A Concurrent Model for Linear Logic

Emmanuel Beffara

PPS, Université Paris 7 & CNRS

Abstract

We build a realizability model for linear logic using a name-passing process calculus. The construction is based on testing semantics for processes, drawing ideas from spatial and modal logics, and yields a new type system for process calculi that ensures termination while allowing significantly concurrent behaviours. Then we study how embeddings of intuitionistic and classical logics into linear logic induce typed translations of λ and $\lambda\mu$ calculi in which new concurrent instructions can be introduced, thus sketching the basis for a Curry-Howard interpretation of linear and classical proofs in terms of concurrent interaction.

1 Introduction

This paper addresses the problem of finding a proper and meaningful connection between linear logic and concurrency in the tradition of the Curry-Howard isomorphism. To this end, we develop a logic of behavioural properties of processes, using ideas from spatial and modal logics, yielding an interpretation of the logic as a type system for processes.

Since the inception of linear logic [15], there have been several attempts at providing it with a canonical computational counterpart, the same way as λ-calculus is a computational representation of core intuitionistic logic. Proof-nets, which are canonical proof objects for linear logic, have several meaningful computational intuitions, notably about parallelism and resource management, but they are too closely related to logic and proof theory to be used as a basic computational object by themselves. Nevertheless, those intuitions promoted the key ideas of computation by interaction, leading notably to the general approach of game semantics, which shares many intuitions with previously known algebraic formalizations of parallel and concurrent processes. Models of linear logic actually based on concurrent processes were also studied, notably by Abramsky in the context of CCS [1,4], using the very powerful technique of realizability. A notable contribution in this direction is Bellin and Scott's encoding of proof nets into π-calculus [5], which adresses the question of the dynamics of proofs from a concurrent point of view. In a much more sequential flavour, Girard's ludics [17] is a radical approach based

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

on similar principles, and the locative issues it unveiled are of importance in the present work, in the form of the handling of channel names in processes.

Among models of interactive computation, the approach of π-calculus is well suited to a logical study of concurrency in the Curry-Howard tradition. Indeed, name-passing communication and mobility have the same kind of purity and geometrical flavour as λ-calculi, and the geometric intuitions behind process calculi make them, modulo notational variations, rather pure and canonical models for this kind of computation. Although the typing of processes in the Curry-Howard school of thought is not a widespread approach, the search for good logics for describing concurrent processes is not a new topic. Two important kinds of logics have proved to be useful in this respect: dating back to CCS are modal logics in the style of Hennessy-Milner, which are meant to describe the temporal behaviour of a system by studying its possible transitions. In a different perspective, spatial logics have been defined to describe geometric properties of objects, and they have been used more recently for the study of the geometric aspects of mobile processes [8]. Those logics provide standard ways of describing spatial and temporal properties of concurrent behaviours, and as such they should be considered a natural starting point for bridging the gap with linear logic and its interactive principles.

With these ideas in mind, this paper proceeds as follows: in section 2 we define an appropriate formulation of a π-calculus-like process algebra, and in section 3 we define the kind of observational equivalence that we use as the foundation for our logical construction. In section 4 we define combinators over sets of processes from semantic properties of interaction, from which we deduce in section 5 a type system for mobile processes that notably ensures termination. The type system we obtain is precisely a form of second-order linear logic, and we use it in section 6 as a tool to investigate the computational properties of classical logical systems, exploiting previous studies on the embeddings of classical logic into linear logic.

2 The $\vec{\pi}$-calculus

We work with a name-passing process calculus in the family of polyadic π-calculi, with some important features:

- All input and output actions are binders, while all non-private communications are explicitly handled by forwarders. This gives finer control over name unification and ensures good properties of communications.
- We require that all communications occur on bound channels. This gives a clear distinction between private names, with possible redexes, and public names with actions of the same polarity.

The first property is provided by the syntactical structure of the calculus, the second one is obtained by a sorting discipline in the style of i/o types.

Definition 2.1 We assume a countable set $\mathcal{N}$ of channel names, ranged over by lowercase letters. The actions (written α) and terms (written $P, Q \ldots$) of the $\vec{\pi}$-calculus are generated by the following grammar:

$$\begin{array}{rll} \alpha := & u^\varepsilon(x_1 \ldots x_n) & \text{elementary actions, with } \varepsilon \in \{\downarrow, \uparrow\} \\ P, Q := & \alpha.P \mid !\alpha.P & \text{linear and replicated actions} \\ & u{\to}v & \text{forwarders} \\ & 1 \mid (P \mid Q) \mid (\boldsymbol{\nu} x)P & \text{composition and restriction} \end{array}$$

An action $u^\varepsilon(\vec{x})$ is a binder for the object names $\vec{x}$, a restriction $(\boldsymbol{\nu} x)$ is a binder for x, and terms are considered up to renaming of bound names. By convention, $u^\downarrow(\vec{x}) = u(\vec{x})$ is called positive (or input) and $u^\uparrow(\vec{x}) = \bar{u}(\vec{x})$ is called negative (or output).

For precision, the primary operational semantics of the $\vec{\pi}$-calculus is defined as a labelled transition system (see table 1). This kind of formulation is somewhat heavy but the intuition behind the semantics is natural: a communication can occur between a sender $\bar{u}(x).P$ and a receiver $v(y).Q$ when the surrounding process contains forwarders from channel u to channel v, hence the prototypical reduction rule:

$$\bar{u}(x).P \mid u{\to}v \mid v(x).Q \quad \to \quad (\boldsymbol{\nu} x)(P \mid Q) \mid u{\to}v$$

The fact that both agents use the same name x can be guaranteed by renaming since both actions are binders. Connections are assumed to be permanent (that is why $u{\to}v$ is present in the right-hand side in the rule above) and reliable, in the sense that a forwarder cannot trigger a communication unless there are two compatible actions. In this respect, a forwarder $u{\to}v$ is different from a π-calculus process like $!u(x).\bar{v}\langle x\rangle$, at least in a synchronous setting.

Definition 2.2 The *forwarding* relation of a process P is the relation $\mathcal{F}(P)$ over names defined as follows (where $(\cdot)^*$ is the reflexive transitive closure and $R \setminus X$ is defined as $(R \cap (\mathcal{N} \setminus X)) \cup \mathrm{id}_X$ for any $X \subseteq \mathcal{N}$):

$$\begin{array}{rlrl} \mathcal{F}(u{\to}v) &= \mathrm{id} \cup \{(u, v)\} & \mathcal{F}(P \mid Q) &= (\mathcal{F}(P) \cup \mathcal{F}(Q))^* \\ \mathcal{F}((\boldsymbol{\nu} x)P) &= \mathcal{F}(P) \setminus \{x\} & \mathcal{F}(\alpha.P) &= \mathcal{F}(!\alpha.P) = \mathcal{F}(1) = \mathrm{id} \end{array}$$

Transitions are of three kinds: actions $u^\varepsilon(\vec{x})$, conditions $[u{\to}v]$ and τ-actions. The free names $\mathrm{fn}(a)$ of a label a are the subjects (i.e. $\{u\}$ and $\{u, v\}$ respectively), the bound names $\mathrm{bn}(a)$ are the objects ($\vec{x}$ and $\emptyset$ respectively), and we define $\mathrm{n}(a) = \mathrm{fn}(a) \cup \mathrm{bn}(a)$. Transitions are derived by the rules in table 1.

