisely encoded into the π-calculus. Around the same time, Girard introduced linear logic as a logic to study fine properties of denotational models of intuitionistic logic. Indeed the ideas behind it led to significant insights on the structure and semantics of the λ-calculus and functional computation, along the lines of the *functions as proofs* slogan. It might not be a coincidence that these two contributions appeared at the same time, and intuitions from one have been seen in the other from the beginning. Formal connections appeared some years later, giving formal ground to the *proofs as processes* idea, in particular in work by Abramsky [1, 2] and in a notable contribution by Bellin and Scott as an encoding of proof nets in the π-calculus [5].

The purpose of this paper is to present a formal way to make these pieces fit together. We use a recent version of the proofs-as-processes approach [4, 3] as a way to make a link between a form of π-calculus (with more symmetry and expressiveness) and a form of linear logic (with the significant difference that formulas have arities). In this framework, we adapt previous work by Danos, Joinet and Schellinx on the translation of classical logic into linear logic [8, 9]. We show that, when considering each logic as a type system, we can extract a family of typed translations of the λ- and $\lambda\mu$-calculi into the π-calculus. The now familiar duality [7] between call-by-name and call-by-value appears clearly in our system, moreover several previously known translations are shown to fit in as particular instances of the general technique.

Our type system for the π-calculus was developed by realisability as a logic of behaviours of concurrent processes. We show that this realisability construction can be used to prove properties of the considered execution models of the λ-calculus. We also argue that realisability provides a way to introduce new constructs in functional calculi while keeping the type system semantically correct.

Parallel composition and scoping:

$$p \mid q \equiv q \mid p \qquad (p \mid q) \mid r \equiv p \mid (q \mid r) \qquad p \mid 1 \equiv p$$
$$(\boldsymbol{\nu}x)(\boldsymbol{\nu}y)p \equiv (\boldsymbol{\nu}y)(\boldsymbol{\nu}x)p \qquad (\boldsymbol{\nu}x)(p \mid q) \equiv p \mid (\boldsymbol{\nu}x)q \quad \text{if } x \notin \mathrm{fv}(p) \qquad (\boldsymbol{\nu}x)1 \equiv 1$$

Equators:

$$1 \equiv x{=}x \qquad x{=}y \equiv y{=}x \qquad x{=}y \mid p[x/z] \equiv x{=}y \mid p[y/z]$$

Replication and reduction:

$$!\alpha.p \equiv \alpha.(p \mid !\alpha.p) \qquad u(\vec{x}).p \mid \bar{u}(\vec{x}).q \to (\boldsymbol{\nu}\vec{x})(p \mid q)$$

Table 1: Structural congruence and reduction.

2 Framework

2.1 The calculus

The concurrent calculus we use, hereafter named $\pi^{=}$-calculus, is a formulation of π-calculus with explicit fusions (à la Gardner and Wischik [11]) with binding input and output. We assume an infinite set $\boldsymbol{N}$ of names, ranged over by the letters u, v, x, y, z. The calculus is generated by the following grammar:

actions:	$\alpha ::= u(x_1 \dots x_n)$	input
	$\bar{u}(x_1 \dots x_n)$	binding output
processes:	$p, q ::= \alpha.p, \quad !\alpha.p$	linear action, guarded replication
	$1, \quad p \mid q, \quad (\boldsymbol{\nu}x)p$	inaction, parallel composition, hiding
	$x{=}y$	name unification

The operational semantics of the calculus is defined as a reduction relation up to structural congruence, with the rules in table 1. The reduction relation is the smallest relation $\to$ that is closed under structural congruence, parallel composition and hiding and that contains $u(\vec{x}).p \mid \bar{u}(\vec{x}).q \to (\boldsymbol{\nu}\vec{x})(p \mid q)$. We consider a strong bisimilarity relation $\cong$ whose precise definition (that can be found in the appendix) uses a labelled transition system. The point is that $\cong$ is a congruent equivalence such that $p \cong q$ implies that for each reduction $p \to p'$ there is a reduction $q \to q'$ with $p' \cong q'$.

We use this calculus instead of a more standard form of π-calculus because it provides a clear distinction between synchronisation and name substitution. It also allows a cleaner type system. By combining binding actions and equators, we get usual non-binding actions with their usual semantics, by defining

$$\bar{u}\langle x_1 \dots x_n\rangle := \bar{u}(y_1 \dots y_n).(x_1{=}y_1 \mid \dots \mid x_n{=}y_n)$$

2.2 The type system: linear logic with arities

We assume a set $\boldsymbol{V}$ of type variables, ranged over by X or Y. The language of formulas is generated by the following grammar:

$$A, B ::= X \mid X^{\perp} \mid A \otimes B \mid A \parr B \mid {\downarrow}A \mid {\uparrow}A \mid !A \mid ?A \mid \exists X.A \mid \forall X.A$$

Each variable is supposed to have a fixed arity. Given an arity function $\mathrm{ar} : \boldsymbol{V} \to \mathbb{N}$, the arity of a formula is defined as

$$\mathrm{ar}(\dagger A) := 1 \qquad \text{with } \dagger \in \{\uparrow, \downarrow, ?, !\}$$
$$\mathrm{ar}(\exists X.A) := \mathrm{ar}(\forall X.A) := \mathrm{ar}(A)$$
$$\mathrm{ar}(A \otimes B) := \mathrm{ar}(A \parr B) := \mathrm{ar}(A) + \mathrm{ar}(B)$$

The dual (or linear negation) is the involution $(\cdot)^\perp$ defined as $X^{\perp\perp} := X$ and

$$(A \otimes B)^\perp := A^\perp \parr B^\perp \qquad\qquad (\downarrow A)^\perp := \uparrow(A^\perp)$$
$$(\forall X.A)^\perp := \exists X.(A^\perp) \qquad\qquad (!A)^\perp := ?(A^\perp)$$

A type Γ is a sequence $\vec{x}_1 : A_1, \ldots, \vec{x}_n : A_n$ where each A_i is a formula and each $\vec{x}_i$ is a sequence of names of length $\mathrm{ar}(A_i)$. All the names occurring in all the $\vec{x}_i$ must be distinct. $\uparrow\Gamma$ denotes a sequent where all formulas have the form $\uparrow A$ or $?A$, and $?\Gamma$ denotes a sequent where all formulas have the form $?A$. A typing judgement is written $p \vdash \Gamma$, where p is a process and Γ is a type. A process p has type Γ if $p \vdash \Gamma$ is derivable by the rules of table 2.

We call LLa (for linear logic with arities) this logical system. The inference rules are those of multiplicative-exponential linear logic (MELL), extended with the linear modalities $\uparrow$ and $\downarrow$. The main difference is in the rule for the existential quantifier: $\exists X.A$ can be deduced from $A[B/X]$ only when X and B have the same arity. As a consequence, although the language of MELL is a subset of our language of types, provability of a sequent Γ in MELL is not equivalent to provability of Γ in LLa.

2.3 Second-order $\lambda\mu$-calculus

Our model of functional computation is the $\lambda\mu$-calculus [15]. We assume an infinite set of λ-variables ranged over by x, y and an infinite set of μ-variables ranged over by α, β. Terms are generated by the following grammar:

$$M, N ::= x \mid \lambda x.M \mid (M)N \mid \mu\alpha[\beta]M$$

Thus we consider the version of the calculus where $\mu\alpha$ and $[\beta]$ cannot appear separately. The language of types is minimal second-order logic, i.e.

$$A, B ::= X \mid A \to B \mid \forall X.A$$

A typing judgement has the form $\Gamma \vdash M : A \mid \Delta$ where Γ is a sequence of type assignments $x : A$ for distinct λ-variables and Δ is a sequence of type assignments $\alpha : A$ for distinct μ-variables. The typing rules are given in table 3. The intuitionistic fragment, i.e. system F, is the fragment of this calculus where $\mu\alpha[\beta]$ is never used and where the Δ part is always empty.

3 Simply typed λ-calculus and head linear reduction

The basis of linear logic is the decomposition of intuitionistic implication $A \to B$ into an linear implication and an exponential modality, as $!A \multimap B$. The idea is that linear implication $A \multimap B = A^\perp \parr B$ is the actual implication, while the modalities $!A$ and $?A$ control weakening and contraction. In this section, we describe the operational meaning of this embedding.

