Fundamenta Informaticae XXX (2013) 1000–1038
DOI 10.3233/FI-2012-0000
IOS Press

On the π-calculus and co-intuitionistic logic. Notes on logic for concurrency and λP systems

Gianluigi Bellin* [C]

Alessandro Menti*

Dipartimento di Informatica, Università degli Studi di Verona

Abstract. We reconsider work by Bellin and Scott in the 1990s on R. Milner and S. Abramsky's encoding of linear logic in the π-calculus and give an account of efforts to establish a tight connection between the structure of proofs and of the cut elimination process in multiplicative linear logic, on one hand, and the *input-output* behaviour of the processes that represent them, on the other, resulting in a proof-theoretic account of (a variant of) Chu's construction. But Milner's encoding of the linear lambda calculus suggests consideration of multiplicative *co-intuitionistic linear logic*: we provide a term assignment for it, a *calculus of coroutines* which presents features of concurrent and distributed computing. Finally, as a test case of its adequacy as a logic for *distributed computation*, we represent our term assignment as a *λP system*. We argue that translations of typed functional languages in concurrent and distributed systems (such as π-calculi or λP systems) are best typed with co-intuitionistic logic, where some features of computations match the logical properties in a natural way.

1. Introduction

Among the early applications of R. Milner's π-calculus [28] we find translations of *linear logic* and of the normalization of proofs in linear logic into the π-calculus, which were pursued both by S. Abramsky and by

*We thank Phil Scott and Tristan Crolard for their cooperation over the years on the topics of this paper. We had useful conversations with Alessandra Carbone, Giuditta Franco, Hugo Herbelin, Neil Jones, Damiano Macedonio, Vincenzo Manca, Andy Pitts and Carolyn Talcott. We thank an anonymous referee for helpful suggestions.

[C]Corresponding author
Address for correspondence: Gianluigi Bellin, Dipartimento di Informatica — Università degli Studi di Verona, Strada Le Grazie 15, 37134 Verona, Italy, e-mail: gianluigi.bellin@univr.it

R. Milner around 1991 and are documented in the papers by Abramsky [1] and by Bellin and Scott [10].[1] The π-calculus is meant as a paradigmatic language for concurrent and distributed computing in the same way as the λ-calculus is paradigmatic for functional programming; since the fundamental role of the typed λ-calculus is confirmed by the Curry-Howard correspondence with propositions of intuitionistic logic and proofs in natural deduction, so there was the expectation that linear logic might provide a meaningful type system for the π-calculus and normalization of proofs in linear logic might correspond to the behaviour of concurrent processes of the π-calculus in a natural way. The need for conceptual clarity in the foundations of concurrent and distributed computing is even more evident today, not only because, as Philip Wadler puts it, "with the advent of multicores, mobile phones, and server farms, we have all become concurrent programmers" [34], but also because our understanding of information processing in basic biological entities takes the form of distributed computing.

Linear logic [16] appeared as a good candidate for a type system for *concurrent* logical computations: Girard himself had presented the representation of derivations in classical linear logic in the system of *proof nets* as a way to realize a "parallelization of the syntax"; in [16] this goal was regarded as already achieved for the *multiplicative fragment* of linear logic and was proposed as a research project for larger parts of the system. Therefore the encodings in the π-calculus of linear logic proofs, and in particular of proof-nets, and the representations of proof normalization through the transformation of the corresponding π-calculus processes were regarded as interesting tests both of π-calculus expressivity and of the claim of linear logic to be a logic for concurrency.

Girard's notion of a "parallelization of the syntax" through the representation of proofs as *multiple conclusions* proof-nets $\mathcal{R}_{A_1,\ldots,A_n}$ is based on the principle that all conclusions $A_1, \ldots, A_n$ must be regarded as equivalent "interaction ports": this is made possible by the facts that linear negation is an *involution* (i.e., $A^{\perp\perp} = A$); thus, e.g., there is no reason to give "privileged access" to one of the two "ports" of an axiom $\overline{A\ A^{\perp}}$. Such interchangeability of conclusions is a feature of a *classical* logical system: the opposite is indeed true in intuitionistic logic; more generally, when the notion of *polarity* is introduced, the *information flow* in a logical deduction is given a definite direction. The point is delicate: within *intuitionistic* proofs the notion of an information flow from *input formulas* A_{I} in the elimination part of proof-branches to *output formulas* C_{O} in the introduction parts is already in Gentzen and Prawitz; but we learnt from Girard's proof-nets that a more complex information flow occurs within *classical* linear proofs as well. This idea, developed at length in section 5 of [10], was also studied at the time by François Lamarche [24]. Later the "information flow" was related to Chu's construction and to the abstract form of game-semantics, see Bellin [4]. But game-theory is a paradigm of sequential computation, rather than of distributed computing. Much progress in linear logic in recent years are based on game-semantics and use polarized logics and polarized proof-nets as basic tools in proof-theory. One could say that in its maturity linear logic has moved away form the initial interest in concurrency and distribute computing; thus today fresh new ideas are needed for a fruitful development of research in this direction.

Abramsky's [1] and Bellin and Scott's [10] papers witness the importance of linear logic in the early development of the π-calculus. The goal of obtaining a tight correspondence between proof normalization and process reduction under structural congruence only gave R. Milner a motivation for developing a *synchronous* version of the π-calculus: as stated in the introduction of [10], this version of the π-calculus was "purposely supporting some of the logical rewriting envisioned by Abramsky". In particular,

[1] In 1991-2 Gianluigi Bellin and Philip Scott were in Edinburgh, learning the π-calculus from Robin Milner; also they were well aware of the work by Samson Abramsky, who gave a lecture on it in Edinburgh in 1992.

contextual rewriting is generally required by normalisation in logic and in the lambda calculus but becomes impossible within a term P in the scope of a *guarding prefix* $x(\tilde{z})Q$ or $\overline{x}\langle\tilde{a}\rangle Q$, namely, when no interaction involving subterms of Q is allowed to start until the channel x has acted. To minimize syntactic restrictions to interaction, the congruence $\omega_1\omega_2 P \equiv \omega_2\omega_1 P$ was introduced for arbitrary prefixes ω_1 and ω_3, when no free variable becomes bound and no bound variable becomes free. But Milner also allowed the use of the "guarding dot" $\omega.P$, regarded as an independent operator. The translation was successful for the *multiplicative fragment* **MLL** of linear logic, where the "guarding dot" was not used, but in the *multiplicative and additive* fragment the representability of cut-elimination was seriously limited by the use of such a guard.