The implementation of forwarding in the transition rules is very similar to the approach of explicit fusions [14]: the interaction rule derives transitions labelled $[u{\to}v]$, which represent τ-transitions modulo the existence of forwarders from u to v, and the forwarding rules can turn a transition $[u{\to}v]$ into a proper τ-transition when this forwarding actually exists in the term. Remark that if

Emission of actions and interaction (with the symmetric interaction rule):

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{}{!\alpha.P \xrightarrow{\alpha} !\alpha.P \mid P} \qquad \frac{P \xrightarrow{\bar{u}(\vec{x})} P' \qquad Q \xrightarrow{v(\vec{x})} Q'}{P \mid Q \xrightarrow{[u \to v]} (\boldsymbol{\nu}\vec{x})(P' \mid Q')}$$

Forwarding rules:

$$\frac{P \xrightarrow{[u \to v]} P'}{P \xrightarrow{[u' \to v']} P'} (u, u'), (v', v) \in \mathcal{F}(P) \qquad \frac{P \xrightarrow{[u \to v]} P'}{P \xrightarrow{\tau} P'} (u, v) \in \mathcal{F}(P)$$

$$\frac{P \xrightarrow{\bar{u}(\vec{x})} P'}{P \xrightarrow{\bar{v}(\vec{x})} P'} (u, v) \in \mathcal{F}(P) \qquad \frac{P \xrightarrow{v(\vec{x})} P'}{P \xrightarrow{u(\vec{x})} P'} (u, v) \in \mathcal{F}(P)$$

Contextual rules (where a is any label):

$$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \qquad \frac{P \xrightarrow{a} P'}{Q \mid P \xrightarrow{a} Q \mid P'} \qquad \frac{P \xrightarrow{a} P'}{(\boldsymbol{\nu}x)P \xrightarrow{a} (\boldsymbol{\nu}x)P'} x \notin \mathrm{n}(a)$$

Table 1
The labelled transition system.

the actions involved in a communication are on the same channel, then a silent interaction between them is always possible, since $\mathcal{F}(P)$ is always reflexive. Note that all four forwarding rules are required since the transition system is defined strictly on the syntax of terms, with no scope extrusion.

Definition 2.3 A bisimulation is a symmetric relation $\mathcal{S}$ over processes such that $P \mathcal{S} Q$ implies $\mathcal{F}(P) = \mathcal{F}(Q)$ and for any transition $P \xrightarrow{a} P'$ there is a transition $Q \xrightarrow{a} Q'$ with $P' \mathcal{S} Q'$. Bisimilarity is the union of all bisimulations.

The interaction rule requires that two opposite actions have the same arity in order to interact. In order to avoid problems with arities, we will require the channels of each process to have a specified type (not to be confused with process types as defined later). This type system can be seen as a fragment of the usual i/o types of the π-calculus [23], with a strong restriction: we forbid the "input *and* output" capability, instead all situations where opposite actions can interact are created on private channels using the restriction rule.

Definition 2.4 Channel types and interfaces are defined as follows:

channel types	$s := \varepsilon(s_1 \dots s_n) \mid *$	with $\varepsilon \in \{\downarrow, \uparrow\}$
interfaces	$I := x_1 : s_1, \dots, x_n : s_n$	

Let $\overline{(\cdot)}$ be the involution defined inductively as $\overline{\downarrow(s_1 \dots s_n)} = \uparrow(\bar{s}_1 \dots \bar{s}_n)$, naturally extended to interfaces. A typing judgement has the form $P :: I$, a term P respects an interface I if $P :: I$ is derivable using the rules in table 2.

$$\frac{\dfrac{}{u{\to}v :: I, u : {\downarrow}(s_1, \dots, s_n), v : {\uparrow}(\bar{s}_1, \dots, \bar{s}_n)} \qquad P :: I, u : \varepsilon(s_1 \dots s_n), x_1 : s_1, \dots, x_n : s_n}{u^\varepsilon(\vec{x}).P :: I, u : \varepsilon(s_1 \dots s_n)}$$

$$\frac{}{1 :: I} \qquad \frac{P :: I \qquad Q :: I}{P \mid Q :: I} \qquad \frac{P :: I}{!P :: I} \qquad \frac{P :: I, x_1 : s, x_2 : \bar{s}}{(\boldsymbol{\nu} x)P[x/x_1, x/x_2] :: I}$$

Table 2
Typing rules for channels.

The formulation of the restriction rule assures that a name with capabilities of both polarities always occurs under a restriction: in the typing derivation, a distinct name is used for each polarity, and these names are unified only when inserting the binder. The type $*$ is used when typing is unnecessary (and to make channel types finite). The language of channel types could be extended with variables and fixed points, but this is not needed for the present work.

Proposition 2.5 *If $P :: I$ is derivable, then all reducts of P by τ or $[u{\to}v]$ transitions respect I, and for each transition $P \xrightarrow{\alpha} P'$ with $\alpha = u^\varepsilon(x_1 \dots x_n)$ there is $u : \varepsilon(s_1 \dots s_n)$ in I and $P' :: I, x_1 : s_1, \dots, x_n : s_n$ is derivable.*

Proposition 2.6 *The congruence $\equiv$ generated by the abelian monoid laws for $(|, 1)$, the standard scoping laws and the following rules is a bisimulation:*

$$(\boldsymbol{\nu} x)(P \mid x{\to}y) \equiv P[y/x] \qquad \text{if } P \text{ has no input on } x$$
$$(\boldsymbol{\nu} x)(y{\to}x \mid P) \equiv P[y/x] \qquad \text{if } P \text{ has no output on } x$$

If $P \equiv Q$ then $P :: I$ is derivable if and only if $Q :: I$ is derivable.

The substitution rule in this proposition gives a translation of process calculi with communication of free names, including π-calculus and fusion calculi [22,14], for processes that respect our strict channel typing constraints. Indeed, an action $u^\varepsilon\langle\vec{x}\rangle.P$ that communicates the free names $\vec{x}$ can be encoded as a process $u^\varepsilon(\vec{y}).(P \mid \prod_i x_i{\to}y_i \mid \prod_j y_j{\to}x_j)$, where x_i ranges over names in $\vec{x}$ that should have input capabilities and x_j ranges over those that should have output capabilities. In the sequel, we sometimes use this notation to make expressions lighter, for instance $u(x).\bar{v}\langle x\rangle$ stands for $u(x).\bar{v}(y).(x{\to}y)$ if the environment is expected to send actions $\bar{u}(x)$ with output capabilities on x.

In the sequel, we use the notation $\mathcal{A} : I$ to denote a set $\mathcal{A}$ of terms that all respect an interface I. As an alternative notation, if I and J are two disjoint interfaces, we write $I \to J$ for the interface $\bar{I} \cup J$ (so in particular we have $\overline{I \to J} = J \to I$). This simple notational device will make some definitions cleaner, as it splits the interfaces into arbitrary inputs and outputs (unrelated to capabilities), in a way similar to Abramsky's approach in interaction categories [2]. The primary application is indeed the associativity of composition:

Definition 2.7 Let I, J, K be three pairwise disjoint interfaces. For any processes $P :: I \to J$ and $Q :: J \to K$, the composition $P \cdot Q :: I \to K$ is defined as $P \cdot Q = (\boldsymbol{\nu}\vec{x})(P \mid Q)$ where $\vec{x}$ is the domain of J.

Proposition 2.8 *For any interfaces I, J, K, L pairwise disjoint and any processes $P :: I \to J$, $Q :: J \to K$, $R :: K \to L$ we have $(P \cdot Q) \cdot R \equiv P \cdot (Q \cdot R)$.*

Proof. Write $\cdot_J$ the composition with restriction on interface J, then the composition is $(P \cdot_J Q) \cdot_K R$. The restrictions introduced by $P \cdot Q$ affect the names of $\mathrm{dom}(J)$ and those introduced by $Q \cdot R$ affect the names of $\mathrm{dom}(K)$, so these restrictions commute because they are on disjoint sets of names. □

3 Observational equivalence

The kind of behavioural equivalence we are interested in is the traditional notion of testing, i.e. we only observe the outcome of interactions instead of the actual transitions. The basic ingredient is to choose a set of processes that are considered to act well (for a parametrized notion of well), and to consider that two processes interact well whenever their composition is element of this set. In the sequel we will focus on termination, by defining well-behavedness as the absence of infinite reduction and observing only the possibility of divergence. Of course this is one choice among many, but we consider it to be meaningful both from the logical and concurrent points of view.

3.1 Orthogonality

Definition 3.1 The *observation* is the set $\bot$ of all terms that have no infinite reduction.