Axiom and cut:

$$\frac{}{u_1{=}v_1 \mid \ldots \mid u_k{=}v_k \vdash \vec{u} : X^\perp, \vec{v} : X} \qquad \frac{p \vdash \Gamma, \vec{x} : A \qquad q \vdash \vec{x} : A^\perp, \Delta}{(\boldsymbol{\nu}\vec{x})(p \mid q) \vdash \Gamma, \Delta}$$

Multiplicatives:

$$\frac{p \vdash \Gamma, \vec{x} : A \qquad q \vdash \Delta, \vec{y} : B}{p \mid q \vdash \Gamma, \Delta, \vec{x}\vec{y} : A \otimes B} \qquad \frac{p \vdash \Gamma, \vec{x} : A, \vec{y} : B}{p \vdash \Gamma, \vec{x}\vec{y} : A \parr B}$$

Actions:

$$\frac{p \vdash \Gamma, \vec{x} : A}{\bar{u}(\vec{x}).p \vdash \Gamma, u : \uparrow A} \quad \frac{p \vdash \Gamma, \vec{x} : A}{\bar{u}(\vec{x}).p \vdash \Gamma, u : ?A} \quad \frac{p \vdash \uparrow\Gamma, \vec{x} : A}{u(\vec{x}).p \vdash \uparrow\Gamma, u : \downarrow A} \quad \frac{p \vdash ?\Gamma, \vec{x} : A}{!u(\vec{x}).p \vdash ?\Gamma, u : !A}$$

Exchange, contraction and weakening:

$$\frac{p \vdash \Gamma, \vec{x} : A, \vec{y} : B, \Delta}{p \vdash \Gamma, \vec{y} : B, \vec{x} : A, \Delta} \qquad \frac{p \vdash \Gamma, u : ?A, v : ?A}{p[w/u, v] \vdash \Gamma, w : ?A} \qquad \frac{p \vdash \Gamma}{p \vdash \Gamma, u : ?A}$$

Quantifiers:

$$\frac{p \vdash \Gamma, \vec{x} : A \qquad X \notin \mathrm{fv}(\Gamma)}{p \vdash \Gamma, \vec{x} : \forall X.A} \qquad \frac{p \vdash \Gamma, \vec{x} : A[B/X] \qquad \mathrm{ar}(B) = \mathrm{ar}(X)}{p \vdash \Gamma, \vec{x} : \exists X.A}$$

Table 2: Typing rules for the $\pi^=$-calculus.

Intuitionistic rules:

$$\frac{}{\Gamma, x : A \vdash x : A \mid \Delta} \qquad \frac{\Gamma, x : A \vdash M : B \mid \Delta}{\Gamma \vdash \lambda x.M : A \to B \mid \Delta}$$

$$\frac{\Gamma \vdash M : A \to B \mid \Delta \qquad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash (M)N : B \mid \Delta}$$

Quantifiers:

$$\frac{\Gamma \vdash M : A \mid \Delta \qquad X \notin \mathrm{fv}(\Gamma, \Delta)}{\Gamma \vdash M : \forall X.A \mid \Delta} \qquad \frac{\Gamma \vdash M : \forall X.A \mid \Delta}{\Gamma \vdash M : A[B/X] \mid \Delta}$$

Control:

$$\frac{\Gamma \vdash M : B \mid \alpha : A, \beta : B, \Delta}{\Gamma \vdash \mu\alpha[\beta]M : A \mid \beta : B, \Delta}$$

Table 3: Typing rules for the $\lambda\mu$-calculus.

1 **Definition.** Let LJ_0 be the language of formulas generated by variables and $\to$ as the only connective. The translation A^0 of a formula A is defined as

$$X^0 := X \qquad\qquad (A \to B)^0 := {!A^0} \multimap B^0$$

where each variable of LJ_0 is mapped to a variable of arity 1 in LLa.

Here propositional variables are considered as base types of arity 1. A functional type $A_1 \to \cdots \to A_n \to X$ is thus translated into a formula of arity $n+1$. The translation of formulas naturally induces a translation of type derivations.

2 **Definition.** The translation of a simply typed λ-term M at type A on channels $\vec{y}$ (with $|\vec{y}| = \mathrm{ar}(A^0)$) is the process $[\![M]\!]^A\vec{y}$ defined as

$$\begin{aligned} [\![x]\!]^A\vec{y} &:= \bar{x}\langle\vec{y}\rangle \\ [\![\lambda x.M]\!]^{A\to B}x\vec{y} &:= [\![M]\!]^B\vec{y} \\ [\![(M)N]\!]^B\vec{y} &:= (\boldsymbol{\nu}x)([\![M]\!]^{A\to B}x\vec{y} \mid {!x(\vec{z})}.[\![N]\!]^A\vec{z}) \end{aligned}$$

The soundness and faithfulness of this translation are easily checked:

3 **Proposition.** *A judgement $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ is derivable in* LJ_0 *if and only if $[\![M]\!]^B\vec{y} \vdash x_1 : ?(A_1^0)^\perp, \ldots, x_n : ?(A_n^0)^\perp, \vec{y} : B^0$ is derivable in* LLa.

Let us now study the operational meaning of the translation. Remark that, up to structural congruence, redexes can be permuted without affecting the translation, i.e. the translation captures σ-equivalence [16]. Subsequently, we get that τ-transitions in the translations correspond to what is known as head linear reduction [10]. We briefly recall the definition of these two notions:

4 **Definition.** σ-equivalence is the congruence over λ-terms generated by

$$(\lambda x.M)NP =_\sigma (\lambda x.(M)P)N \qquad\qquad (\lambda xy.M)N =_\sigma \lambda y.(\lambda x.M)N$$

with $x \notin \mathrm{fv}(P)$ and $y \notin \mathrm{fv}(N)$. Any λ-term M can be normalised as

$$M =_\sigma \lambda x_1 \ldots x_k(\lambda y_1 \ldots y_n.(x)M_1 \ldots M_p)N_1 \ldots N_n$$

Head linear reduction is the relation over σ-equivalence classes generated by

$$\begin{aligned} &\lambda x_1 \ldots x_k.(\lambda y_1 \ldots y_n.(y_i)M_1 \ldots M_p)N_1 \ldots N_n \\ &\qquad\qquad \to \lambda x_1 \ldots x_k.(\lambda y_1 \ldots y_n.(N_i)M_1 \ldots M_p)N_1 \ldots N_n \end{aligned}$$

5 **Proposition.** *For any simply typed λ-term $\Gamma \vdash M : A$, $[\![M]\!]^A\vec{y}$ is bisimilar to M for head linear reduction.*

Proof. First note that for M and N of type A, if $M =_\sigma N$ then $[\![M]\!]^A\vec{y} \equiv [\![N]\!]^A\vec{y}$, so we can consider terms up to σ-equivalence. Consider a typed term $\Gamma \vdash M : A$. By σ-equivalence we assume that M is written $\lambda\vec{x}.(\lambda\vec{y}.(x)\vec{M})\vec{N}$ with $|\vec{y}| = |\vec{N}|$. Call A_i the type of each x_i, B_i the type of each y_i and N_i (these are the same since M is well typed), and call C_i the type of each M_i. Thus we have $A = A_1 \ldots A_k \to B$ and x has type $C_1 \ldots C_p \to B$. Then we have

$$[\![M]\!]^A\vec{x}\vec{z} = (\boldsymbol{\nu}\vec{y}\vec{u})\big(\bar{x}\langle\vec{u}\vec{z}\rangle \mid \textstyle\prod_{i=1}^p [\![u_i = M_i]\!]^{C_i} \mid \prod_{j=1}^n [\![y_j = N_j]\!]^{B_j}\big)$$

with $[\![x = T]\!]^A := {!x(\vec{y})}.[\![T]\!]^A\vec{y}$. The only possible reduction in this process is $\bar{x}\langle\vec{u}\vec{z}\rangle$ interacting with one of the $[\![u_i = M_i]\!]^{C_i}$ or $[\![y_j = N_j]\!]^{B_j}$. By construction x cannot be one of the u_i, so

$[\![M]\!]^A\vec{x}\vec{z}$ has a τ-transition if and only if $x = y_j$ for some j. In this case, we can remark that the following reduction holds:

$$\bar{y}_j\langle\vec{u}\vec{z}\rangle \mid [\![y_j = N_j]\!]^{B_j} \to [\![N_j]\!]^{B_j}\vec{u}\vec{z} \mid [\![y_j = N_j]\!]^{B_j}$$

Putting this reduction in context, we get that the reduct of $[\![M]\!]^A\vec{x}\vec{z}$, up to structural congruence, is $[\![\lambda\vec{x}.(\lambda\vec{y}.(N_j)\vec{M})\vec{N}]\!]^A\vec{x}\vec{z}$. Therefore, τ-transitions in translations of λ-terms strictly correspond to head linear reductions in the terms. □

Interestingly, this translation was first described by Hyland and Ong as a syntax for strategies in a game semantics of PCF [13], thus with different (but clearly related) arguments.

4 System F and modal translations

The translation presented above is remarkably light. However, the arity of the translation of a term depends on its type, and as a consequence polymorphism in the style of system F does not hold. Modal translations [8] are a generalisation of the standard embedding of intuitionistic logic into linear logic, which allow full polymorphism by providing a type-independent (and type-safe) translation.