In the π-calculus there is an evident asymmetry between *receiving prefixes* $x(\tilde{z})Q$, that bind the variables $\tilde{z}$ in Q and *sending prefixes* $\overline{x}\langle\tilde{a}\rangle P$ which do not. In the case of *senders* there seems to be no reason to distinguish between $\overline{x}\langle\tilde{a}\rangle\overline{y}\langle\tilde{b}\rangle P$ and $\overline{y}\langle\tilde{b}\rangle\overline{x}\langle\tilde{a}\rangle P$ and even between $\overline{x}\langle\tilde{a}\rangle\overline{y}\langle\tilde{b}\rangle P$ and $\overline{y}\langle\tilde{b}\rangle\|\overline{x}\langle\tilde{a}\rangle\|P$, while *receivers* may impose an ordering, as in the case of $x(y)y(z)P$. About ten years later, e.g., in work by C. Laneve and B. Victor [25] (particularly in the second encoding) the asymmetry between *senders* and *receivers* is removed and binding is performed only by the ν-operator (*abstraction*).

However already in Bellin and Scott [10] the asymmetry between senders and receivers was regarded as a feature to exploit rather than an obstacle to eliminate. The goal was to simulate the *information flow*, which occurs within proofs, in the π-calculus translation, by representing *input* and *output formulas* with a *receiving* and *sending* process, respectively.

Abramsky's and Bellin and Scott's early work has been reconsidered and developed, among others, by E. Beffara [3]. In L. Caires and F. Pfenning [12] and in P. Wadler [34], π-calculus-like processes are typed with linear types regarded as *session types* and occur in the framework of a more standard system of functional programming. We remark that here too the context is intuitionistic, or polarized; however it is beyond the limits of this paper to examine these interesting new ideas in detail.

1.1. Plan of the paper

This paper consists of three parts. In Part I we recall Milner's Basic Synchronous π-calculus in the 1992 version (Section 2.1) and Abramsky's translation of multiplicative linear logic $\mathbf{MLL}^{\otimes\wp}$ in the π-calculus and we give an account the efforts in [10] and then in [4] to establish a tight connection between the *"flow of information"* in logical computation and the *input-output* behaviour of its π-calculus encoding. In order to represent the dynamics of cut-elimination faithfully, processes are required to reflect the structure of sequent derivations and of proof-nets; as it turns out that the *correctness conditions* for proof nets relate proof-nets for *classical* $\mathbf{MLL}^{\otimes\wp}$ with derivations in intuitionistic multiplicative linear logic with products $\mathbf{IMLL}^{\multimap\otimes\&}$, our investigation leads to a variant of Chu's construction, a functor mapping a *free $*$-autonomous category* into a *symmetric monoidal closed category with products* that somehow encodes all intuitionistic derivations induced by the correctness condition.

After remarking that Milner's translation of the (linear) λ-calculus in the π-calculus appears to invert the *"flow of information"* traditionally recognized in simply typed λ terms, we conjecture that π-calculus translations and CPS transforms involve a passage to a dual typing. Thus in Section 3 we define *multiplicative co-intuitionistic linear logic* $\mathbf{co}\text{-}\mathbf{IMLL}^{\diagdown\wp}$, a linear *calculus of coroutines* (Section 4.2) and a typing of this calculus through a term assignment to $\mathbf{co}\text{-}\mathbf{IMLL}^{\diagdown\wp}$ derivations.

In Part III we touch the issue whether our logical encodings and calculus of coroutines are suitable to represent not only *concurrent* but also *distributed* aspects of computation, by taking a look at *membrane*

computing. Our basic references are G. Păun [29, 30]. Păun's membrane computing, using *maximally parallel rewriting*, is paradigmatic as a distributed system; such features as (what we called) "remote broadcasting" of substitutions are familiar in biological computing. We translate the dual linear calculus into λP systems.

Our work can be compared with the encoding of the typed λ-calculus (and of Gödel's system **T**) into λP systems by Colson, Jonoska and Margenstern [14]. In the comparison we see immediately the advantage of translating the dual (linear) system rather than the (linear) λ calculus: here a redex for subtraction is immediately recognizable from the form of the p-term and does not need to be detected by going through the membrane structure. As a consequence, *any reduction strategy* can be implemented in a straightforward way, not the *leftmost reduction*.

In the Appendix (Section 7) we give an example of co-intuitionistic derivation *with cut* in the form of a assignment of terms of the linear calculus of coroutines to *Prawitz trees*, together with its translation in our system of membrane computing.

2. PART I: Abramsky and Milner's π-calculus translations of linear logic and Chu's construction.

We reconsider work in [10] and [4], aiming at a tight connection in π-calculus translations between the *input-output* behaviour of π-calculus processes and the *logical flow of information* in **MLL** proofs.

2.1. Basic Synchronous π-calculus

We review here the *basic synchronous π-calculus* presented by Robin Milner in the early 1990s.

We are given a countable family $\mathcal{X}$ of variables (*names*), denoted by $a, d, c, \ldots, x, y, z$; we denote vectors of names with $\tilde{x}$, $\tilde{y}$, etc.

Definition 2.1. (Grammar)
The π-calculus *processes* P are denoted by the expressions *(π-terms)* defined by the following grammar:

$$P, Q \ := \ \bullet \mid (\nu\tilde{x})P \mid (P \| Q) \mid \overline{x}\langle\tilde{y}\rangle P \mid x(\tilde{y})P$$

Here

(i) $\bullet$ denotes the null process. $(P\|Q)$ denotes the process resulting from P and Q acting concurrently.

(ii) A *prefix* of the form $\overline{x}\langle\tilde{y}\rangle$ *(sender)* or $x(\tilde{y})$ *(receiver)* denotes a *channel* named x through which the names in $\tilde{y}$ can be sent or received, respectively.