Note that by "reduction" we refer to τ-transitions, so termination guarantees that a system cannot go into an infinite loop without interacting with its environment. This property is a reasonable expectation, while the existence of infinite sequences of observable transitions is not a problem, and it is even required for server-like processes, which are supposed to answer any arbitrary number of requests. So we consider that two processes interact correctly if their composition cannot create a divergence:

Definition 3.2 Orthogonality is defined between processes of opposite interfaces. Two process $P :: I$ and $Q :: \bar{I}$ are orthogonal, written $P \perp Q$, if $P \mid Q \in \bot$. Two sets of terms $\mathcal{A} : I$ and $\mathcal{B} : \bar{I}$ are orthogonal when $P \perp Q$ for every $P \in \mathcal{A}$ and $Q \in \mathcal{B}$. The orthogonal of a set $\mathcal{A} : I$ is the set $\mathcal{A}^\perp = \{Q \mid \forall P \in \mathcal{A}, P \mid Q \in \bot\}$.

The constraint that interfaces of orthogonal processes should be opposed is not as strong a restriction as it may look. Indeed, if a process P has more free

names than Q, then $P|Q$ terminates if and only if $(\boldsymbol{\nu}\vec{x})P|Q$ terminates, where $\vec{x}$ is the set of public names known by P but not by Q, and then $(\boldsymbol{\nu}\vec{x})P$ and Q do have opposed interfaces. In other words, interfaces can be considered as annotations that indicate the names involved in interactions, while the other names are not shared by interacting processes.

Other similar observations could be used, like the related ideas of may- and must-testing [7,9], using other definitions of $\bot$. Then the interpretation of modal connectives in section 4.2 would have to be adapted, but the general technique, and most importantly the type system, would be exactly the same.

Proposition 3.3 *Let $\mathcal{A}$, $(\mathcal{A}_i)$ and $\mathcal{B}$ be sets of terms that respect a common interface I. We have the following properties:*

$$\mathcal{A}^{\bot\bot\bot} = \mathcal{A}^{\bot} \qquad (\textstyle\bigcup_i \mathcal{A}_i)^{\bot} = \bigcap_i \mathcal{A}_i^{\bot}$$
$$\text{if } \mathcal{A} \subseteq \mathcal{B} \text{ then } \mathcal{B}^{\bot} \subseteq \mathcal{A}^{\bot} \qquad (\textstyle\bigcup_i \mathcal{A}_i)^{\bot\bot} = (\bigcup_i \mathcal{A}_i^{\bot\bot})^{\bot\bot}$$

Proof. Those properties are immediate, they hold for any pair of sets (here the sets of processes of interface I and $\bar{I}$) linked by a binary relation $\bot$. □

Therefore the bi-orthogonal is a closure operator, and the orthogonal of any set is closed. A closed set $\mathcal{A} : I$ will be called a *behaviour*, since the bi-orthogonal is a closure by behavioural equivalence. The lemma above states that the orthogonal is a decreasing operator, involutive on behaviours, and that the set of behaviours, ordered by inclusion, has arbitrary lower bounds (the intersections) and upper bounds (the closures of unions).

3.2 Pre-orders and subtyping

The standard notion of testing pre-order can be rephrased here, and this relation yields a natural order (actually the inverse of inclusion) over behaviours. We leave the interfaces implicit when they are clear from the context.

Definition 3.4 The behavioural pre-order over processes is defined as $P \sqsubseteq Q$ if $\{P\}^{\bot} \subseteq \{Q\}^{\bot}$. The associated equivalence is written $P \simeq Q$.

Proposition 3.5 *Behaviours are closed under reduction. They are closed under bisimulation among terms that respect their interface.*

Proof. For any processes P and Q, assuming a reduction $P \to P'$, if $P \mid Q$ cannot diverge then $P' \mid Q$ cannot diverge either since it is a reduct of the latter, so $Q \bot P$ implies $Q \bot P'$, hence $\{P\}^{\bot} \subseteq \{P'\}^{\bot}$ and $P' \in \{P\}^{\bot\bot}$. If two processes are bisimilar, then they have exactly the same transitions, so they diverge in the same contexts. □

This fact derives directly from the same properties of the observation, and it allows us in the sequel to consider terms up to bisimulation (only checking the channel typing constraint). The closure by reduction can be rephrased by saying that $P \to^* Q$ implies $\{P\}^{\bot} \subseteq \{Q\}^{\bot}$. One can interpret reduction

steps that make the orthogonal strictly increase as those that make a choice in some situation of non-determinism, while the reduction steps that leave the orthogonal unchanged are concerned with the deterministic part of reduction.

Definition 3.6 Let I be an interface, define

$$\mathbf{0}_I = (\emptyset : I)^{\perp\perp} \qquad \top_I = \text{all processes of interface } I$$
$$\mathbf{1}_I = (\{1\} : I)^{\perp\perp} \qquad \bot_I = \{P \mid P \in \bot, P :: I\}$$

This defines a subtyping structure among behaviours of a given interface. From proposition 3.3 we easily deduce the commutative monoid structure of behaviours for $(\vee, \mathbf{0})$ and $(\wedge, \top)$ and the duality between those structures.

Since the observation is termination, no process can be orthogonal to a process that may diverge, so any behaviour that contains a diverging process is equal to $\top_I$, and $\mathbf{0}_I$ is always empty (since there are diverging processes). Subsequently $\bot_I$ is the second largest behaviour after $\top_I$ and its orthogonal $\mathbf{1}_I$ is the smallest non-empty behaviour. At interface $\emptyset$ we have $\mathbf{1} = \bot$, so there are three behaviours with empty interface, ordered as $\mathbf{0}_\emptyset \subset \mathbf{1}_\emptyset = \bot_\emptyset \subset \top_\emptyset$.

Definition 3.7 A behaviour is *non-degenerate* if it is distinct from $\mathbf{0}$ and $\top$.

In other words, a non-degenerate behaviour $\mathcal{A}$ is not empty and contains only terminating processes, or equivalently $\mathbf{1} \subseteq \mathcal{A} \subseteq \bot$, which shows that the set of non-degenerate behaviours is closed by arbitrary upper and lower bounds.

4 Behavioural connectives

In this section, we define operators on behaviours from the semantic properties of processes, from which we deduce the type system in the next section.

4.1 Interaction and composition

Definition 4.1 Let I, J and K be three pairwise disjoint interfaces, let $\vec{x}$ be an arbitrary enumeration of $\mathrm{dom}(J)$. For any behaviours $\mathcal{A} : I \to J$ and $\mathcal{B} : J \to K$, define the composition as

$$\mathcal{A} \cdot \mathcal{B} : I \to K = \big\{(\boldsymbol{\nu}\vec{x})(P \mid Q) \mid P \in \mathcal{A}, Q \in \mathcal{B}\big\}^{\perp\perp}$$

Proposition 4.2 *Behaviour composition is associative, commutative, monotonic and distributive over upper bounds.*

Proof. The key is to prove $\mathcal{A} \cdot \mathcal{B} = \mathcal{A}^{\perp\perp} \cdot \mathcal{B}^{\perp\perp}$ for all $\mathcal{A}, \mathcal{B}$, from which everything follows. For this, remark that for any P, Q, R it holds that $P \cdot Q \perp R$ if and only if $P \perp Q \cdot R$. Then $P \perp \mathcal{A} \cdot \mathcal{B}$ implies $P \perp Q \cdot R$ for any $Q \in \mathcal{A}$ and $R \in \mathcal{B}$, hence $P \cdot Q \perp R$ and $P \cdot Q \perp \mathcal{B}^{\perp\perp}$, so $P \perp \mathcal{A} \cdot \mathcal{B}^{\perp\perp}$, and $P \perp \mathcal{A}^{\perp\perp} \cdot \mathcal{B}^{\perp\perp}$ is proved similarly, so $\mathcal{A}^{\perp\perp} \cdot \mathcal{B}^{\perp\perp} \subseteq \mathcal{A} \cdot \mathcal{B}$, and the reverse is trivial. □

An important special case of composition is when it is applied to behaviours of opposite interfaces $\mathcal{A} : I \to J$ and $\mathcal{B} : J \to I$, in this case the composed behaviour has an empty interface. This gives a characterization of the orthogonality of behaviours: $\mathcal{A} \perp \mathcal{B}$ if and only if $\mathcal{A} \cdot \mathcal{B} \subseteq \bot_\emptyset$, with equality if $\mathcal{A}$ and $\mathcal{B}$ are not empty, so in particular $\mathcal{A} \cdot \mathcal{A}^\bot = \bot_\emptyset$ in the non-degenerate case.