6 **Definition.** A generalised modality is a word γ over $\{\uparrow, \downarrow, ?, !\}$. The dual of γ is the modality $\bar{\gamma}$ such that $(\gamma X)^\perp = \bar{\gamma}X^\perp$. A modal translation of LK into LLa is defined by a pair (γ, δ) of generalised modalities. The translation A^* of a formula A is defined as

$$X^* := X \qquad (A \to B)^* := \gamma(A^*) \multimap \delta(B^*) \qquad (\forall X.A)^* := \forall X.(A^*)$$

where each variable of LK is mapped to a variable of arity 2 in LLa. For $\Gamma = \{x_i : A_i\}_{1\leqslant i\leqslant n}$, define $\Gamma^* := \{x_i : A_i^*\}_i$ and $\Gamma^{*\perp} := \{x_i : (A_i^*)^\perp\}_i$. For a generalised modality γ, define $\gamma\Gamma := \{x_i : \gamma A_i\}_i$. A type $\Gamma \vdash A \mid \Delta$ is translated at a channel u into the type $\bar{\gamma}\Gamma^{*\perp}, u : \delta A^*, \delta\Delta^*$. A modal translation (γ, δ) is valid if $\Gamma \vdash A \mid \Delta$ holds if and only if $\vdash_{\mathrm{LLa}} \bar{\gamma}\Gamma^{*\perp}, \delta A^*, \delta\Delta^*$ holds.

An important fact needs to be stressed: in the source language LK, any variable can be substituted by any formula. On the other hand, in the target language LLa, a variable can only be substituted by a formula of the same arity. Note that a translation commutes with substitution, i.e. $(A[B/X])^* = A^*[B^*/X]$, if and only if variables are preserved, therefore any translation must assign sensible arities to variables. For this substitution to be correct in any case, we must ensure that the arity of A^* is independent from A. This condition is satisfied if and only if neither γ nor δ is empty, and then $\mathrm{ar}(A^*) = 2$ for any A.

4.1 General translation

7 **Definition.** Given a non-empty generalised modality γ and names u and $\vec{x}$, define the *protocol* $\gamma u(\vec{x}).p$ as $\downarrow u(\vec{x}).p := u(\vec{x}).p$, $\uparrow u(\vec{x}).p := ?u(\vec{x}).p := \bar{u}(\vec{x}).p$, and inductively $\gamma\dagger u(\vec{x}).p := \gamma u(v).\dagger v(\vec{x}).p$ for a fresh name v. In the case of the empty modality ε, let $\varepsilon u(x).p := p[u/x]$, and $\varepsilon u(\vec{x}).p$ is undefined for $|\vec{x}| \neq 1$.

Note that protocols are typed in the expected way: $p \vdash \Gamma, \vec{x} : A$ implies $\gamma u(\vec{x}).p \vdash \Gamma, u : \gamma A$. If γ contains ! then the context must be $?\Gamma$, else if γ contains $\downarrow$ then the context must be $\uparrow\Gamma$. For a modal translation (γ, δ) to be valid for classical logic, essentially two conditions are required:

- It must be possible to apply weakening and contraction to formulas $\bar{\gamma}A$ and δA, i.e. γ must start with ! and δ must start with ?.

- For the application rule, it must be possible to deduce a common modality ζ from γ and δ, in a context of $\bar{\gamma}$ and δ modalities, which essentially implies that one of γ, δ must be a suffix of the other.

For a pair (γ, δ) to be valid for intuitionistic logic, contraction and weakening of δ formulas is not required, and contexts only contain $\bar{\gamma}$ formulas.

Let (γ, δ) be a modal translation for which these conditions are satisfied. Let Γ and Δ be types where all formulas start with the modalities $\bar{\gamma}$ or δ. We can deduce the translation of the rules for λ and μ independently of the modalities:

$$\dfrac{\dfrac{p \vdash \Gamma, x : \bar{\gamma}A^{\perp}, v : \delta B}{p \vdash \Gamma, xv : \gamma A \multimap \delta B}}{\delta u(xv).p \vdash \Gamma, u : \delta(\gamma A \multimap \delta B)} \qquad \dfrac{p \vdash \Gamma, u : \delta B, \alpha : \delta A, \beta : \delta B}{p[\beta/u] \vdash \Gamma, \alpha : \delta A, \beta : \delta B}$$

Hence we get

$$[\![\lambda x.M]\!]u := \delta u(xv).[\![M]\!]v \qquad\qquad [\![\mu\alpha[\beta]M]\!]\alpha := [\![M]\!]\beta$$

The formulation of the translation of $\mu\alpha[\beta]M$ is valid since α-conversion can be applied to the variable bound by μ. The fact that $\mu\alpha[\beta]$ does not modify the process in any other way stresses the fact that the μ binder is nothing more than a way to name conclusions of a proof in the sequentialised syntax of λ-calculus.

It is clear that the introduction rule for $\forall$ is not affected by the translation. There is a slight difference for the elimination rule: the λ-calculus is a syntax for natural deduction with intro/elim, while our type system for the π-calculus is a sequent calculus with only introduction rules and an actual cut rule. We can translate the elimination rule for $\forall$ by using an extra cut and axiom:

$$\dfrac{p \vdash \Gamma, u : \forall X.A \qquad \dfrac{u{=}v \vdash u : A[B/X]^{\perp}, v : A[B/X]}{u{=}v \vdash u : \exists X.A^{\perp}, v : A[B/X]}}{(\boldsymbol{\nu} u)(p \mid u{=}v) \vdash \Gamma, v : A[B/X]}$$

By structural congruence we have $(\boldsymbol{\nu} u)(p|u{=}v) \equiv p[v/u]$, hence we can also accept the elimination rule itself in our type system. For the application rule, assume there is a generalised modality ζ of which γ and δ are suffixes, and set γ' and δ' such that $\zeta = \gamma'\gamma = \delta'\delta$. Then the translation of application is:

$$\dfrac{\dfrac{\dfrac{p \vdash \Gamma, v : \delta(\gamma A \multimap \delta B) \qquad \bar{\delta}v\langle xu\rangle \vdash v : \bar{\delta}(\gamma A \otimes (\delta B)^{\perp}), x : (\gamma A)^{\perp}, u : \delta B}{(\boldsymbol{\nu} v)(p \mid \bar{\delta}v\langle xu\rangle) \vdash \Gamma, x : (\gamma A)^{\perp}, u : \delta B}}{\bar{\gamma}'z(x).(\boldsymbol{\nu} v)(p \mid \bar{\delta}v\langle xu\rangle) \vdash \Gamma, z : (\zeta A)^{\perp}, u : \delta B} \qquad \dfrac{q \vdash \Delta, w : \delta A}{\delta'z(w).q \vdash \Delta, z : \zeta A}}{(\boldsymbol{\nu} z)(\bar{\gamma}'z(x).(\boldsymbol{\nu} v)(p \mid \bar{\delta}v\langle xu\rangle) \mid \delta'z(w).q) \vdash \Gamma, \Delta, u : \delta B}$$

As explained above, one of γ, δ must be a suffix of the other, so one of γ', δ' must be empty. We thus have two cases for the axiom, depending on which one it is:

$$\dfrac{\dfrac{}{u'{=}u \vdash u' : (\delta A)^{\perp}, u : \delta A}}{\bar{\delta}'x\langle u\rangle \vdash x : (\gamma A)^{\perp}, u : \delta A} \qquad \dfrac{\dfrac{}{x{=}x' \vdash x : (\gamma A)^{\perp}, x' : \gamma A}}{\gamma' u\langle x\rangle \vdash x : (\gamma A)^{\perp}, u : \delta A}$$

When both γ' and δ' are empty, these cases collapse into $u{=}x \vdash x : (\gamma A)^{\perp}, u : \delta A$.

γ and δ are given, γ', δ' are such that $\gamma'\gamma = \delta'\delta$.

$$
\begin{aligned}
[\![x]\!]^{\gamma\delta}u &:= \begin{cases} u{=}x & \text{if } \gamma = \delta \\ \bar{\delta}' x\langle u\rangle & \text{if } \gamma = \delta'\delta \\ \gamma' u\langle x\rangle & \text{if } \delta = \gamma'\gamma \end{cases} \\
[\![\lambda x.M]\!]^{\gamma\delta}u &:= \delta u(xv).[\![M]\!]^{\gamma\delta}v \\
[\![(M)N]\!]^{\gamma\delta}u &:= (\boldsymbol{\nu} z)\Big(\bar{\gamma}' z(x).(\boldsymbol{\nu} v)\big([\![M]\!]^{\gamma\delta}v \mid \bar{\delta} v\langle xu\rangle\big) \mid \delta' z(w).[\![N]\!]^{\gamma\delta}w\Big) \\
[\![\mu\alpha[\beta]M]\!]^{\gamma\delta}\alpha &:= [\![M]\!]^{\gamma\delta}\beta
\end{aligned}
$$

Table 4: General case translation of $\lambda\mu$ into π.

8 **Definition.** Let (γ, δ) be pair of non-empty generalised modalities. The translation $[\![M]\!]^{\gamma\delta}u$ of a λ-term M is defined inductively by the rules of table 4.

9 **Theorem.** *Let (γ, δ) be a valid modal translation. For any $\lambda\mu$-term M, $\Gamma \vdash M : A \mid \Delta$ is derivable if and only if $[\![M]\!]^{\gamma\delta}u \vdash \bar{\gamma}\Gamma^{*\perp}, u : \delta A^*, \delta\Delta^*$ is derivable.*

Actions in the π-calculus, in particular replications, are blocking. As a consequence, in the standard semantics, there is no reduction inside replications, so the execution of $[\![M]\!]$ does not represent the full β-reduction. In the following sections, we give a detailed description of this execution. As explained above, there are two cases, depending on which of γ, δ is a suffix of the other:

10 **Definition.** A pair of generalised modalities (γ, δ) is called left-handed if δ is a suffix of γ. It is called right-handed if γ is a suffix of δ.