(iii) In the π-terms $x(\tilde{y})P$ and $(\nu\tilde{x})P$ the receiver $x(\tilde{y})$ and the expression $(\nu\tilde{y})$ *(hiding)* are *name-binding operators* and P is their *scope*.

Prefixes are denoted by π and ω denotes either a prefix or a hiding operator.

Definition 2.2. (Free and bound names)
The sets $fn(P)$ and $bn(P)$ of *free* and *bound* names in P are defined as follows:

(a) $fn(\bullet) = bn(\bullet) = \emptyset$;

(b) $fn(P\|Q) = fn(P) \cup fn(Q)$; $bn(P\|Q) = bn(P) \cup bn(Q)$;

(c) $fn(\overline{x}\langle\tilde{y}\rangle P) = \{x\} \cup \tilde{y} \cup fn(P)$; $bn(\overline{x}\langle\tilde{y}\rangle P) = bn(P)$;

(d) $fn(x(\tilde{y})P) = fn(P) \setminus \tilde{y} \cup \{x\}$; $bn(x(\tilde{y})P) = bn(P) \cup \tilde{y}$;

(e) $fn((\nu\tilde{y})P) = fn(P) \setminus \tilde{y}$; $bn((\nu\tilde{y})P) = bn(P) \cup \tilde{y}$.

We define α-equivalence, renaming of bound variables and capture-avoiding simultaneous substitution $P[\tilde{y}/\tilde{x}]$ as usual.

Definition 2.3. (Congruence)
A *congruence relation* $\equiv$ on π-terms is defined as follows:

1. $\omega_1\omega_2 P \equiv \omega_2\omega_1 P$ provided no free variable becomes bound and no bound variable becomes free.
2. $\omega(P\|Q) \equiv \omega P\|Q$, provided $bn(\omega) \cap fn(Q) = \emptyset$.
3. Parallel composition $\|$, regarded as a binary operator on processes, satisfies the axioms of a commutative monoid with unit $\bullet$.
4. $(\nu x)\bullet \equiv \bullet$.

Definition 2.4. (Basic 1-step reduction)

$$(\overline{x}\langle\tilde{y}\rangle P \parallel x(\tilde{z})Q \quad \succ_1 \quad (P\|Q[\tilde{y}/\tilde{z}])$$

Rewriting is *contextual and modulo* $\equiv$, so that for all contexts $\mathcal{C}$, if $P' \equiv P$ and $Q \equiv Q'$, then we have $P \succ Q \Rightarrow \mathcal{C}[P'] \succ_1 \mathcal{C}[Q']$.

Remark 2.5. It is important to notice that in this version of the π-calculus the notion of the *scope* of a prefix is essentially related to binding. For instance, since $\overline{x}\langle\tilde{y}\rangle(P\|Q) \equiv (\overline{x}\langle\tilde{y}\rangle P\|Q)$ we have $\overline{x}\tilde{y}x(\tilde{z}P \equiv (\bullet\|x(\tilde{z}P) \succ_1 P[\tilde{y}/\tilde{z}]$. To constrain the reduction process in the synchronous calculus one uses the *guarding* operator (*dot*): $\pi.P$.

Remark 2.6. We cannot survey here the recent literature in which ideas related to the *synchronous π-calculus* have been developed, but the paper by Laneve and Victor [25] is particularly relevant, as it shows how *guarding prefixes* can be encoded in a calculus where the only binding operator is abstraction (the ν-operator).

2.2. Abramsky's π-calculus processes for $\mathrm{MLL}^{\otimes\wp}$ revisited.

The following is Abramsky's encoding of classical multiplicative linear logic in the π-calculus as given in [10]. We write the π-calculus term in a special *"control area"* of the sequent and decorate the sequent formulas with free variables for the *ports* accessing the term. There is no type in the "control area" and thus no similarity with Girard's *"stoup"* for a distinguished formula in the sequent.

Logical rule	**π-translation**
axiom $\vdash I_{xy} \mid x\colon A^{\perp}, \overline{y}\colon A$	$I_{xy} = x(a)\overline{y}\langle a\rangle$
cut $\dfrac{\vdash F_{\tilde{x}x} \mid \tilde{x}\colon \Gamma, x\colon A^{\perp} \qquad \vdash G_{y\tilde{y}} \mid y\colon A, \tilde{y}\colon \Delta}{\vdash \mathbf{Cut}^{z}(F,G)_{\tilde{x}\tilde{y}} \mid \tilde{x}\colon \Gamma, \tilde{y}\colon \Delta}$	$\mathbf{Cut}^{z}(F,G)_{\tilde{x}\tilde{y}} = \nu z\big(F_{\tilde{x}x}[z/x] \parallel G_{\tilde{y}y}[z/y]\big)$
times $\dfrac{\vdash F_{\tilde{x}x} \mid \tilde{x}\colon \Gamma, x\colon A \qquad \vdash G_{y\tilde{y}} \mid y\colon B, \tilde{y}\colon \Delta}{\vdash \mathbf{Times}_{z}^{xy}(F,G)_{\tilde{x}\tilde{y}} \mid \tilde{x}\colon \Gamma, z\colon A\otimes B, \tilde{y}\colon \Delta}$	$\mathbf{Times}_{z}^{xy}(F,G)_{\tilde{x}\tilde{y}} = \nu xy\big(\overline{z}\langle xy\rangle\ (F_{\tilde{x}x} \parallel G_{y\tilde{y}})\big)$
par $\dfrac{\vdash F_{xy\tilde{x}} \mid x\colon A, y\colon B, \tilde{x}\colon \Gamma}{\vdash \mathbf{Par}_{z}^{xy}(F)_{\tilde{x}} \mid z\colon A\wp B, \tilde{x}\colon \Gamma}$	$\mathbf{Par}_{z}^{xy}(F)_{\tilde{x}} = z(xy)F_{xy\tilde{x}}$