Another important case of this composition is when applied to behaviours of disjoint interfaces, i.e. when $J = \emptyset$. Then the composition of $\mathcal{A} : I$ and $\mathcal{B} : K$ has interface $I \cup K$, no restriction is introduced and the composed process is generated by parallel composition.

Definition 4.3 The composition operator $\cdot$ applied to disjoint interfaces is noted $\otimes$. By duality define $\mathcal{A} \parr \mathcal{B} = (\mathcal{A}^\bot \otimes \mathcal{B}^\bot)^\bot$ and $\mathcal{A} \multimap \mathcal{B} = \mathcal{A}^\bot \parr \mathcal{B}$.

Clearly the operators $\otimes$ and $\parr$ are commutative and associative, and they have neutral elements $\mathbf{1}_\emptyset$ and $\bot_\emptyset$ respectively. From proposition 4.2, $\otimes$ distributes over upper bounds and $\parr$ distributes over intersections. Moreover, the following rules are easy to check, for any $\mathcal{A}$ with interface I:

$$\mathcal{A} \otimes \mathbf{0}_J = \mathbf{0}_{I \cup J} \qquad \mathcal{A} \otimes \top_J = \top_{I \cup J} \quad \text{if } \mathcal{A} \neq \mathbf{0}_I$$
$$\mathcal{A} \parr \top_J = \top_{I \cup J} \qquad \mathcal{A} \parr \mathbf{0}_J = \mathbf{0}_{I \cup J} \quad \text{if } \mathcal{A} \neq \top_I$$

Proposition 4.4 *The operator $\otimes$ is injective on non-degenerate behaviours and distributive over arbitrary intersections*

Proof (sketch). Injectivity is easily checked by constructing appropriate tests. For the distribution, the idea is to consider interaction traces (essentially sequences of transition labels), remarking that behaviours are characterised by their sets of traces, when observing termination. Then show that for any pair of behaviours $\mathcal{A} : I$ and $\mathcal{B} : J$, any trace of $\mathcal{A} \otimes \mathcal{B}$ can be decomposed uniquely as a trace of $\mathcal{A}$ and a trace of $\mathcal{B}$. This decomposition is what ensures the distribution of $\otimes$ over intersections. □

The tensor product $\otimes$ can be seen as a form of separating spatial conjunction, as can be found in spatial logics, and the operator $\parr$ is a symmetric form of spatial implication. In particular, its oriented form $\mathcal{A} \multimap \mathcal{B} = \mathcal{A}^\bot \parr \mathcal{B}$ defines the behaviour of processes that, when composed with processes of $\mathcal{A}$, are in behaviour $\mathcal{B}$. This makes $\parr$ the right operator for expressing contextual properties, indeed we have the following rule:

Proposition 4.5 *For any behaviours $\mathcal{A} : I$, $\mathcal{B} : J \to K$, $\mathcal{C} : K \to L$ and $\mathcal{D} : M$ with I, J, K, L, M pairwise disjoint, $(\mathcal{A} \parr \mathcal{B}) \cdot (\mathcal{C} \parr \mathcal{D}) \subseteq \mathcal{A} \parr (\mathcal{B} \cdot \mathcal{C}) \parr \mathcal{D}$.*

Proof. Let $P \in \mathcal{A} \parr \mathcal{B}$ and $Q \in \mathcal{C} \parr \mathcal{D}$. For any $R \in \mathcal{A}^\bot$ and $T \in \mathcal{B}^\bot$ we have $P \cdot (R \cdot T) \in \bot$, with $P : I \cup (J \to K)$, $R : \bar{I}$ and $T : K \to J$ so the restrictions are independent and composition is associative, so $P \cdot R \perp T$, therefore $P \cdot R \in \mathcal{B}$. The same way, we get that for every $S \in \mathcal{D}^\bot$, $Q \cdot S \in \mathcal{C}$, so $(P \cdot R) \cdot (Q \cdot S) \in \mathcal{B} \cdot \mathcal{C}$. In these compositions, the restrictions are placed on disjoint interfaces I, K, M respectively, so the composition is associative

and we obtain by congruence $(P \cdot Q) \cdot (R \cdot S) \in \mathcal{B} \cdot \mathcal{C}$ for any $R \in \mathcal{A}^\perp$ and $S \in \mathcal{D}^\perp$, from which we deduce $P \cdot Q \in \mathcal{A} \parr \mathcal{D} \parr (\mathcal{B} \cdot \mathcal{C})$. □

Since $\perp_\emptyset$ is the neutral element of $\parr$, the previous proposition gives the rule $(\mathcal{A} \multimap \mathcal{B}) \cdot (\mathcal{B} \multimap \mathcal{C}) \subseteq (\mathcal{A} \multimap \mathcal{C})$ for any behaviours $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ with disjoint interfaces. As another example, we have the following specification of forwarders:

Proposition 4.6 *Let $\mathcal{A}$ be a behaviour of interface $\{u : \uparrow(\vec{s})\}$. For any name $v \neq u$ we have $u{\to}v \in \mathcal{A}^\perp \parr \mathcal{A}[v/u]$.*

Proof. First note that $u{\to}v$ does respect the expected interface. By definition, any term $P \in \mathcal{A}$ only has output capabilities on u, so $P \cdot (u{\to}v) \equiv P[v/u] \in \mathcal{A}[v/u]$, and for any $Q \in \mathcal{A}[v/u]^\perp$ we have $(u{\to}v) \cdot (P \mid Q) \in \perp$. □

4.2 Modalities

In order to describe the dynamic properties of behaviours, we now introduce modalities in the style of Hennessy-Milner logic [19]. An important point about the modalities we use here is their effect on interfaces: given an action α of the form $u^\varepsilon(\vec{x})$, we consider processes whose interface contains only u and whose only transitions lead into a given behaviour, the interface of which is defined on u and $\vec{x}$. In particular the modality $[\alpha]$ acts as a binder for the names $\vec{x}$.

Definition 4.7 Let $\alpha = u^\varepsilon(\vec{x})$ be an action. For any behaviour $\mathcal{A} : I$ with $I = \{u : \varepsilon(\vec{s}), x_1 : s_1, \dots, x_n : s_n\}$, define $[\alpha]\mathcal{A}$ as the set of all processes P that respect the interface $\{u : \varepsilon(\vec{s})\}$ such that P has no infinite reduction and for all transitions $P \to^* \xrightarrow{\alpha} Q$, one has $Q \in \mathcal{A}$.

This is mostly a standard "must" modality, except that it forbids any other name than its subject to be present in the interface of the processes that satisfy it. This is very restrictive at first sight, but significant behaviours can still be expressed by combining such behaviours with the spatial connectives, as illustrated below. A consequence of this constraint is the duality between modalities with opposite polarities:

Proposition 4.8 *For any action α and any behaviour $\mathcal{A}$ with appropriate interface, $([\alpha]\mathcal{A})^\perp = [\bar{\alpha}](\mathcal{A}^\perp)$.*

This proves in particular that $[\alpha]\mathcal{A}$ does not need a bi-orthogonal to be a behaviour. These modalities also behave well with respect to subtyping:

Proposition 4.9 *Let α be an action and $(\mathcal{A}_i)$ a non-empty family of behaviours of the same interface, appropriate for α. Then $[\alpha] \bigwith_i \mathcal{A}_i = \bigwith_i [\alpha]\mathcal{A}_i$ and $[\alpha] \bigvee_i \mathcal{A}_i = \bigvee_i [\alpha]\mathcal{A}_i$. Moreover, $[\alpha]\mathbf{0} = \mathbf{1}$ and $[\alpha]\top = \perp$.*

Proposition 4.10 *Let α be an action, let $\mathcal{A}$ and $\mathcal{B}$ be two behaviour with appropriate interfaces, then for all $P \in \mathcal{A} \multimap \mathcal{B}$, $\alpha.P \in \mathcal{A} \multimap [\alpha]\mathcal{B}$.*

4.3 Replicated actions

Neither the spatial operators nor the modalities defined above allow for directly expressing properties of potentially infinite behaviours. We now define a new kind modality, using the previous operators, in order to capture semantically the behaviour of guarded replications. Informally, the behaviour of a process $!\alpha.P$ is that of processes that, after a transition labelled α, return to the same behaviour and produce an fresh instance of P. This can be formalized as the greatest fixed point of an appropriate operator, as follows:

Definition 4.11 Let $\mathcal{A}$ be a behaviour of interface I and α be an action with object $\vec{x} = \mathrm{dom}(I)$. Define the iteration operator $\mathcal{E}(X) = [\alpha](X \otimes \mathcal{A})$ and let $[!\alpha]\mathcal{A} = \bigcap_{k\in\mathbb{N}} \mathcal{E}^k(\top)$. Define the dual $[?\alpha]\mathcal{A} = ([!\bar{\alpha}]\mathcal{A}^\perp)^\perp$.