4.2 Call-by-name

Here we consider the left-handed case, i.e. with $\gamma = \delta'\delta$ for some non-empty δ'. As a simplification we consider the case where δ and δ' are simple modalities, one easily checks that the other cases are not significantly different. The validity constraints impose $\delta' = !$, and δ has to be ? for the classical case.

To describe precisely the operation of translated terms, we introduce a new form of term $\sharp M$ and define a continuation K as $M_1 \dots M_k\alpha$ where α is a μ-variable and the M_i are terms. An executable is a pair $M * K$, equivalence $\equiv$ and execution $\rightarrow$ of executables are defined as

$$
\begin{aligned}
(M)N * K &\equiv M * NK & \qquad \sharp M * K &\rightarrow M * K \\
\mu\alpha[\beta]M * K &\equiv M[K/\alpha] * \beta & \qquad \lambda x.M * NK &\rightarrow M[\sharp N/x] * K
\end{aligned}
$$

The substitution $M[M_1 \dots M_n\alpha/\beta]$ is the substitution of every subterm of the form $[\beta]N$ of M by $[\alpha](N)M_1 \dots M_n$. The translation of terms is extended to executables as

$$
\begin{aligned}
[\![\alpha]\!]u &:= \alpha{=}u & \qquad [\![x = M]\!] &:= \delta' x(u).[\![M]\!]u \\
[\![MK]\!]u &:= (\boldsymbol{\nu} vz)(\bar{\delta} u\langle zv\rangle \mid [\![z = M]\!] \mid [\![K]\!]v) & \qquad [\![\alpha = K]\!] &:= [\![K]\!]\alpha \\
[\![\sharp M]\!]u &:= (\boldsymbol{\nu} x)([\![x]\!]u \mid [\![x = M]\!]) & \qquad [\![M * K]\!] &:= (\boldsymbol{\nu} u)([\![M]\!]u \mid [\![K]\!]u)
\end{aligned}
$$

11 **Proposition.** *For any call-by-name executables e_1 and e_2, $e_1 \equiv e_2$ implies $[\![e_1]\!] \cong [\![e_2]\!]$ and $e_1 \rightarrow e_2$ if and only if $[\![e_1]\!] \rightarrow [\![e_2]\!]$.*

Classical call-by-name ($\gamma = \,!?$, $\delta = \,?$):

$$\begin{aligned}
[\![x]\!]u &= \bar{x}\langle u\rangle \\
[\![\lambda x.M]\!]u &= \bar{u}(xv).[\![M]\!]v \\
[\![(M)N]\!]u &= (\boldsymbol{\nu} v)\big([\![M]\!]v \mid !v(xy).(!x(w).[\![N]\!]w \mid y{=}u)\big)
\end{aligned}$$

Intuitionistic call-by-name ($\gamma = \,!\downarrow$, $\delta = \,\downarrow$):

$$\begin{aligned}
[\![x]\!]u &= \bar{x}\langle u\rangle \\
[\![\lambda x.M]\!]u &= u(xv).[\![M]\!]v \\
[\![(M)N]\!]u &= (\boldsymbol{\nu} vx)\big([\![M]\!]v \mid !x(w).[\![N]\!]w \mid \bar{v}\langle xu\rangle\big)
\end{aligned}$$

Classical call-by-value ($\gamma = \,!$, $\delta = \,?!$):

$$\begin{aligned}
[\![x]\!]u &= \bar{u}\langle x\rangle \\
[\![\lambda x.M]\!]u &= \bar{u}(y).!y(xv).[\![M]\!]v \\
[\![(M)N]\!]u &= (\boldsymbol{\nu} w)\Big(!w(x).(\boldsymbol{\nu} v)\big([\![M]\!]v \mid !v(w).\bar{w}\langle xu\rangle\big) \;\Big|\; [\![N]\!]w\Big)
\end{aligned}$$

Intuitionistic call-by-value ($\gamma = \,!$, $\delta = \,!$):

$$\begin{aligned}
[\![x]\!]u &= u{=}x \\
[\![\lambda x.M]\!]u &= !u(xv).[\![M]\!]v \\
[\![(M)N]\!]u &= (\boldsymbol{\nu} vw)\big([\![M]\!]v \mid [\![N]\!]w \mid \bar{v}\langle wu\rangle\big)
\end{aligned}$$

Table 5: Particular cases of translations.

Proof. Remark that the translation $[\![M]\!]u$ of a variable or an abstraction has exactly one transition, labelled by an action on u or on a variable. Similarly, the translation $[\![K]\!]u$ of a continuation either is an equator $u{=}\alpha$ or has a unique transition labelled by an action on u. $[\![\sharp M]\!]u$ has a single transition to a process bisimilar to $[\![M]\!]u$. Then the key of the proof is the remark that bindings correctly implement substitution up to bisimilarity, i.e. $(\boldsymbol{\nu}\alpha)([\![e]\!] \mid [\![\alpha = K]\!]) \cong [\![e[K/\alpha]]\!]$ for any fresh name α, and $(\boldsymbol{\nu}x)([\![e]\!] \mid [\![x = M]\!]) \cong [\![e[\sharp M/x]]\!]$ for any fresh name x. The rule for $\mu\alpha[\beta]$ applies only in the classical case, then $\bar{\delta}$ starts with ! and continuations are replicable. Details can be found in the appendix. ☐

Executing a $\lambda\mu$-term simply means executing it on a continuation α for a fresh variable α, since $[\![M]\!]\alpha \equiv [\![M * \alpha]\!]$. Hence we can summarise this result as:

12 **Theorem.** *Left-handed translations implement call-by-name execution.*

The case for $\gamma = !{\downarrow}$ and $\delta = {\downarrow}$ is an adaptation of the standard $!A \multimap B$ decomposition that allows polymorphism. Operationally, it exactly corresponds to Milner's translation [14]. The case for $\gamma = !?$ and $\delta = ?$ corresponds to the system known as LKT in Danos-Joinet-Schellinx. As far as we know, its operational counterpart in the π-calculus is new. These particular translations are shown in table 5. In the classical case, the application uses an equator $y{=}u$ which is not standard π-calculus, however it can be argued that replacing it by a forwarder $!y(ab).\bar{u}\langle ab\rangle$ does not affect the validity of the translation, although the step-by-step operational description is a bit heavier to formulate.

4.3 Call-by-value

We now consider the right-handed case, i.e. with $\delta = \gamma'\gamma$. As in the previous section, we assume without loss of generality that γ is a single modality, necessarily ! because of the validity constraints. We now have two main choices for γ', namely ? for the classical case and $\uparrow$ for the intuitionistic case. We now have to distinguish values, terms and continuations:

values	$V, W := x \mid \lambda x.V$
terms	$M, N := V \mid (M)N \mid \mu\alpha[K]M \mid V \cdot W$
continuations	$K, L := \alpha \mid KM^f \mid KV^a$

An executable is a pair $K * M$. Equivalence and execution are defined as

$$K * (M)N \equiv KM^f * N \qquad K * \mu\alpha[L]M \equiv L * M[K/\alpha]$$

$$KM^f * V \to KV^a * M \qquad KW^a * V \to K * V \cdot W \qquad K * \lambda x.M \cdot V \to K * M[V/x]$$

A continuation contains functions as unevaluated terms M^f and arguments as values V^a, so arguments are evaluated first. The terms $V \cdot W$ and $\mu\alpha[K]M$ are introduced to get a precise bisimulation. Translations are extended as

$$\begin{aligned}
[\![V]\!]u &:= \gamma' u(x).[\![x = V]\!] \\
[\![(M)N]\!]u &:= (\boldsymbol{\nu} v)([\![v = uM^f]\!] \mid [\![N]\!]v) \\
[\![V \cdot W]\!]u &:= (\boldsymbol{\nu} xy)([\![x = V]\!] \mid [\![y = W]\!] \mid \bar{\gamma} x\langle yu\rangle) \\
[\![x = y]\!] &:= x{=}y \\
[\![x = \lambda y.M]\!] &:= \gamma x(yu).[\![M]\!]u \\
[\![\alpha = \beta]\!] &:= \alpha{=}\beta \\
[\![\alpha = KM^f]\!] &:= (\boldsymbol{\nu} v)(\bar{\gamma}'\alpha(x).(\boldsymbol{\nu} u)([\![M]\!]u \mid \bar{\delta} u\langle xv\rangle) \mid [\![v = K]\!]) \\
[\![\alpha = KV^a]\!] &:= (\boldsymbol{\nu} vx)([\![x = V]\!] \mid \bar{\delta}\alpha\langle xv\rangle \mid [\![v = K]\!]) \\
[\![\mu\alpha[K]M]\!]\alpha &:= (\boldsymbol{\nu}\beta)([\![\beta = K]\!] \mid [\![M]\!]\beta) \\
[\![K * M]\!] &:= (\boldsymbol{\nu} u)([\![u = K]\!] \mid [\![M]\!]u)
\end{aligned}$$