2.3. The Cut Algebra for $\mathbf{MLL}^{\otimes\wp}$

Symmetric Reductions:

$$\mathbf{Cut}^{z}(F_{\tilde{x}}, G_{\tilde{y}}) = \mathbf{Cut}^{z}(G_{\tilde{y}}, F_{\tilde{x}}) \tag{1}$$

$$\mathbf{Cut}^{x}(F_{\tilde{v}x}, I_{xy}) \succ F_{\tilde{v}x}[y/x] \tag{2}$$

$$\mathbf{Cut}^{z}(\mathbf{Times}_{z}^{xy}(F_x, G_y), \mathbf{Par}_{z}^{xy}(H_{xy}))_{\tilde{x}\tilde{y}\tilde{w}} \succ \mathbf{Cut}^{y}(\mathbf{Cut}^{x}(F_x, H_{xy}), G_y)_{\tilde{x}\tilde{y}\tilde{w}} \tag{3}$$

$$\equiv \mathbf{Cut}^{y}(G_y, \mathbf{Cut}^{x}(F_x, H_{xy}))_{\tilde{x}\tilde{y}\tilde{w}} \tag{4}$$

Commutative Reductions: if $\mathbf{Par}_{v}^{cd}$ and $\mathbf{Times}_{v}^{cd}$ do not react with x,[2] and neither c nor d occur in H, then

$$\mathbf{Cut}^{x}(\mathbf{Par}_{v}^{cd}(F_{cdx}), H_x)_{\tilde{x}\tilde{w}} = \mathbf{Par}_{v}^{cd}(\mathbf{Cut}^{x}(F_{cdx}, H_x))_{\tilde{x}\tilde{w}} \tag{5}$$

$$\mathbf{Cut}^{x}(\mathbf{Times}_{v}^{cd}(F_c, G_{dx}), H_x)_{\tilde{x}\tilde{y}\tilde{w}} = \mathbf{Times}_{v}^{cd}(F_c, \mathbf{Cut}^{x}(G_{dx}, H_x))_{\tilde{x}\tilde{y}\tilde{w}} \text{ and} \tag{6}$$

$$\mathbf{Cut}^{x}(\mathbf{Times}_{v}^{cd}(F_{cx}, G_{dx}), H_x)_{\tilde{x}\tilde{y}\tilde{w}} = \mathbf{Times}_{v}^{cd}(\mathbf{Cut}^{x}(F_{cx}, H_x), G_d)_{\tilde{x}\tilde{y}\tilde{w}}.$$

Theorem 2.7. (Soundness)
(See [10].) Let $\mathcal{D}$ be a proof in $\mathbf{MLL}^{\otimes\wp}$ and let $\pi(\mathcal{D})$ be its π-calculus translation.

(i) If $\mathcal{D} \succ'_{\mathcal{D}}$ by a 1-step symmetric reduction in the cut-elimination process, then $\pi(\mathcal{D}) \succ_1 \pi(\mathcal{D}')$ in the synchronous π-calculus.

(ii) If $\mathcal{D} \succ'_{\mathcal{D}}$ by a 1-step commutative reduction in the cut-elimination process, then $\pi(\mathcal{D}) \equiv \pi(\mathcal{D}')$ in the synchronous π-calculus.

[2] In the context of this translation it suffice to assume that $c \neq x \neq d$.

2.4. Input-Output orientations on linear types.

Any π-calculus translation induces an *Input-Output orientation* C_I or C_O on the sequent formulas depending on whether the port x associated with C is a *receiver* or a *sender*. Thus the above definition yields the following I-O orientations:

$$\textbf{axiom} \qquad \vdash x\colon A^{\perp}_{\mathbf{I}}, \overline{y}\colon A_{\mathbf{O}} \qquad\qquad \textbf{cut} \qquad \frac{\vdash \Gamma_?, x\colon A^{\perp}_? \qquad \vdash y\colon A_?, \Delta_?}{\vdash \Gamma_?, \Delta_?}$$

$$\textbf{Times} \qquad \frac{\vdash \Gamma_?, x\colon A_? \qquad \vdash B_?, \Delta_?}{\vdash \Gamma_?, z\colon (A \otimes B)_{\mathbf{O}}, \Delta_?} \qquad\qquad \textbf{Par} \qquad \frac{\vdash x\colon A_?, y\colon B_?, \Gamma_?}{\vdash z\colon (A \wp B)_{\mathbf{I}}, \Gamma_?}$$

Remark 2.8. (Computational consistency)

1. There is total symmetry between $A^{\perp}$ and A in classical multiplicative linear logic, thus there is no reason why the port associated with $A^{\perp}$ should be an *input* and that of A an *output*. Hence

   $$\vdash I_{x,y} \mid x\colon A_{\mathbf{I}}, \overline{y}\colon A^{\perp}_{\mathbf{O}}$$

   is also an acceptable translation for an axiom.

2. Similarly there is no compelling reason why the port associated with the formulas $A \otimes B$ in a *times* rule should be an *output* and the port of $A \wp B$ in a *par* rule an *input*. The following translations are also acceptable:

   $$\mathbf{Times}^{xy}_z(F,G)_{\tilde{x}\tilde{y}} = z(xy)(F_{\tilde{x}x} \parallel G_{y\tilde{y}})) \qquad \mathbf{Par}^{xy}_z(F)_{\tilde{x}} \;=\; \nu xy(\overline{z}\langle xy\rangle F_{xy\tilde{x}}$$

3. Although the π-calculus translation does not put any syntactic constraint on the orientation of the two *cut formulas*, a substantial requirement for a correct simulation of the cut-elimination process, called *computational consistency* in [10], is that in the translation of a derivation

   > *in any cut and in any cut resulting from the cut-elimination process, the two cut formulas must receive opposite orientation.*

   Indeed if both cut-formulas $A^{\perp}$ and A are *inputs* or both *outputs* no interaction is possible: the translation yields a *computational deadlock*.