The idea is that a process in $P \in [!w(\vec{x})]\mathcal{A}$ is a server located at w that serves mutually independent processes of type $\mathcal{A}$. Then if we duplicate the port where P listens, by writing $(\boldsymbol{\nu} w)(u{\to}w \mid v{\to}w \mid P)$, the result serves mutually independent processes of type $\mathcal{A}$ on the chanels u and v, which is indistinguishable from two independent servers providing processes of the same type at u and v. That is the meaning of the following contraction lemma:

Lemma 4.12 *For any distinct channels u, v, w and any behaviour $\mathcal{A} : I$ with $\mathrm{dom}(I) = \vec{x}$, $u{\to}w \mid v{\to}w \in [!w(\vec{x})]\mathcal{A} \multimap [!u(\vec{x})]\mathcal{A} \otimes [!v(\vec{x})]\mathcal{A}$.*

Proof (sketch). With the notations of definition 4.11, consider the behaviour $\mathcal{C}_{i,j} = \mathcal{E}_u^i(\top)^\perp \parr \mathcal{E}_v^j(\top)^\perp$, and let $F = u{\to}w \mid v{\to}w$. Then show by recurrence that $\mathcal{C}_{i,j} \cdot F \subseteq \mathcal{E}_w^{i+j}(\top)^\perp$ for all integers i, j. Computing the upper bound over all i, j, deduce $([?\bar{u}(\vec{x})]\mathcal{A}^\perp \parr [?\bar{v}(\vec{x})]\mathcal{A}^\perp) \cdot F \subseteq [?\bar{w}(\vec{x})]\mathcal{A}^\perp$. □

This lemma proves the adequacy of contraction, from which we deduce the adequacy of the promotion rule of linear logic, with the usual constraint that the context is supposed to consist explicitly of non-linear actions, i.e. $[?\alpha]$ modalities. The adequacy of the weakening and dereliction rules derive directly from the definition of replicated modalities.

Proposition 4.13 *Let $\mathcal{A}$ and $(\mathcal{B}_i)_{1\leqslant i\leqslant n}$ be behaviours with disjoint interfaces and α, β_i be appropriate actions for those behaviours. Let $\mathcal{C} = [?\beta_1]\mathcal{B}_1 \parr \cdots \parr [?\beta_n]\mathcal{B}_n$. For each $P \in \mathcal{A} \parr \mathcal{C}$, we have $!\alpha.P \in [!\alpha]\mathcal{A} \parr \mathcal{C}$.*

Proof. The behaviour $\mathcal{C}^\perp$ is generated by the $Q = Q_1 \mid \cdots \mid Q_n$ with $Q_i \in [?\beta_i]\mathcal{B}_i$. Let $P \in \mathcal{A} \parr \mathcal{C}$, using the notations of definition 4.11, we prove by recurrence that for each k, $!\alpha.P \in \mathcal{E}^k(\top) \parr \mathcal{C}$. The case $k = 0$ is trivial since $\top$ is absorbing for $\parr$. Now assume the result holds for k; the only transition of $!\alpha.P$ is labelled α and leads to $!\alpha.P \mid P$, with $!\alpha.P \in \mathcal{E}^k(\top) \parr \mathcal{C}$ by induction hypothesis and $P \in \mathcal{A} \parr \mathcal{C}$. Let σ be a renaming that maps the subjects of the β_i to fresh names, then we have $!\alpha.P \mid P\sigma \in (\mathcal{E}^k(\top) \parr \mathcal{C}) \otimes (\mathcal{A} \parr \mathcal{C}\sigma)$ which is included in $(\mathcal{E}^k(\top) \otimes \mathcal{A}) \parr \mathcal{C} \parr \mathcal{C}\sigma$ by lemma 4.5. Applying the contraction lemma to each pair $\mathcal{B}_i \parr \mathcal{B}_i\sigma$ in $\mathcal{C} \parr \mathcal{C}\sigma$ we obtain that $(\mathcal{C} \parr \mathcal{C}\sigma)\sigma^{-1} \subseteq \mathcal{C}$,

which proves $!\alpha.P \mid P \in (\mathcal{E}^k(\top) \otimes \mathcal{A}) \parr \mathcal{C}$. For each $Q \in \mathcal{C}^\perp$, there can be no interaction between $!\alpha.P$ and Q before a transition on α; after such a transition from $!\alpha.P \cdot Q$, the reduct is $(!\alpha.P \mid P) \cdot Q'$ for some $Q \to^* Q' \in \mathcal{C}^\perp$, so this term is in $[\alpha](\mathcal{E}^k(\top) \otimes \mathcal{A}) = \mathcal{E}^{k+1}(\top)$, as a consequence we get $!\alpha.P \in \mathcal{E}^{k+1}(\top) \parr \mathcal{C}$. So this holds for each k and we have $!\alpha.P \in \bigcap_k \mathcal{E}^k(\top) \parr \mathcal{C} = [!\alpha]\mathcal{A} \parr \mathcal{C}$ since $\parr$ distributes over intersections. □

5 Linear type system

The behavioural connectives from the previous section are defined from semantic observations on processes. Their interactions and contextual properties make them a proper basis for defining a logic over processes, i.e. a proper type system, in which formulas are defined as anonymous versions of the behavioural connectives, and types are sequents made of named (or localized) formulas.

Definition 5.1 An *anonymous interface* is a finite sequence i of channel types, as of definition 2.4. The concatenation of anonymous interfaces i and j is noted $i; j$. Given an anonymous interface i and a vector $\vec{x}$ of distinct names of the same length, the instance of i at $\vec{x}$ is the interface $\vec{x} : i = \{x_k : i_k\}_{1 \leqslant k \leqslant |i|}$.

Definition 5.2 For each anonymous interface i, we assume a countable set of type variables, ranged over by X^i. Formulas are built on the following syntax:

$$\begin{aligned} A, B := A \otimes B \mid \mathbf{1}^i \mid {\downarrow}A \mid\ !A \mid \exists X^i.A \mid X^i \\ A \parr B \mid \bot^i \mid {\uparrow}A \mid\ ?A \mid \forall X^i.A \mid X^{i\perp} \end{aligned}$$

Each formula has an anonymous interface defined as

$$\begin{array}{llll} I(X^i) = i & & I(c^i) = i & \text{for } c \in \{\mathbf{1}, \bot\} \\ I(\dagger A) = {\downarrow}I(A) & \text{for } \dagger \in \{\downarrow, !\} & I(A \odot B) = I(A); I(B) & \text{for } \odot \in \{\otimes, \parr\} \\ I(\dagger A) = {\uparrow}I(A) & \text{for } \dagger \in \{\uparrow, ?\} & I(\nabla X^i.A) = I(A) & \text{for } \nabla \in \{\exists, \forall\} \end{array}$$

A type Γ is a sequence $\vec{x}_1 : A_1, \ldots, \vec{x}_n : A_n$ where each $\vec{x}_i$ has the length of $I(A_i)$ and a typing judgement has the form $P \vdash \Gamma$.

Thus types are named sequents of second-order multiplicative exponential linear logic, with two extra "shifting" modalities $\downarrow$ and $\uparrow$, dual to each other. Negation $(\cdot)^\perp$ is defined syntactically in the usual way, the operators on the first line being dual to those on the second line.

Definition 5.3 A process P has type Γ if the typing judgement $P \vdash \Gamma$ is derivable by the rules of table 3 and the exchange rule.