13 **Proposition.** *For any call-by-value executables e_1 and e_2, $e_1 \equiv e_2$ implies $[\![e_1]\!] \cong [\![e_2]\!]$ and $e_1 \to e_2$ if and only if $[\![e_1]\!] \to [\![e_2]\!]$.*

Proof. The proof follows the same principle as in call-by-name. The substitution lemma now states $(\boldsymbol{\nu} x)([\![e]\!] \mid [\![x = V]\!]) \cong [\![e[V/x]]\!]$ where e is an executable, V is a value and x is a λ-variable; the same lemma for μ-variables and continuations also holds. We then remark that translations of terms and continuations always have at most one transition, and the correspondence with the operational semantics above is easily checked. Details can be found in the appendix. □

Given a fresh μ-variable α, once again we get $[\![M]\!]\alpha = [\![\alpha * M]\!]$, hence the semantics above precisely describes the execution of translations of $\lambda\mu$-terms in right-handed translations, which can be summarised as follows:

14 **Theorem.** *Right-handed translations implement call-by-value execution.*

The case for $\gamma = !$ and $\delta = ?!$ corresponds to the system called LKQ in Danos-Joinet-Schellinx. Operationally, we get exactly Honda, Yoshida and Berger's translation [6, 12]. The case for $\gamma = !$ and $\delta = \uparrow!$ is a version of this translation linearised with respect to conclusions. It is actually very close to Milner's encoding of call-by-value λ-calculus [14], which corresponds to the slightly more expensive decomposition $(A \to B)^* = \downarrow(!A^* \multimap \uparrow!B^*)$.

The simplest intuitionistic version is obtained by taking $\gamma = \delta = !$, which is both left- and right-handed. It is easy to check that the operational meaning of this translation is an extension of the call-by-value strategy where functions and arguments can be executed in parallel. These translations are shown in table 5.

5 Realisability interpretations

The previous sections define a family of type-preserving translations of the $\lambda\mu$-calculus into the $\pi^=$-calculus, and provide a detailed description of the operational semantics induced by the translations. Since the operational translations are deduced from simple embeddings of intuitionistic and classical logics into linear logic, we can expect more semantic interpretations.

The soundness of the type system we use for processes is formulated using realisability, as described in the following section.

5.1 Soundness of LLa

For a finite set of names I, a process p has interface I if $\mathrm{fv}(P) \subseteq I$.

15 **Definition.** An observation is a set $\perp$ of processes of empty interface. Given an observation $\perp$, two processes p and q of interface I are orthogonal, written $p \perp q$, if $(\boldsymbol{\nu} I)(p \mid q) \in \perp$. An observation $\perp$ is valid if

- $\perp$ is closed under bisimilarity,
- if p has a unique labelled transition $p \xrightarrow{\tau} p'$ and $p' \perp q$ then $p \perp q$.

If $\mathcal{A}$ is a set of processes of interface I, its orthogonal is the set $\mathcal{A}^\perp := \{p : I \mid \forall q \in \mathcal{A}, p \perp q\}$. A behaviour is a set $\mathcal{A}$ such that $\mathcal{A} = \mathcal{A}^{\perp\perp}$. The complete lattice of behaviours of interface I is noted $\boldsymbol{B}_I$.

Let $(u_i)_{i\in\mathbb{N}}$ be an infinite sequence of pairwise distinct names. Let $\boldsymbol{B}_k := \boldsymbol{B}_{u_1\ldots u_k}$. A valuation of propositional variables is a function ρ that associates, to each variable X of arity k, a behaviour $\rho(X) \in \boldsymbol{B}_k$. Given a valuation ρ, the interpretation of a type A localised at $\vec{x}$, with $|\vec{x}| = \mathrm{ar}(A)$, is the behaviour $[\![\vec{x} : A]\!]\rho$ of interface $\vec{x}$ defined inductively by

$$\begin{aligned}
[\![x_1 \ldots x_n : X]\!]\rho &:= v(X)[x_1/u_1, \ldots, x_n/u_n] \\
[\![\vec{x}\vec{y} : A \otimes B]\!]\rho &:= \{(p \mid q) \mid p \in [\![\vec{x} : A]\!]\rho, q \in [\![\vec{y} : B]\!]\rho\}^{\perp\perp} \\
[\![u : \downarrow A]\!]\rho &:= \{u(\vec{x}).p \mid p \in [\![\vec{x} : A]\!]\rho\}^{\perp\perp} \\
[\![\vec{x} : \exists X^k.A]\!]\rho &:= \big(\textstyle\bigcup_{\mathcal{X}\in\boldsymbol{B}_k} [\![\vec{x} : A]\!](\rho[X := \mathcal{X}])\big)^{\perp\perp}
\end{aligned}$$

and $[\![\vec{x} : A^\perp]\!]\rho := ([\![\vec{x} : A]\!]\rho)^\perp$. Exponential modalities require a more subtle definition: for each name u, define the contraction δ_u over behaviours of interface $\{u\}$ as

$$\delta_u(\mathcal{A}) := \{p[u/v, w] \mid p \in \mathcal{A}[v/u] \mathbin{⅋} \mathcal{A}[w/u]\}^{\perp\perp}$$

where v and w are fresh names. Then, for a behaviour $\mathcal{B}$ of interface $\{x_1 \ldots x_n\}$, define $F_u(\mathcal{B}, \mathcal{X}) := \big([\![u : \uparrow\mathcal{B}]\!] \cup \{1 : u\}^\perp \cup \delta_u(\mathcal{X})\big)^{\perp\perp}$. This operator is obviously monotonic in $\mathcal{X}$, and the interpretation of exponential modalities is defined as a fixed point of it:

$$[\![u : ?A]\!]\rho := \mathrm{lfp}(\mathcal{X} \mapsto F_u([\![\vec{x} : A]\!]\rho, \mathcal{X})) \qquad [\![u : !A]\!]\rho := ([\![u : ?(A^\perp)]\!]\rho)^\perp$$

Finally, a type $\Gamma = \vec{x}_1 : A_1, \ldots, \vec{x}_n : A_n$ is interpreted as

$$[\![\Gamma]\!]\rho := \big\{(p_1 \mid \ldots \mid p_n) \bigm| p_1 \in [\![\vec{x}_1 : A_1]\!]\rho^\perp, \ldots, p_n \in [\![\vec{x}_n : A_n]\!]\rho^\perp\big\}^\perp$$

16 **Definition.** Given an observation, a process p realises a type Γ if $p \in [\![\Gamma]\!]\rho$ for any valuation ρ. This fact is written $p \Vdash \Gamma$.

From the definition of observations and the interpretation of formulas, we easily deduce the adequacy theorem (we do not expose the proof here, a detailed study on this technique can be found in other works by the author [3, 4]):

17 **Theorem.** *If $p \vdash \Gamma$ is derivable, then $p \Vdash \Gamma$ for any observation $\perp$.*

The usual notions of testing fit in our notion of observation, for instance:

18 **Proposition.** *Let ω be a channel, assume ω is not taken into account in interfaces. Define the must-testing observation as $\{p \mid \forall p \to^* q, \exists q \to^* \omega \mid r\}$. Must-testing is a valid observation.*

Properties of typed processes, such as termination or deadlock-freeness, can be obtained by choosing appropriate observations. For instance:

19 **Proposition.** *Let $p \vdash \Gamma$ be a typed process such that any propositional variable occurring in Γ is under a modality. For any reduction $p \to^* p'$ there is a reduction $p' \to^* p''$ such that p'' has a visible action.*

Proof. We use the must-testing observation with a channel ω that does not occur in p. Note that $\omega \in [\![\vec{x} : A]\!]$ for any formula A, hence $u(\vec{x}).\omega \in [\![u : \downarrow A]\!]$. By similar arguments we get $u(\vec{x}).\omega \in [\![u : !A]\!]$, $\bar{u}(\vec{x}).\omega \in [\![u : \uparrow A]\!]$ and $\bar{u}(\vec{x}).\omega \in [\![u : ?A]\!]$. Moreover it is clear that, for $q \in [\![A]\!]$ and $r \in [\![B]\!]$, $(q \mid r) \in [\![A \otimes B]\!]$ and $(q \mid r) \in [\![A \parr B]\!]$. Each name u_i occurring in Γ occurs with a polarity ε_i (depending on the modality that introduces it) and a particular arity. Let $t := \prod_i \bar{u}_i^{\varepsilon_i}(\vec{x}).\omega$, by the above remarks we know that $t \in ([\![\Gamma]\!]\rho)^\perp$ for any valuation ρ. This implies that, for any reduction $p \mid t \to^* p' \mid t$ there is a reduction $p' \mid t \to^* \omega \mid q$. Since ω only occurs in t, this implies that an action in t must be triggered during this reduction. By induction on the typing rules, on proves that if all type variables occur under modalities, no equator in p can relate free names, hence triggering an action in t must be done by an action in a reduct of p'. ☐

20 **Corollary.** *The execution of a typed $\lambda\mu$-term in call-by-name or call-by-value always ends with a λ- or μ-variable in active position.*

Proof. Let $\Gamma \vdash M : A \mid \Delta$ be a typed $\lambda\mu$-term. Using non-divergence as the observation we can prove that $[\![M]\!]\alpha$ has no infinite reduction. Consider a reduction $[\![M]\!]\alpha \to^* p$ with p irreducible. By proposition 19 we deduce that p must have a visible action, and this action can only be on α or a name that occurs in Γ or Δ. Conclude by reasoning on the shape of translations of terms: in call-by-name, executables with visible actions are $x * K$ or $\lambda x.M * \alpha$; in call-by-value they are $K * x \cdot V$ or $\alpha * \lambda x.M$. ☐

5.2 Extending the $\lambda\mu$-calculus

Realisability presents the type system LLa as an axiomatisation of the algebra of process behaviours. This allows for the introduction of new logical connectives and new rules: by semantic means (i.e. by reasoning on the reductions of processes) we can define the interpretation of a connective as an operation on sets of processes. If we prove the adequacy of a new logical rule, we can then use it as a typing rule for processes with the guarantee that any property that is proved by realisability is preserved; this includes termination and deadlock-freeness.