Thus the problem of characterizing possible Abramsky-style translations of $\mathbf{MLL}^{\otimes\wp}$ derivations into the *pi*-calculus becomes that of finding *computationally consistent* I-O orientations of derivations, namely, functions δ assigning an orientation $\delta(A) \in \{\mathbf{I}, \mathbf{O}\}$ to formula-occurrences A in a sequent derivation d such that *computational consistency* is satisfied in the cut-elimination process. As it stands, the problem makes sense only *globally* for orientations $\delta(d)$ *of derivations*: it cannot be solved *locally*, i.e., considering only the active formulas of inferences

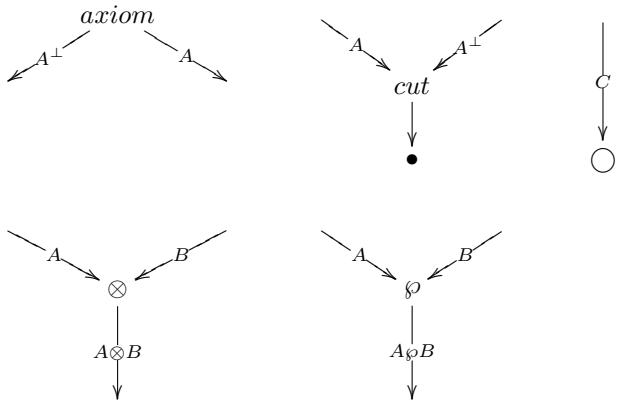
$$\frac{\delta(A) \quad \delta(B)}{\delta(A \bullet B)} \text{ with } \bullet = \otimes \text{ or } \wp.$$

2.5. Proof-nets and orientations from correctness conditions.

A new twist came from the study of *correctness conditions* for proof-nets. We recall the basic definitions.

Definition 2.9. A *proof structure* is a directed graph with at least one external point ◯ where each edge is typed and every node *(link)* has one of the forms in Table 1.[3]

Table 1. Links of $\mathbf{MLL}^{\otimes\wp}$ proof nets



Given a proof-structure $\mathcal{R}$, a *Danos-Regnier switching* s is a choice of an external node ◯ and of one of the incoming edges in each *par* node.

A *D-R graph* $s\mathcal{R}$ is the graph resulting from disconnecting the incoming edge not chosen by the switching from each *par* node.

A *proof-net* is a proof-structure such that for any switching s the D-R graph $s\mathcal{R}$ is acyclic and connected.

Theorem 2.10. (Girard's theorem, [16])
There exists a "context forgetful map" $(\ )^-$ from sequent derivations in $\mathbf{MLL}^{\otimes\wp}$ to proof-nets for with the following properties:

(i) If d is a sequent derivation of $\vdash \Gamma$, then $(d)^-$ is a proof-net with conclusions Γ;

(ii) *(sequentialization)* if $\mathcal{R}$ is a proof-net with conclusions Γ, then there is a sequent calculus derivation d of $\vdash \Gamma$ such that $\mathcal{R} = (d)^-$.

Following [10, 4] we consider orientations that are induced by *Girard trips* [16] on D-R graphs.

[3] In early work on proof-nets (e.g., in [16]) the equivalent *dual graph* representation of proof-structures was used, where vertices are labelled with formulas and edges with links of the form

$$\overline{A \quad A^\perp} \qquad \frac{A \quad A^\perp}{cut} \qquad \frac{A \quad B}{A \otimes B} \qquad \frac{A \quad B}{A \wp B}.$$

Definition 2.11. (i) Given a proof net $\mathcal{R}$ and a D-R graph $s\mathcal{R}$, a *Girard trip* on $s\mathcal{R}$ is the trip starting with the selected conclusion C and running through each edge twice in opposite directions until reaching C.

(ii) We say that the trip is *covariant on an edge* A if the *second passage* of the trip on A has the same direction as the edge A, otherwise the trip is *contravariant on* A.

(iii) The *I-O orientation* $\delta(\mathcal{R})$ *determined by the switching* s assigns $\delta(A) = \mathbf{O}$ if the Girard trip on $s\mathcal{R}$ is covariant on A and $\delta(A) = \mathbf{I}$ otherwise.

An interesting feature of such orientations lies in the following remark:

Proposition 2.12. (Bellin and de Wiele, 1991)
In a proof net $\mathcal{R}$ every orientation induced by a Girard trip makes the selected conclusion an *output*, all other conclusions are *inputs*. Each link is oriented in one of the following admissible ways:[4]

$$\frac{}{A^{\perp}_{\mathbf{I}} \quad A_{\mathbf{O}}}\ \text{axiom 1:} \qquad \frac{}{A_{\mathbf{I}} \quad A^{\perp}_{\mathbf{O}}}\ \text{axiom 2:} \qquad \frac{A_{\mathbf{O}} \quad A^{\perp}_{\mathbf{I}}}{}\ \text{cut 1:} \qquad \frac{A^{\perp}_{\mathbf{O}} \quad A_{\mathbf{I}}}{}\ \text{cut 2:}$$

$$\frac{A_{\mathbf{O}} \quad B_{\mathbf{O}}}{(A \otimes B)_{\mathbf{O}}}\ \text{times 1:} \qquad \frac{A_{\mathbf{O}} \quad B_{\mathbf{I}}}{(A \otimes B)_{\mathbf{I}}}\ \text{times 2:} \qquad \frac{A_{\mathbf{I}} \quad B_{\mathbf{O}}}{(A \otimes B)_{\mathbf{I}}}\ \text{times 3:}$$

$$\frac{A_{\mathbf{I}} \quad B_{\mathbf{I}}}{(A \wp B)_{\mathbf{I}}}\ \text{par 1:} \qquad \frac{A_{\mathbf{I}} \quad B_{\mathbf{O}}}{(A \wp B)_{\mathbf{O}}}\ \text{par 2:} \qquad \frac{A_{\mathbf{O}} \quad B_{\mathbf{I}}}{A \wp B_{\mathbf{O}}}\ \text{par 3:}$$

Does Proposition 2.12 solve the problem of finding *computationally consistent* orientations and of determining suitable π-calculus translations? Clearly not. Danos-Regnier switches make arbitrary choices on *par* links and thus may assign an I-O orientation to a cut-formula $A_{\mathbf{I}} \wp B_{\mathbf{O}}$ which is inconsistent with the one assigned by the trip algorithm to the other cut-formula $A^{\perp}_{\mathbf{I}} \otimes B^{\perp}_{\mathbf{O}}$.