Definition 5.4 A *valuation* is a function $\mathcal{V}$ that associates each type variable X^i, for each vector $\vec{x}$ of $|i|$ distinct names, to a behaviour $\mathcal{V}(X^i)_{\vec{x}}$ of interface

Actions:

$$\frac{}{u{\to}v \vdash u : {\downarrow}A^\perp, v : {\uparrow}A} \qquad \frac{}{u{\to}v \vdash u : !A^\perp, v : ?A}$$

$$\frac{P \vdash \Gamma, \vec{x} : A}{u(\vec{x}).P \vdash \Gamma, u : {\downarrow}A} \qquad \frac{P \vdash \Gamma, \vec{x} : A}{\bar{u}(\vec{x}).P \vdash \Gamma, u : {\uparrow}A}$$

Multiplicatives:

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

Exponentials:

$$\frac{P \vdash ?\Gamma, \vec{x} : A}{!u(\vec{x}).P \vdash ?\Gamma, u : !A} \qquad \frac{P \vdash \Gamma, u : {\uparrow}A}{P \vdash \Gamma, u : ?A}$$

$$\frac{P \vdash \Gamma, u : ?A, v : ?A}{P[u/v] \vdash \Gamma, u : ?A} \qquad \frac{P \vdash \Gamma}{P \vdash \Gamma, u : ?A}$$

Quantifiers:

$$\frac{P \vdash \Gamma, \vec{u} : A}{P \vdash \Gamma, \vec{u} : \exists X^i.A[X^i/B]}\, I(B) = i \qquad \frac{P \vdash \Gamma, \vec{u} : A}{P \vdash \Gamma, \vec{u} : \forall X^i.A}\, X^i \notin \mathrm{fv}(\Gamma)$$

Constants:

$$\frac{}{1 \vdash \mathbf{1}^i} \qquad \frac{P \vdash \Gamma}{P \vdash \Gamma, \bot^i}$$

Cut:

$$\frac{P \vdash \Gamma, \vec{x} : A \qquad Q \vdash \vec{x} : A^\perp, \Delta}{(\boldsymbol{\nu}\vec{x})(P \mid Q) \vdash \Gamma, \Delta}$$

Table 3
The typing rules.

$\vec{x} : i$, with the natural constraint $\mathcal{V}(X^i)_{\vec{y}} = \mathcal{V}(X^i)_{\vec{x}}[\vec{y}/\vec{x}]$ for all $\vec{x}$ and $\vec{y}$. For such a valuation, the interpretation $[\![A]\!]_{\vec{x}}$ of a type A, parametrized by a vector $\vec{x}$ of length $|I(A)|$ consisting of distinct names, is defined inductively as

$$[\![X^i]\!]_{\vec{x}} = \mathcal{V}(X^i)_{\vec{x}} \qquad [\![X^{i\perp}]\!]_{\vec{x}} = \mathcal{V}(X^i)^\perp_{\vec{x}}$$
$$[\![\mathbf{1}^i]\!]_{\vec{x}} = \mathbf{1}_{\vec{x}:i} \qquad [\![\bot^i]\!]_{\vec{x}} = \bot_{\vec{x}:i}$$
$$[\![{\downarrow}A]\!]_u = [u(\vec{x})][\![A]\!]_{\vec{x}} \qquad [\![{\uparrow}A]\!]_u = [\bar{u}(\vec{x})][\![A]\!]_{\vec{x}}$$
$$[\![!A]\!]_u = [!u(\vec{x})][\![A]\!]_{\vec{x}} \qquad [\![?A]\!]_u = [?\bar{u}(\vec{x})][\![A]\!]_{\vec{x}}$$
$$[\![A \otimes B]\!]_{\vec{x}\vec{y}} = [\![A]\!]_{\vec{x}} \otimes [\![B]\!]_{\vec{y}} \qquad [\![A \parr B]\!]_{\vec{x}\vec{y}} = [\![A]\!]_{\vec{x}} \parr [\![B]\!]_{\vec{y}}$$
$$[\![\exists X^i.A]\!]_{\vec{x}} = \bigvee\nolimits_{\mathcal{V}(X^i):i} [\![A]\!]_{\vec{x}} \qquad [\![\forall X^i.A]\!]_{\vec{x}} = \bigwedge\nolimits_{\mathcal{V}(X^i):i} [\![A]\!]_{\vec{x}}$$

The interpretations of $\exists X^i.A$ and $\forall X^i.A$ are the upper and lower bounds of the interpretations of A in valuations obtained from $\mathcal{V}$ by replacing $\mathcal{V}(X^i)$ by arbitrary behaviours. A process P is said to realize a type $\Gamma = \vec{x}_1 : A_1, \dots, x_n : A_n$, written $P \Vdash \Gamma$, if $P \in [\![A_1]\!]_{\vec{x}_1} \parr \cdots \parr [\![A_n]\!]_{\vec{x}_n}$.

It can be easily checked that any derivable type is non-degenerate, notably

because behaviours $[\alpha]\mathcal{A}$ always are, and from this and the propositions of section 4 we get adequacy and termination of typed terms:

Theorem 5.5 *For any valuation of the type variables, $P \vdash \Gamma$ implies $P \Vdash \Gamma$.*

Proof. Adequacy is proved by induction on the typing rules, using the various propositions of the previous section. □

Theorem 5.6 *All typable terms terminate.*

Proof. An equivalent formulation is that all types derivable by the rules of table 3 are non-degenerate. This is guaranteed by the fact that the interpretation of any $\downarrow A$ is non-degenerate (by definition) and that this property is clearly preserved by all connectives. □

This type system also enjoys a "weak" subject reduction, in the sense that $P \vdash \Gamma$ and $P \to P'$ imply the existence of a process $P'' \equiv P'$ such that $P'' \vdash \Gamma$. The proof requires a handful of lemmas to describe the type of processes after observable transitions, and we have to leave it out for lack of space.

Subtyping

This type system is a rather restrictive fragment of what we can get from our behavioural operators. For instance, the only form of subtyping we have in the rules of table 3 is quantification, but the type system can be enriched with intersection and union types. The language of formulas gets extended with the dual constructs $A \wedge B$ and $A \vee B$, with the constraint that A and B have the same interface in well-formed types, and the following rules apply:

$$\frac{P \vdash \Gamma, \vec{u} : A}{P \vdash \Gamma, \vec{u} : A \vee B}\, I(A) = I(B) \qquad \frac{P \vdash \Gamma, \vec{u} : A \qquad P \vdash \Gamma, \vec{u} : B}{P \vdash \Gamma, \vec{u} : A \wedge B}\, I(A) = I(B)$$

Concurrent realizers

Because of the constraints our typing imposes on communication schemes (each channel has at most one input capability, channels with multiple output capabilities have one replicable input capability), typable terms are always confluent. Nevertheless, derivable behaviours may contain processes that are not confluent, and those may be equally interesting from the point of view of concurrency. A simple example is the type of co-contraction:

Proposition 5.7 $w{\to}u \mid w{\to}v \Vdash (u : !A \otimes v : !A) \multimap (w : !A)$.

Proof. The proof uses mostly the same technique as that of lemma 4.12. □

This gives a symmetric (and non-deterministic) implementation of the type $!A \otimes !A \multimap !A$ that takes two servers of the same type $!A$ on channels u and v and lets them listen on the same channel w. Although the typing judgement $w{\to}u \mid w{\to}v \vdash (u : !A \otimes v : !A) \multimap (w : !A)$ is not derivable with the rules of table 3, the type itself is actually provable, but the only way is to forget

one side of the tensor (by weakening) and keeping the other unchanged. In other words, standard proofs remove the non-determinism by making always the same choice. This non-deterministic implementation can be rephrased by the following rule:

$$\frac{P \vdash \Gamma, u : !A \qquad Q \vdash \Delta, u : !A}{P \mid Q \vdash \Gamma, \Delta, u : !A}$$

Non-determinism is indeed built in our construction, in the sense that if $P \Vdash \Gamma$ and $Q \Vdash \Gamma$ we always have $P \oplus Q \Vdash \Gamma$, where $P \oplus Q$ is an internal choice, for instance defined as $(u)(u.P \mid u.Q \mid \bar{u})$.