This technique can be used to extend the typed $\lambda\mu$-calculus. As soon as a connective can be translated into LLa (possibly extended as explained above), a translation of the underlying syntax is deduced the same way as for the core calculus, which induces an evaluation strategy. This provides a framework for extending our type-preserving translations, without loosing any of the properties of the translations. We now provide some examples of these ideas.

Product types Products can be added to the $\lambda\mu$-calculus by means of a pair of constructs for introduction and elimination:

$$\frac{\Gamma \vdash M : A \mid \Delta \qquad \Gamma \vdash N : B \mid \Delta}{\Gamma \vdash (M, N) : A \times B \mid \Delta} \quad \frac{\Gamma \vdash M : A \times B \mid \Delta \qquad \Gamma, x : A, y : B \vdash N : C \mid \Delta}{\Gamma \vdash \text{let } x, y = M \text{ in } N : C \mid \Delta}$$

Given a pair (γ, δ), we extend the translation of types by $(A \times B)^* = \gamma A^* \otimes \gamma B^*$. Note that, when γ and δ are not empty, the arity of $(A \otimes B)^*$ is 2, hence polymorphism is preserved. The translation of terms is extended as follows:

$$[\![(M,N)]\!]^{\gamma\delta}u := \delta u(xy).(\delta' x(v).[\![M]\!]^{\gamma\delta}v \mid \delta' y(w).[\![N]\!]^{\gamma\delta}w)$$
$$[\![\text{let } x,y = M \text{ in } N]\!]^{\gamma\delta}u := (\boldsymbol{\nu}v)([\![M]\!]^{\gamma\delta}v \mid \bar{\delta}v(xy).[\![N]\!]^{\gamma\delta}u)$$

In both strategies, let $x,y = M$ in N must reduce M into a pair before evaluating N. The evaluation of the parts of a pair in call-by-value is done in parallel since δ' is empty. We leave to the reader the formulation of precise evaluation rules.

Sum types Sum types in $\lambda\mu$ can be defined as follows (with $i \in \{1,2\}$):

$$\frac{\Gamma \vdash M : A_i \mid \Delta}{\Gamma \vdash \text{inj}_i\, M : A_1 + A_2 \mid \Delta} \qquad \frac{\Gamma \vdash M : A_1 + A_2 \mid \Delta \qquad \Gamma, x_i : A_i \vdash N_i : C \mid \Delta}{\Gamma \vdash \text{case } M \text{ of } \{\text{inj}_i\, x_i \to N_i\} : C \mid \Delta}$$

Decomposing this in linear logic requires the additives $\oplus$ and $\&$. The general rules in LLa are complicated, but here we only need simplified versions:

$$\frac{p \vdash \Gamma, u : {\uparrow}A}{p \vdash \Gamma, uv : {\uparrow}A \oplus {\uparrow}B} \qquad \frac{p \vdash \Gamma, \vec{x} : A \qquad q \vdash \Gamma, \vec{y} : B}{u(\vec{x}).p + v(\vec{y}).q \vdash \Gamma, uv : {\downarrow}A \,\&\, {\downarrow}B}$$

assuming the underlying π-calculus has guarded choice. We get adequacy by defining $[\![uv : A \oplus B]\!]\rho := ([\![u : A]\!]\rho \cup [\![v : B]\!]\rho)^{\perp\perp}$ and interpreting $A \,\&\, B$ by duality. The sum type of $\lambda\mu$ is translated as $(A+B)^* = {\uparrow}\gamma A^* \oplus {\uparrow}\gamma B^*$ (which preserves polymorphism). The translation of terms follows:

$$[\![\text{inj}_i\, M]\!]u := \delta u(a_1 a_2).{\uparrow}\delta' a_i(v).[\![M]\!]v$$
$$[\![\text{case } M \text{ of } \{\text{inj}_i\, x_i \to N_i\}]\!]u := (\boldsymbol{\nu}v)\big([\![M]\!]v \;\big|\; \bar{\delta}v(ab).\textstyle\sum_i a_i(x_i).[\![N_i]\!]u\big)$$

Obviously, in any strategy, the evaluation of case M of $\{\text{inj}_i\, x_i \to N_i\}$ must always reduce M into an inj_i before proceeding.

Subtyping Behaviours of a given interface form a complete lattice, with intersection as the lower bound and bi-orthogonal of the union as the upper bound. Write $\wedge$ and $\vee$ these dual connectives with $\text{ar}(A \wedge B) = \text{ar}(A) = \text{ar}(B)$. This induces subtyping over types, defined as $A \leqslant B$ if $[\![A]\!] \subseteq [\![B]\!]$, and the rules:

$$\frac{p \vdash \Gamma, \vec{x} : A \qquad p \vdash \Gamma, \vec{x} : B}{p \vdash \Gamma, \vec{x} : A \wedge B} \qquad \frac{p \vdash \Gamma, \vec{x} : A}{p \vdash \Gamma, \vec{x} : A \vee B} \qquad \frac{p \vdash \Gamma, \vec{x} : A \qquad A \leqslant B}{p \vdash \Gamma, \vec{x} : B}$$

It is clear that all connectives except negation are increasing for this relation, and that $A \leqslant B$ if and only if $B^\perp \leqslant A^\perp$. By the interpretation of modalities we also get $!A \leqslant {\downarrow}A$ and ${\uparrow}A \leqslant ?A$. Subtyping rules in $\lambda\mu$ can be written as

$$\frac{\Gamma \vdash M : A \mid \Delta \qquad \Gamma \vdash M : B \mid \Delta}{\Gamma \vdash M : A \cap B \mid \Delta} \qquad \frac{\Gamma \vdash M : A \mid \Delta \qquad A \leqslant B}{\Gamma \vdash M : B \mid \Delta}$$

Translations are extended as $(A \cap B)^* = A^* \wedge B^*$. The usual subtyping rules, like $(A \to B) \leqslant (A' \to B')$ if $A' \leqslant A$ and $B \leqslant B'$, hold through translation.

Fix points The fact that behaviours form complete lattices also guarantees that any increasing function over behaviours of a fixed interface have (least and greatest) fix points. We can thus extend LLa with dual constructs $\mu X.A$ and $\nu X.A$, with the constraints that $\mathrm{ar}(X) = \mathrm{ar}(A)$ and that X does not occur as $X^\perp$ in A. The typing rules for fix points are rather technical to formulate, mainly because the proper rule for $\nu X.A$ requires the introduction of a recursion operator in the π-calculus. Fix points in the types for $\lambda\mu$-calculus would be simply translated as $(\mu X.A)^* = \mu X.(A^*)$. The constraint that permits polymorphism à la system F also allows this fix point to be used for any A where X only occurs positively.

These various extensions to the type system can be freely combined. Other extensions, notably with concurrent primitives, could be studied in a similar way. However, for this purpose, it seems necessary to enforce serious linearity in the calculus. This fits naturally in our type system for the π-calculus but it is incompatible with full control in the style we get from translations of full classical logic. Precise studies of this idea are deferred to further work.

References

[1] Samson Abramsky. *Computational interpretations of linear logic.* Theoretical Computer Science, 111(1–2):3–57, 1993.

[2] Samson Abramsky. *Proofs as processes.* Theoretical Computer Science, 135(1):5–9, 1994.

[3] Emmanuel Beffara. *Logique, réalisabilité et concurrence.* PhD thesis, Université Paris 7, dec 2005.

[4] Emmanuel Beffara. *A concurrent model for linear logic.* In 21st International Conference on Mathematical Foundations of Programming Semantics (MFPS), volume 155, pages 147–168, may 2006.

[5] Gianluigi Bellin and Philip J. Scott. *On the π-calculus and linear logic.* Theoretical Computer Science, 135(1):11–65, 1994.

[6] Martin Berger, Kohei Honda and Nobuko Yoshida. *Genericity and the π-calculus.* In 6th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS), volume 2620, pages 103–119. Springer Verlag, apr 2003.