However computationally consistent orientations can be obtained by suitable switches for links "above" cut-formulas, as one can prove by induction on the cut-elimination process for proof-nets (see [10], 5.4, Theorem 13.)

But the real interest of the above analysis of orientations from Girard trips lies in the fact that it opens the way to studying the relations between *classical* and *intuitionistic* linear logic, which are best understood in terms of *Chu's construction* [4].

2.6. Logical Input-Output orientations.

A different notion of orientation comes from the tradition of proof-theory and categorical logic. An intuitionistic derivation either in Natural Deduction **NJ** or in the sequent calculus **LJ** and decorated with λ-terms $x_1 \colon A_1, \ldots, x_n \colon A_n \vdash t \colon C$ represents a method to transform proofs of $A_1, \ldots, A_n$ into a proof of C. Here it is inescapable to think of the $x_i \colon A_i$ as the *input* and of $t \colon C$ as the *output* of a logical computation. Categorically, such a derivation is a morphism $t \colon A_1 \times \ldots \times A_n \to C$ in a Cartesian closed category with a source (*input*) and a target (*output*).

[4] Here we use the dual graphical notation where nodes are labelled by formulas and links by hedges, as in [10].

Similar ideas apply to *intuitionistic multiplicative linear logic* $\mathbf{IMLL}^{-\circ\otimes}$ with products (written in linear logic as *with* $\&$) and units, which can be formalized in the sequent calculus as in Table (2).

Table 2. Sequent Calculus $\mathbf{IMLL}^{-\circ\otimes\&}$

$$\textit{axiom}\qquad x\colon A \vdash x\colon A$$

$$\textit{cut}\qquad \frac{\tilde{x}\colon \Gamma \vdash t\colon A \qquad x\colon A, \tilde{y}\colon \Delta \vdash u\colon C}{\tilde{x}\colon \Gamma, \tilde{y}\colon \Delta \vdash u[t/x]C}$$

$$-\circ\text{-R}\qquad \frac{\tilde{x}\colon \Gamma \vdash t\colon A \qquad x\colon B, \tilde{y}\colon \Delta \vdash u\colon C}{\tilde{x}\colon \Gamma, f\colon A \multimap B, \tilde{y}\colon \Delta \vdash u[f(t)/x]\colon C}$$

$$-\circ\text{-L}\qquad \frac{\tilde{x}\colon \Delta, x\colon A \vdash t\colon B}{\tilde{x}\colon \Delta \vdash \lambda x.t\colon A \multimap B}$$

$$\otimes\text{-R}\qquad \frac{\tilde{x}\colon \Gamma_0 \vdash t_0\colon C_0 \qquad \tilde{y}\colon \Gamma_1 \vdash t_1\colon C_1}{\tilde{x}\colon \Gamma_0, \tilde{y}\colon \Gamma_1 \vdash t_0 \otimes t_1\colon C_0 \otimes C_1}$$

$$\otimes\text{-L}\qquad \frac{\tilde{x}\colon \Gamma, x\colon A_0, y\colon A_1 \vdash t\colon C}{\tilde{x}\colon \Gamma, z\colon A_0 \otimes A_1 \vdash \mathtt{let}\ z\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ t\colon C}$$

$$\&\text{-R}\qquad \frac{\tilde{x}\colon \Gamma \vdash t_0\colon C_0 \qquad \tilde{x}\colon \Gamma \vdash t_1\colon C_1}{\tilde{x}\colon \Gamma \vdash \langle t_0, t_1\rangle\colon C_0 \& C_1}$$

$$\&\text{-L}\qquad \frac{\tilde{x}\colon \Gamma, x\colon A_i \vdash t\colon C}{\tilde{x}\colon \Gamma, z\colon A_0 \& A_1 \vdash t[\pi_i z/x]\colon C}\ i = 0 \text{ or } 1.$$

$$\tilde{x}\colon \Gamma \vdash \star\colon \top \qquad\qquad \vdash *\colon \mathbf{1} \qquad\qquad \frac{\tilde{x} \vdash t\colon C}{y\colon \bot, \tilde{x}\colon \Gamma \vdash t\colon C}$$

What *computationally consistent* Girard trips do is to recover information which can be used to map proofs in *classical* multiplicative linear logic $\mathbf{MLL}^{\otimes\wp}$ into proofs in *intuitionistic* linear logic $\mathbf{IMLL}^{-\circ\otimes}$. Let $\mathcal{L}^{\otimes\wp}$ be the language of $\mathbf{MLL}^{\otimes\wp}$ built from on a set of atoms P, P', ...; let $\mathcal{L}^{-\circ\otimes}$ be the language of $\mathbf{IMLL}^{-\circ\otimes}$ on the set of atoms $P_{\mathbf{O}}$, $P_{\mathbf{I}}$, $P'_{\mathbf{O}}$, $P'_{\mathbf{I}}$, ... (a pair of atoms in $\mathcal{L}^{-\circ\otimes}$ for each atom in $\mathcal{L}^{\otimes\wp}$). Let s be a D-R switching for a proof-net $\mathcal{R}$ with conclusions Γ, C: we write them $\Gamma, \mathbf{C}$ if C is the conclusion selected by s.

Theorem 2.13. ([10], 5.4, Theorem 13.)

(i) Any computationally consistent orientation given by a Girard trip on a proof-net with conclusions $\Gamma, \mathbf{C}$ in $\mathbf{MLL}^{\otimes\wp}$ corresponds to (a class of) sequent calculus derivations of $\Gamma_{\mathbf{I}} \vdash C_{\mathbf{O}}$ in $\mathbf{IMLL}^{-\circ\otimes}$.

(ii) Every sequent derivation in $\mathbf{IMLL}^{-\circ\otimes}$ of $\Gamma_{\mathbf{I}} \vdash C_{\mathbf{O}}$ corresponds to a computationally consistent Girard trip on a proof-net with conclusions $\Gamma, \mathbf{C}$ in $\mathbf{MLL}^{\otimes\wp}$.