6 Embedding classical systems

The type system we get for processes is a variant of linear logic. As such, it is a tool of choice for the study of intuitionistic and classical logical systems. We now describe a generic way of embedding those logics into our type system, leading to canonical typed embeddings of λ and $\lambda\mu$-calculi into processes. We will focus on sequent calculi using only implication and quantification, with associated term calculi. The formulas and the sequents are thus generated by the following grammar:

formulas:	$A, B := X \mid A \to B \mid \forall X.A$
sequents:	$x_1 : A_1, \dots, x_n : A_n \vdash \alpha_1 : B_1, \dots, \alpha_p : B_p$

We use the standard deductions rules in multiplicative form:

$$\frac{}{X \vdash X} \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta} \qquad \frac{\Gamma_1 \vdash A \to B, \Delta_1 \qquad \Gamma_2 \vdash A, \Delta_2}{\Gamma_1, \Gamma_2 \vdash B, \Delta_1, \Delta_2}$$

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

Contractions and weakenings are allowed on both sides of the sequents. Those naturally correspond to typing rules for $\lambda\mu$-calculus, assuming some formula on the right-hand side is chosen as the active one, or for λ-calculus for intuitionistic systems where right-hand sides always have just one formula.

Definition 6.1 A modality is a non-empty word on $\{\downarrow, \uparrow, !, ?\}$. A *modal translation* $(\cdot)^*$ is a pair of modalities (μ, ν) such that formulas are translated as

$$(X)^* = X \qquad (A \to B)^* = \mu A^* \multimap \nu B^* \qquad (\forall X.A)^* = \forall X.A^*$$

and a sequent $x_1 : A_1, \dots, x_n : A_n \vdash \alpha_1 : B_1, \dots, \alpha_p : B_p$ is translated as

$$\vdash x_1 : (\mu A_1^*)^\perp, \dots, x_n : (\mu A_n^*)^\perp, \alpha_1 : \nu B_1^*, \dots, \alpha_p : \nu B_p^*$$

together with a translation of deduction rules.

Such translations of classical systems into linear logic have been studied from the logical side by Danos, Joinet and Schellinx [12], leading to the identification of dual translations that were later formalized as the logical counterpart of call-by-name and call-by-value reduction strategies [10]. We now adapt this study to get associated embeddings of the underlying calculi into processes.

Our axiom rules imply the introduction of at least one modality, so μ and ν must have a non-empty common suffix for any valid translation. Moreover, since left-hand sides of sequents allow contractions and weakenings, μ must always begin with !, and in classical systems ν must begin with ? since contraction and weakening are also allowed on the right. In the sequel we will focus on the representative case where one of μ, ν is a suffix of the other.

Definition 6.2 Given a modality μ, and names u and $\vec{x}$, define the *protocol* $u^\mu(\vec{x}).P$ as $u^\downarrow(\vec{x}).P = u(\vec{x}).P$, $u^!(\vec{x}).P = !u(\vec{x}).P$, $u^\uparrow(\vec{x}).P = u^?(\vec{x}).P = \bar{u}(\vec{x}).P$, and inductively $u^{\mu\dagger}(\vec{x}).P = u^\mu(v).v^\dagger(\vec{x}).P$ for a fresh name v.

Observe that this translation of modalities into interaction protocols is typed, since $P \vdash ?\Gamma, \vec{x} : A$ always implies $u^\mu(\vec{x})P \vdash ?\Gamma, u : \mu A$. Subsequently, since modalities are the only rules that introduce actions, translating a proof system into processes simply means placing the right protocols at the right places, between forwarders (as axioms) and compositions (as cuts).

For the computational part of proof systems, we use the notations of $\lambda\mu$-calculus [21]. A term $t : C$ with free λ-variables $x_i : A_i$ and μ-variables $\alpha_j : B_j$ will be translated into a process $[\![t]\!]_\beta$ with an interface composed of channels $x_i : (\mu A_i^*)^\perp$, $\alpha_j : \nu B_j^*$ and $\beta : \nu C^*$. The active output β is purely arbitrary, indeed the first notable property of the translations is the following:

$$[\![\mu\alpha[\beta]t]\!]_\gamma = [\![t]\!]_\beta[\gamma/\alpha]$$

From the typing of protocols by modalities, we also deduce the translation of the λ-abstraction rule, which deduces $\alpha : \nu(\mu A \multimap \nu B)$ from $x : (\mu A)^\perp, \beta : \nu B$:

$$[\![\lambda x.t]\!]_\alpha = \alpha^\nu(x\beta).[\![t]\!]_\beta$$

Call-by-name

The translation of the other rules depends on the modalities. The first case is when ν is a suffix of μ, with the symptomatic example $\mu = !?$ and $\nu = ?$, which corresponds to the system LKT in Danos-Joinet-Schellinx, or call-by-name in λ-calculus parlance. Writing $\mu = \xi\nu$, the axiom rule becomes

$$\dfrac{\dfrac{}{y{\to}\alpha \vdash y : (\nu A)^\perp, \alpha : \nu A}}{x^{\bar{\xi}}(y).(y{\to}\alpha) \vdash x : (\mu A)^\perp, \alpha : \nu A}$$

which can be rewritten as $x^{\bar{\xi}}\langle\alpha\rangle$ using the simplified notation from proposition 2.6. The application rule is translated with the first proof in table 4, and

Call-by-name (with $\mu = \xi\nu$):

$$\dfrac{P \vdash ?\Gamma, \beta : \nu(\mu A \multimap \nu B) \qquad \dfrac{\dfrac{Q \vdash ?\Delta, \gamma : \nu A}{x^{\xi}(\gamma).Q \vdash ?\Delta, x : \mu A} \qquad \dfrac{}{y{\to}\alpha \vdash y : \bar\nu B^{\perp}, \alpha : \nu B}}{x^{\xi}(\gamma).Q \mid y{\to}\alpha \vdash ?\Delta, xy : \mu A \otimes \bar\nu B^{\perp}, \alpha : \nu B}}{\dfrac{\beta^{\bar\nu}(xy).(x^{\xi}(\gamma).Q \mid y{\to}\alpha) \vdash ?\Delta, \beta : \bar\nu(\mu A \otimes \bar\nu B^{\perp}), \alpha : \nu B}{(\boldsymbol{\nu}\beta)(P \mid \beta^{\bar\nu}(xy).(x^{\xi}(\gamma).Q \mid y{\to}\alpha)) \vdash ?\Gamma, ?\Delta, \alpha : \nu B}}$$

Call-by-value (with $\nu = \xi\mu$):

$$\dfrac{\dfrac{P \vdash ?\Gamma, \beta : \nu(\mu A \multimap \nu B) \qquad \dfrac{\dfrac{\dfrac{}{x{\to}y \vdash x : \mu A, y : \bar\mu A^{\perp}} \qquad \dfrac{}{z{\to}\alpha \vdash z : \bar\nu B^{\perp}, \alpha : \nu B}}{x{\to}y \mid z{\to}\alpha \vdash xz : \mu A \otimes \nu B^{\perp}, y : \bar\mu A^{\perp}, \alpha : \nu B}}{\beta^{\bar\nu}\langle y\alpha\rangle \vdash \beta : \bar\nu(\mu A \otimes \nu B^{\perp}), y : \bar\mu A^{\perp}, \alpha : \nu B}}{(\boldsymbol{\nu}\beta)(P \mid \beta^{\bar\nu}\langle y\alpha\rangle) \vdash ?\Gamma, y : \bar\mu A^{\perp}, \alpha : \nu B} \qquad Q \vdash ?\Delta, \gamma : \nu A}{\dfrac{\gamma^{\bar\xi}(y).(\boldsymbol{\nu}\beta)(P \mid \beta^{\bar\nu}\langle y\alpha\rangle) \vdash ?\Gamma, \gamma : \bar\nu A^{\perp}, \alpha : \nu B}{(\boldsymbol{\nu}\gamma)(\gamma^{\bar\xi}(y).(\boldsymbol{\nu}\beta)(P \mid \beta^{\bar\nu}\langle y\alpha\rangle) \mid Q) \vdash ?\Gamma, ?\Delta, \alpha : \nu B}}$$

Table 4
Call-by-name and call-by-value translations of application.