[7] Pierre-Louis Curien and Hugo Herbelin. *The duality of computation.* In 5th ACM International Conference on Functional Programming (ICFP), pages 233–243. ACM Press, 2000.

[8] Vincent Danos, Jean-Baptiste Joinet and Harold Schellinx. *Lkq and lkt: Sequent calculi for second order logic based upon linear decomposition of classical implication.* In Jean-Yves Girard, Yves Lafont and Laurent Regnier, editors, Advances in Linear Logic, pages 211–224. Cambridge University Press, 1995.

[9] Vincent Danos, Jean-Baptiste Joinet and Harold Schellinx. *A new deconstructive logic: linear logic.* Journal of Symbolic Logic, 62:755–807, 1996.

[10] Vincent Danos and Laurent Regnier. *How abstract machines implement head linear reduction.* Submitted, 2003.

[11] Philippa Gardner and Lucian Wischik. *Explicit fusions.* In Mogens Nielsen and Branislav Rovan, editors, 25th International Symposium on Mathematical Foundations of Computer Science (MFCS), volume 1893, pages 373–382. Springer Verlag, 2000.

[12] Kohei Honda, Nobuko Yoshida and Martin Berger. *Control in the π-calculus.* In 4th ACM-SIGPLAN Continuation Workshop, 2004.

[13] J. Martin E. Hyland and Chih-Hao Luke Ong. *Pi-calculus, dialogue games and pcf.* In 7th ACM Conference on Functional Programming Languages and Computer Architecture, pages 96–107, 1995.

[14] Robin Milner. *Functions as processes.* In 17th International Colloquium on Automata, Languages and Programming (ICALP), pages 167–180. Springer Verlag, 1990.

[15] Michel Parigot. *$\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction.* In 3rd International Conference on Logic Programming and Automated Reasoning (LPAR), volume 624, chapter Lecture Notes in Computer Science, pages 190–201. Springer Verlag, 1992.

[16] Laurent Regnier. *Une équivalence sur les λ-termes.* Theoretical Computer Science, 126(2):281–292, 1994.

Axioms and context rules for unification:

$$\frac{}{x{=}y \vDash x = y} \qquad \frac{p \vDash x = y}{p \mid q \vDash x = y} \qquad \frac{p \vDash x = y}{q \mid p \vDash x = y} \qquad \frac{p \vDash x = y \quad z \notin \{x, y\}}{(\boldsymbol{\nu} z)p \vDash x = y}$$

Reflexivity, symmetry and transitivity of equators:

$$\frac{}{p \vDash x = y} \qquad \frac{p \vDash x = y}{p \vDash y = x} \qquad \frac{p \vDash x = y \quad p \vDash y = z}{p \vDash x = z}$$

Renaming of transition labels:

$$\frac{p \vDash u = v \quad u, v \notin \{x_1 \ldots x_n\}}{p \vDash u^\varepsilon(x_1 \ldots x_n) = v^\varepsilon(x_1 \ldots x_n)} \qquad \frac{p \vDash u = u' \quad p \vDash v = v'}{p \vDash [u{=}v] = [u'{=}v']}$$

Table 6: Rules for name unification.

A Technical details

A.1 Bisimulation in $\boldsymbol{\pi}^=$

A polarity ε is an element of $\{\downarrow, \uparrow\}$. $\downarrow$ is called positive and $\uparrow$ is called negative. The notation $u^\varepsilon(\vec{x})$ stands for $u(\vec{x})$ if $\varepsilon = \downarrow$ and for $\bar{u}(\vec{x})$ if $\varepsilon = \uparrow$.

Two names x and y are unified by a process p if $p \vDash x = y$ is derivable using the rules of table 6. Note that an action like $u(x).y{=}z$ does not unify y and z, i.e. the equator $y{=}z$ is inactive as long as the action $u(x)$ has not been consumed. A transition can have one of three kinds of labels:

$e ::= u^\varepsilon(x_1 \ldots x_n)$	visible action (with the x_i fresh and distinct)
$[u{=}v]$	conditional internal reduction
τ	internal reduction

The notation $p \vDash a = b$ is extended to transition labels as detailed in table 6. For a label e, $\mathrm{n}(e)$ is the set of names that occur in e, i.e. $\mathrm{n}(u(x_1 \ldots x_n)) = \{u, x_1 \ldots x_n\}$, $\mathrm{n}([u{=}v]) = \{u, v\}$ and $\mathrm{n}(\tau) = \emptyset$. The labelled transition system of the calculus is defined in table 7.

A simulation is a relation $\mathcal{S}$ over processes such that $p\mathcal{S}q$ implies that

- for any $x, y \in \boldsymbol{N}$, $p \vDash x = y$ implies $q \vDash x = y$,
- for each transition $p \xrightarrow{e} p'$ there is a transition $q \xrightarrow{e} q'$ such that $p'\mathcal{S}q'$.

A bisimulation is a relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are simulations. Two processes p and q are bisimilar if there is a bisimulation $\mathcal{S}$ such that $p\mathcal{S}q$.

Actions (with $\alpha = u^\varepsilon(x_1 \dots x_n)$) and composition:

$$\frac{}{\alpha.p \xrightarrow{\alpha} p} \qquad \frac{}{!\alpha.p \xrightarrow{\alpha} p \mid !\alpha.p} \qquad \frac{p \xrightarrow{\bar{u}(x_1 \dots x_n)} p' \qquad q \xrightarrow{v(x_1 \dots x_n)} q'}{p \mid q \xrightarrow{[u=v]} (\boldsymbol{\nu} x_1 \dots x_n)(p' \mid q')}$$

Renaming:

$$\frac{p \xrightarrow{e} p' \qquad p \vDash e = e'}{p \xrightarrow{e'} p'} \qquad \frac{p \xrightarrow{[u=v]} p' \qquad p \vDash u = v}{p \xrightarrow{\tau} p'}$$

Context:

$$\frac{p \xrightarrow{e} p'}{p \mid q \xrightarrow{e} p' \mid q} \qquad \frac{p \xrightarrow{e} p'}{q \mid p \xrightarrow{e} q \mid p'} \qquad \frac{p \xrightarrow{e} p' \qquad x \notin \mathrm{n}(e)}{(\boldsymbol{\nu} x)p \xrightarrow{e} (\boldsymbol{\nu} x)p'}$$

Table 7: Labelled transition system.

A.2 Simulation in call-by-name

For the "push" rule, we have:

$$\begin{aligned}
[\![(M)N * K]\!] &= (\boldsymbol{\nu} u)\Big((\boldsymbol{\nu} z)\big((\boldsymbol{\nu} v)([\![M]\!]v \mid \bar{\delta}v\langle zu\rangle) \bigm| [\![z = N]\!]\big) \Bigm| [\![K]\!]u\Big) \\
&\equiv (\boldsymbol{\nu} uvz)\big([\![M]\!]v \bigm| \bar{\delta}v\langle zu\rangle \bigm| [\![z = N]\!] \bigm| [\![K]\!]u\big) \\
&\equiv (\boldsymbol{\nu} v)\Big([\![M]\!]v \Bigm| (\boldsymbol{\nu} u)\big((\boldsymbol{\nu} z)(\bar{\delta}v\langle zu\rangle \mid [\![z = N]\!]) \bigm| [\![K]\!]u\big)\Big) \\
&\equiv (\boldsymbol{\nu} v)\big([\![M]\!]v \bigm| (\boldsymbol{\nu} uz)([\![v = Nu]\!] \mid [\![K]\!]u)\big) \\
&= [\![M * NK]\!]
\end{aligned}$$

For the substitution rule for continuations, consider a process $(\boldsymbol{\nu}\alpha)([\![M]\!]u \mid [\![K]\!]\alpha)$, with $K = M_1 \dots M_k\beta$. When δ starts with ?, each $[\![K]\!]\alpha$ is a guarded replication on channel α. By construction there is no other input on α so each output on α can only interact with $[\![K]\!]\alpha$. Hence, up to bisimilarity, we can distribute $[\![K]\!]\alpha$ in $[\![M]\!]u$ by substituting each action $\bar{\alpha}(\vec{x}).p$ by $(\boldsymbol{\nu}\alpha')(\bar{\alpha}'(\vec{x}).p \mid [\![K]\!]\alpha')$ for a fresh α'. All output occurrences of α occur in processes of the form $[\![\mu\theta[\alpha]N]\!]\theta = [\![N]\!]\alpha$, but $(\boldsymbol{\nu}\alpha')([\![N]\!]\alpha' \mid [\![K]\!]\alpha') = [\![N * K]\!]$ and by the previous rule we have $[\![N * K]\!] \equiv [\![(N)M_1 \dots M_k * \beta]\!] = [\![\mu\theta[\beta](N)M_1 \dots M_k]\!]\theta$. By this rule we can deduce the validity of the rule for $\mu\alpha[\beta]$:

$$[\![\mu\alpha[\beta]M * K]\!] = (\boldsymbol{\nu}\alpha)\big([\![M]\!]\beta \bigm| [\![K]\!]\alpha\big) \cong [\![M[K/\alpha]]\!]\beta \equiv [\![M[K/\alpha] * \beta]\!]$$

In the intuitionistic case the rule is not applicable, but it would hold too under the condition that each μ-variable is used linearly. For the $\sharp M$ rule, we have:

$$\begin{aligned}
[\![\sharp M * K]\!] &= (\boldsymbol{\nu} ux)\big(\bar{\delta}'x\langle u\rangle \bigm| \delta'x(v).[\![M]\!]v \bigm| [\![K]\!]u\big) \\
&\to (\boldsymbol{\nu} ux)\big([\![M]\!]u \bigm| [\![K]\!]u \bigm| \delta'x(v).[\![M]\!]v\big) \\
&\cong (\boldsymbol{\nu} u)\big([\![M]\!]u \bigm| [\![K]\!]u\big) = [\![M * K]\!]
\end{aligned}$$

where $\to$ contains one transition for each modality in the word δ'. Since $[\![K]\!]u$ and E are blocked on actions that cannot be on channel x, this reduction is clearly the only one possible. The term $\delta'x(v).[\![M]\!]v$ is not consumed since δ' must contain !, however there is no other occurrence of x so we can discard it by bisimilarity.