Proof:

(Idea of the proof of (i)). Given a *cut free* proof-net $\mathcal{R}$ and a Danos-Regnier switching for it, from a Girard trip on $s\mathcal{R}$ we obtain an orientation $\delta(\mathcal{R})$ with the properties given in by Proposition 2.12. The atoms of $\mathbf{IMLL}^{-\circ\otimes}$ formulas are a pair $P_{\mathbf{O}}, P_{\mathbf{I}}$ for each atom P of $\mathbf{MLL}^{\otimes\wp}$. Then non-atomic formulas in a $\mathcal{R}$ are translated inductively into $\mathcal{L}^{-\circ\otimes}$ formulas as follows:

$$(P^{\perp})_{\mathbf{O}} = P_{\mathbf{I}}, \qquad (P^{\perp})_{\mathbf{I}} = P_{\mathbf{O}} \quad \text{for } P \text{ atomic};$$
$$(A \otimes B)_{\mathbf{O}} = A_{\mathbf{O}} \otimes B_{\mathbf{O}}, \qquad (A \wp B)_{\mathbf{I}} = A_{\mathbf{I}} \otimes B_{\mathbf{I}};$$
$$(A \otimes B)_{\mathbf{I}} = A_{\mathbf{O}} \multimap B_{\mathbf{I}} \text{ if } \delta(A) = \mathbf{O}; \qquad (A \wp B)_{\mathbf{O}} = A_{\mathbf{I}} \multimap B_{\mathbf{O}} \text{ with } \textit{right switch};$$
$$(A \otimes B)_{\mathbf{I}} = B_{\mathbf{O}} \multimap A_{\mathbf{I}} \text{ if } \delta(B) = \mathbf{O}; \qquad (A \wp B)_{\mathbf{O}} = B_{\mathbf{I}} \multimap A_{\mathbf{O}} \text{ with } \textit{left switch}.$$

A *sequent* $\vdash \Gamma, \mathbf{C}$ with selected formula C is translated into $\mathbf{IMLL}^{\multimap\otimes}$ as $\Gamma_{\mathbf{I}} \vdash C_{\mathbf{O}}$. Since from the proof-net $\mathcal{R}$ we can recover a $\mathbf{MLL}^{\otimes\wp}$ sequent derivation, it is easy to recover an $\mathbf{IMLL}^{\multimap\otimes}$ derivation from it and from the translation of the formulas. Finally, if the given proof-net *contains cuts and the orientation is computationally consistent*, then one shows by induction on the cut-elimination procedure for proof-nets in $\mathbf{MLL}^{\otimes\wp}$ that each application of cut in a corresponding $\mathbf{IMLL}^{\multimap\otimes}$ derivation is correct, in the sense that both cut-formulas are of the same form. □

Remark 2.14. (i) F. Lamarche [24] independently presented a theory of proof nets for Intuitionistic Linear Logic (*Essential Nets*) carefully developing ideas related to part (ii) of the theorem.

(ii) The above translation is *non-functorial*, in the sense that the map for formulas (*objects*) essentially depends on the map for proofs (*morphisms*), as the assumption of *computational consistency* shows.

2.7. A functorial translation and Chu's construction.

The key idea for removing the constraint of computational completeness from Theorem 2.13 and thus obtaining *functorial translations* is to map $\mathbf{MLL}^{\otimes\wp}$ proof-nets to sequent derivations in multiplicative intuitionistic linear logic $\mathbf{IMLL}^{\otimes\wp\&}$ *with products*. We only give here the translations and state the result on Chu's construction.

Table 3. Functorial trip translation, the propositions.

$(P^{\perp})_{\mathbf{O}} = P_{\mathbf{I}}$ (P atomic)	$(P^{\perp})_{\mathbf{I}} = P_{\mathbf{O}}$;
$\mathbf{1}_{\mathbf{O}} = \mathbf{1}, \quad \mathbf{1}_{\mathbf{I}} = \top$	$\perp_{\mathbf{I}} = \mathbf{1} \quad \perp_{\mathbf{O}} = \top$;
$(A \otimes B)_{\mathbf{O}} = A_{\mathbf{O}} \otimes B_{\mathbf{O}}$	$(A \wp B)_{\mathbf{I}} = A_{\mathbf{I}} \otimes B_{\mathbf{I}}$;
$(A \otimes B)_{\mathbf{I}} = (A_{\mathbf{O}} \multimap B_{\mathbf{I}}) \& (B_{\mathbf{O}} \multimap A_{\mathbf{I}})$	$(A \wp B)_{\mathbf{O}} = (A_{\mathbf{I}} \multimap B_{\mathbf{O}}) \& (B_{\mathbf{I}} \multimap A_{\mathbf{O}})$

Theorem 2.15. ([4], section 3)
Let $\mathcal{A}$ be the free $*$-autonomous category on a set of objects $\{P, P', \ldots\}$ and let $\mathcal{C}$ be the symmetric monoidal closed category with products, free on the set of objects $\{P_{\mathbf{O}}, P_{\mathbf{I}}, P'_{\mathbf{O}}, P'_{\mathbf{I}}, \ldots\}$ (a pair $P_{\mathbf{O}}, P_{\mathbf{I}}$ in $\mathcal{C}$ for each P in $\mathcal{A}$).

We can give $\mathcal{C} \times \mathcal{C}^{op}$ the structure of a $*$-autonomous category thus:

$(X_{\mathbf{O}}, X_{\mathbf{I}}) \otimes (Y_{\mathbf{O}}, Y_{\mathbf{I}}) =_{df} (X_{\mathbf{O}} \otimes Y_{\mathbf{O}}, (X_{\mathbf{O}} \multimap Y_{\mathbf{I}}) \times (Y_{\mathbf{O}} \multimap X_{\mathbf{I}}))$
with unit $(\mathbf{1}, \top)$ and involution $(X_{\mathbf{O}}, X_{\mathbf{I}})^{\perp} = (X_{\mathbf{I}}, X_{\mathbf{O}})$

where $\mathbf{1}$ is the unit of $\otimes$ and $\top$ the terminal object of $\mathcal{C}$.