with $\mu = !?$ and $\nu = ?$, we get the translation of call-by-name $\lambda\mu$-calculus:

$$\begin{aligned}
[\![x]\!]^n_\alpha &= \bar x\langle\alpha\rangle \\
[\![\lambda x.t]\!]^n_\alpha &= \bar\alpha(x\beta).[\![t]\!]^n_\beta \\
[\![t\,u]\!]^n_\alpha &= (\boldsymbol{\nu}\beta)\big([\![t]\!]^n_\beta \mid !\beta(xy).(!x(\gamma).[\![u]\!]^n_\gamma \mid y{\to}\alpha)\big)
\end{aligned}$$

By choosing for ν a linear modality, we can get a simplified variant for intuitionistic logic. Indeed, with $\mu = !\downarrow$ and $\nu = \downarrow$, we get exactly Milner's translation of pure λ-calculus into polyadic π-calculus:

$$\begin{aligned}
[\![x]\!]^{in}_\alpha &= \bar x\langle\alpha\rangle \\
[\![\lambda x.t]\!]^{in}_\alpha &= \alpha(x\beta).[\![t]\!]^{in}_\beta \\
[\![t\,u]\!]^{in}_\alpha &= (\boldsymbol{\nu}\beta x)\big([\![t]\!]^{in}_\beta \mid !x(\gamma).[\![u]\!]^{in}_\gamma \mid \bar\beta\langle x\alpha\rangle\big)
\end{aligned}$$

Call-by-value

The dual case is when the modality μ is a suffix of ν. Now we write $\nu = \xi\mu$, where μ begins with !, and we get the axiom by the rule

$$\dfrac{\dfrac{}{y{\to}x \vdash x : (\mu A)^{\perp}, y : \mu A}}{\alpha^{\xi}(y).(y{\to}x) \vdash x : (\mu A)^{\perp}, \alpha : \nu A}$$

which simplifies into $\alpha^{\xi}\langle x\rangle$, the reverse of the previous case. The generic application rule is the second proof in table 4. The symptomatic instance now

is $\mu = !$ and $\nu = ?!$, which corresponds to LKQ in Danos-Joinet-Schellinx and to call-by-value in λ-calculus parlance, leads to the following:

$$\begin{aligned}
[\![x]\!]^v_\alpha &= \bar\alpha\langle x\rangle \\
[\![\lambda x.t]\!]^v_\alpha &= \bar\alpha(y).!y(x\beta).[\![t]\!]^v_\beta \\
[\![t\,u]\!]^v_\alpha &= (\boldsymbol{\nu}\gamma)\Big(!\gamma(x).(\boldsymbol{\nu}\beta)\big([\![t]\!]^v_\beta \mid !\beta(w).\bar w\langle x\alpha\rangle\big) \;\Big|\; [\![u]\!]^v_\gamma\Big)
\end{aligned}$$

Not surprisingly, this is exactly the translation of call-by-value $\lambda\mu$-calculus into π-calculus formulated by Honda, Yoshida and Berger in their recent work [6,18]. The simplest intuitionistic version is obtained by taking for ξ the empty modality (which is linear), and thus $\mu = \nu = !$, which gives:

$$\begin{aligned}
[\![x]\!]^{iv}_\alpha &= \alpha{\to}x \\
[\![\lambda x.t]\!]^{iv}_\alpha &= !\alpha(x\beta).[\![t]\!]^{iv}_\beta \\
[\![t\,u]\!]^{iv}_\alpha &= (\boldsymbol{\nu}\beta\gamma)\big([\![t]\!]^{iv}_\beta \mid [\![u]\!]^{iv}_\gamma \mid \bar\beta\langle\gamma\alpha\rangle\big)
\end{aligned}$$

For all those translations, the rules for quantifiers are translated unchanged from classical types into linear types, thus we get soundness by construction:

Proposition 6.3 *For any translation $(\cdot)^*$ with modalities (μ,ν), for any typed $\lambda\mu$-term $\Gamma \vdash t : A, \Delta$, the translation of t is typed $[\![t]\!]^*_\alpha \vdash (\mu\Gamma^*)^\perp, \alpha : \nu A^*, \nu\Delta^*$.*

As a by-product, this gives a general normalization result for typed $\lambda\mu$-terms for the reduction strategies induced by the translations. The precise description of those strategies is beyond the scope of this paper, but previous work on the topic shows that they are strongly related to evaluation by abstract machines.

7 Conclusions and future works

Int this paper, we develop a powerful approach to testing semantics using realizability techniques. At the core is the notion of orthogonality, here defined as non-divergence of the interaction between processes. From this we get a meaningful model of multiplicative-exponential linear logic, with connectives defined by spatial and temporal observations, and a type system that ensures termination. This sketches the foundation of a Curry-Howard connection between linear logic and process calculi (with a rather restricted form of concurrency at present), and this connection integrates well with similar interpretations of intuitionistic and classical logics, notably by generalising previoulsy known embeddings of λ-calculi into π-calculus.

The technique can be painlessly extended to a calculus with guarded choice, whose typing will naturally extend the Curry-Howard interpretation to additive connectives, and thus full second order linear logic. Other observations can be handled with the same methods, notably may- and must-testing. Although the calculus we use here is formulated in order to get appropriate

properties, the type system can be applied to other similar name-passing calculi, like variants of polyadic π and fusion calculi. A challenging direction, although largely speculative, would then be to extend the rules and connectives of logic in order to capture behaviours that are too concurrent for pure linear logic, as suggested at the end of section 5.

Although the construction of behavioural connectives has intuitions from modal and spatial logics, precise links with those formalisms are yet to be made. Interesting comparisons are expected with previous work on the semantic links between relevant logics and intuitionsitic linear logic [11], and with proof systems like the logic of bunched implication [20]. In a different perspective, the typing rules induce translations of proof-nets into processes that are expected to be closely related to previous encodings, as developped by Ambramsky [3] and Bellin and Scott [5].

The algebraic structure of behaviours suggests many directions in which the semantics of concurrent processes can be studied, in relation with linear logic and its models. Links with game semantics could be fruitfully studied, probably by studying interaction traces with the same techniques as here. The approach of the Geometry of Interaction [16] seems applicable to the present case, and could provide new kinds of semantics for concurrent processes. The model could also provide intuitions on recent developments in linear logic, notably about differential λ-calculus [13] and its concurrent flavour.

References

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

[2] Samson Abramsky. Interaction categories. In Geoffrey L. Burn, Simon J. Gay, and Mark Ryan, editors, *Theory and Formal Methods*, Workshops in Computing, pages 57–69. Springer Verlag, 1993.

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

[4] Samson Abramsky. Process realizability. In Lars Birkedal, Jaap van Oosten, Giuseppe Rosolini, and Dana S. Scott, editors, *Electronic Notes in Theoretical Computer Science*, volume 23. Elsevier, 2001.

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

[6] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the π-calculus. In *Proceedings of FoSSaCS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 103–119. Springer Verlag, April 2003.

[7] Michele Boreale and Rocco de Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, 1995.

[8] Luís Caires and Luca Cardelli. A spatial logic for concurrency (part II). *Theoretical Computer Science*, 322(3):517–565, 2004.

[9] Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 11–23. Springer Verlag, 1990.

[10] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of ICFP'00*, pages 233–243. ACM, 2000.

[11] Mads Dam. Process-algebraic interpretations of positive linear and relevant logics. *Journal of Logic and Computation*, 4:939–973, 1994.

[12] 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.

[13] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1):1–41, 2003.

[14] Philippa Gardner and Lucian Wischik. Explicit fusions. In Mogens Nielsen and Branislav Rovan, editors, *Proceedings of MFCS 2000*, volume 1893 of *Lecture Notes in Computer Science*, pages 373–382. Springer Verlag, 2000.

[15] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[16] Jean-Yves Girard. Geometry of interaction III: The general case. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, pages 329–389. Cambridge University Press, 1995.

[17] Jean-Yves Girard. Locus solum. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001.

[18] Kohei Honda, Nobuko Yoshida, and Martin Berger. Control in the π-calculus. In *Proceedings of the Fourth ACM-SIGPLAN Continuation Workshop*, 2004.

[19] Robin Milner, Joachim Parrow, and David Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.

[20] Peter O'Hearn and David Pym. The logic of bunched implication. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.

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

[22] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS'98*, pages 176–185, 1998.

[23] Davide Sangiorgi and David Walker. *Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.