For the substitution rule for terms, the argument is the same as for continuations. In this case, the only outputs on the channel of a λ-variable x are of the form $[\![x]\!]u = \bar{\delta}'x\langle u\rangle$, hence after distribution of $[\![x = M]\!]$ we get $(\boldsymbol{\nu}x')([\![x']\!]u \mid [\![x' = M]\!]) = [\![\sharp M]\!]u$ for a fresh x'. For the "pop" rule, we thus have

$$\begin{aligned}
[\![\lambda x.M * NK]\!] &= (\boldsymbol{\nu}u)\Big(\delta u(xv).[\![M]\!]v \;\Big|\; (\boldsymbol{\nu}w)\big((\boldsymbol{\nu}z)(\bar{\delta}u\langle zw\rangle \mid [\![z = N]\!]) \;\big|\; [\![K]\!]w\big)\Big) \\
&\equiv (\boldsymbol{\nu}uwx)\big(\delta u(xv).[\![M]\!]v \;\big|\; \bar{\delta}u\langle xw\rangle \;\big|\; [\![x = N]\!] \;\big|\; [\![K]\!]w\big) \\
&\to (\boldsymbol{\nu}uwx)\big([\![M]\!]w \;\big|\; \bar{\delta}u\langle xw\rangle \;\big|\; [\![x = N]\!] \;\big|\; [\![K]\!]w\big) \\
&\equiv (\boldsymbol{\nu}wx)\big([\![M]\!]w \;\big|\; (\boldsymbol{\nu}u)\bar{\delta}u\langle xw\rangle \;\big|\; [\![x = N]\!] \;\big|\; [\![K]\!]w\big) \\
&\cong (\boldsymbol{\nu}wx)\big([\![M]\!]w \;\big|\; [\![x = N]\!] \;\big|\; [\![K]\!]w\big) \\
&\cong (\boldsymbol{\nu}w)\big([\![M[\sharp N/x]]\!]w \;\big|\; [\![K]\!]w\big) \\
&= [\![M[\sharp N/x] * K]\!]
\end{aligned}$$

where $\to$ contains one transition for each modality in the word δ. In the classical case, $\bar{\delta}u\langle xw\rangle$ is not consumed since $\bar{\delta}$ contains !, however we know that u does not occur elsewhere since all duplications of continuations are performed by the rule for $\mu\alpha[\beta]$, so this action becomes inactive and it is bisimilar to the empty process. As above, this reduction is the only one possible.

A.3 Simulation in call-by-value

The substitution rule for continuations and the equivalence rule for $\mu\alpha[\beta]$ hold by the same arguments as in the case of call-by-name.

For the first equivalence, we have

$$\begin{aligned}
[\![K * (M)N]\!] &= (\boldsymbol{\nu}u)\big([\![u = K]\!] \;\big|\; (\boldsymbol{\nu}v)([\![v = uM^f]\!] \mid [\![N]\!]v)\big) \\
&\equiv (\boldsymbol{\nu}v)\big((\boldsymbol{\nu}u)([\![u = K]\!] \mid [\![v = uM^f]\!]) \;\big|\; [\![N]\!]v\big) \\
&= [\![KM^f * N]\!]
\end{aligned}$$

For the first reduction rule, we have

$$\begin{aligned}
[\![KM^f * V]\!] &= (\boldsymbol{\nu}u)\Big((\boldsymbol{\nu}v)\big(\bar{\gamma}'u(x).(\boldsymbol{\nu}w)([\![M]\!]w \mid \bar{\delta}w\langle xv\rangle) \;\big|\; [\![v = K]\!]\big) \;\Big|\; \gamma'u(x).[\![x = V]\!]\Big) \\
&\equiv (\boldsymbol{\nu}uv)\big(\bar{\gamma}'u(x).(\boldsymbol{\nu}w)([\![M]\!]w \mid \bar{\delta}w\langle xv\rangle) \;\big|\; [\![v = K]\!] \;\big|\; \gamma'u(x).[\![x = V]\!]\big) \\
&\to (\boldsymbol{\nu}uvx)\big(\bar{\gamma}'u(x).(\boldsymbol{\nu}w)([\![M]\!]w \mid \bar{\delta}w\langle xv\rangle) \\
&\qquad \;\big|\; (\boldsymbol{\nu}w)([\![M]\!]w \mid \bar{\delta}w\langle xv\rangle) \;\big|\; [\![v = K]\!] \;\big|\; [\![x = V]\!]\big) \\
&\equiv (\boldsymbol{\nu}w)\big((\boldsymbol{\nu}u)\bar{\gamma}'u(x).(\boldsymbol{\nu}w)([\![M]\!]w \mid \bar{\delta}w\langle xv\rangle) \\
&\qquad \;\big|\; [\![M]\!]w \;\big|\; (\boldsymbol{\nu}vx)([\![x = V]\!] \mid \bar{\delta}w\langle xv\rangle \mid [\![v = K]\!])\big) \\
&\cong [\![KV^a * M]\!]
\end{aligned}$$

where $\to$ contains one transition for each modality in the word γ'. In the classical case $\bar{\gamma}'$ contains ! so the continuation at u is not consumed, however we know that u has no other occurrence since continuations are duplicated by the rule for μ, so we can erase the residual term on u by bisimilarity. This is the only possible reduction as soon as γ' is not empty. The second reduction rule is deduced as

$$
\begin{aligned}
[\![KW^a * V]\!] &= (\boldsymbol{\nu}u)\big((\boldsymbol{\nu}vx)([\![x = W]\!] \mid \bar{\delta}u\langle xv\rangle \mid [\![v = K]\!]) \bigm| \gamma' u(z).[\![z = V]\!]\big) \\
&\equiv (\boldsymbol{\nu}uvx)\big([\![x = W]\!] \bigm| \bar{\delta}u\langle xv\rangle \bigm| [\![v = K]\!] \bigm| \gamma' u(z).[\![z = V]\!]\big) \\
&\to (\boldsymbol{\nu}uvx)\big([\![x = W]\!] \bigm| \bar{\gamma}z\langle xv\rangle \bigm| [\![v = K]\!] \bigm| [\![z = V]\!]\big) \\
&\equiv [\![K * V \cdot W]\!]
\end{aligned}
$$

where $\to$ contains one transition for each modality in the word γ', since $\delta = \gamma'\gamma$. As above, this is the only reduction. For the substitution rule, we have

$$
\begin{aligned}
[\![K * \lambda x.M \cdot V]\!] &= (\boldsymbol{\nu}u)\big([\![u = K]\!] \bigm| (\boldsymbol{\nu}vz)(\gamma v(xw).[\![M]\!]w \mid [\![z = V]\!] \mid \bar{\gamma}v\langle zu\rangle)\big) \\
&\to (\boldsymbol{\nu}u)\big([\![u = K]\!] \bigm| (\boldsymbol{\nu}x)([\![M]\!]u \mid [\![x = V]\!])\big) \\
&\cong (\boldsymbol{\nu}u)\big([\![u = K]\!] \bigm| [\![M[V/x]]\!]u\big) \\
&= [\![K * M[V/x]]\!]
\end{aligned}
$$

where there is one transition for each modality in γ. The step after the reduction is an instance of the substitution lemma $(\boldsymbol{\nu}x)([\![M]\!]u \mid [\![x = V]\!]) \cong [\![M[V/x]]\!]$. This lemma holds by the same argument as in the case of call-by-name: the binding $[\![x = V]\!]$ can be distributed to all occurrences of x, but any occurrence of x occurs in a binding $[\![y = x]\!]$ so we have

$$
\begin{aligned}
&(\boldsymbol{\nu}x)([\![y = x]\!] \mid [\![x = M]\!]) = (\boldsymbol{\nu}x)(y{=}x \mid [\![x = M]\!]) \\
&\qquad\qquad \equiv (\boldsymbol{\nu}x)(y{=}x) \mid [\![y = M]\!] \cong [\![y = M]\!]
\end{aligned}
$$

using the obvious bisimilarity $(\boldsymbol{\nu}x)(x{=}y) \cong 1$.