Therefore there is a functor F from $\mathcal{A}$ to $\mathcal{C} \times \mathcal{C}^{op}$ sending an object P to $(P_{\mathbf{O}}, P_{\mathbf{I}}$.

Table 4. Functorial trip translation, the proofs.

$$\vdash P^{\perp}, \mathbf{P} \quad \Rightarrow \quad P_{\mathbf{O}} \vdash P_{\mathbf{O}} \qquad\qquad \vdash \mathbf{P}^{\perp}, P \quad \Rightarrow \quad P_{\mathbf{I}} \vdash P_{\mathbf{I}}$$

$$\textit{cut}\ \frac{\vdash \Gamma, \mathbf{A} \quad \vdash A^{\perp}, \Delta, \mathbf{C}}{\vdash \Gamma, \Delta, \mathbf{C}} \quad \Rightarrow \quad \frac{\Gamma_{\mathbf{I}} \vdash A_{\mathbf{O}} \quad A^{\perp}_{\mathbf{I}}, \Delta_{\mathbf{I}} \vdash C_{\mathbf{O}}}{\Gamma_{\mathbf{I}}, \Delta_{\mathbf{I}} \vdash C_{\mathbf{O}}}\ \textit{cut}$$

$$\otimes_{\mathbf{O}}\ \frac{\vdash \Gamma, \mathbf{A} \quad \vdash \Delta, \mathbf{B}}{\vdash \Gamma, \Delta, \mathbf{A} \otimes \mathbf{B}} \quad \Rightarrow \quad \frac{\Gamma_{\mathbf{I}} \vdash A_{\mathbf{O}} \quad \Delta_{\mathbf{I}} \vdash B_{\mathbf{O}}}{\Gamma_{\mathbf{I}}, \Delta_{\mathbf{I}} \vdash A_{\mathbf{O}} \otimes B_{\mathbf{O}}} \otimes\text{-R}$$

$$\otimes_{\mathbf{I}}\ \frac{\vdash \Gamma, \mathbf{A} \quad \vdash B, \Delta, \mathbf{C}}{\vdash \Gamma, \Delta, A \otimes B, \mathbf{C}} \quad \Rightarrow \quad \frac{\dfrac{\Gamma_{\mathbf{I}} \vdash A_{\mathbf{O}} \quad B_{\mathcal{I}}, \Delta_{\mathbf{I}} \vdash C_{\mathbf{O}}}{\Gamma_{\mathbf{I}}, A_{\mathbf{O}} \multimap B_{\mathbf{I}}, \Delta_{\mathbf{I}} \vdash C_{\mathbf{O}}} \multimap\text{-L}}{\Gamma_{\mathbf{I}}, (A_{\mathbf{O}} \multimap B_{\mathbf{I}}) \& (B_{\mathbf{O}} \multimap A_{\mathbf{I}}), \Delta_{\mathbf{I}} \vdash C_{\mathbf{O}}}$$

$$\wp_{\mathbf{O}}\ \frac{\vdash \Gamma, A, \mathbf{B}}{\vdash \Gamma, \mathbf{A} \wp \mathbf{B}} \text{ and } \frac{\vdash \Gamma, \mathbf{A}, B}{\vdash \Gamma, \mathbf{A} \wp \mathbf{B}} \quad \Rightarrow \quad \frac{\dfrac{\Gamma_{\mathbf{I}}, A_{\mathbf{I}} \vdash B_{\mathbf{O}}}{\Gamma_{\mathbf{I}} \vdash A_{\mathbf{I}} \multimap B_{\mathbf{O}}} \quad \dfrac{\Gamma_{\mathbf{I}}, A_{\mathbf{O}} \vdash B_{\mathbf{I}}}{\Gamma_{\mathbf{I}} \vdash B_{\mathbf{I}} \multimap A_{\mathbf{O}}}}{\Gamma_{\mathbf{I}} \vdash (A_{\mathbf{I}} \multimap B_{\mathbf{O}}) \& (A_{\mathbf{I}} \multimap B_{\mathbf{O}})} \multimap\text{-R}$$

$$\wp_{\mathbf{I}}\ \frac{\vdash A, B, \Gamma, \mathbf{C}}{\vdash A \wp B, \Gamma, \mathbf{C}} \quad \Rightarrow \quad \frac{A_{\mathbf{I}}, B_{\mathbf{I}}, \Gamma_{\mathbf{I}} \vdash C_{\mathbf{O}}}{A_{\mathbf{I}} \otimes B_{\mathbf{I}}, \Gamma_{\mathbf{I}} \vdash C_{\mathbf{O}}} \otimes\text{-L}$$

$$\vdash \mathbf{1} \quad \Rightarrow \quad \vdash \mathbf{1} \qquad\qquad \perp_{\mathbf{O}}\ \frac{\vdash \Gamma}{\vdash \Gamma, \perp} \quad \Rightarrow \quad \Gamma_{\mathbf{I}} \vdash \top$$

$$\perp_{\mathbf{I}}\ \frac{\vdash \Gamma, \mathbf{A}}{\vdash \perp, \Gamma, \mathbf{A}} \quad \Rightarrow \quad \frac{\Gamma_{\mathbf{I}} \vdash A_{\mathbf{O}}}{\mathbf{1}, \Gamma_{\mathbf{I}} \vdash A_{\mathbf{O}}}\ \mathbf{1}\text{-L}$$

If $\pi : I \to \wp(\Gamma)$ is a morphism of $\mathcal{A}$ represented as a proof-net $\mathcal{R}$ with conclusions Γ, then the morphism $(\mathbf{1}, \top) \to \big(\wp(\Gamma)_{\mathbf{O}}, \wp(\Gamma)_{\mathbf{I}}\big)$ encodes all *Girard's trips (in a sense specified in [4]).*

Remark 2.16. Theorem 2.15 solves the problem of characterizing the *flow of information* within proofs in classical multiplicative linear logic. The solution appears as an abstract form of *"game semantics"*, where the *opponent/player* polarity is related to the *input/output* and *source/target* polarities of intuitionistic logic and functional programming. But does Theorem 2.15 also solve the problem of characterizing computationally consistent translations of multiplicative linear logic into the π-calculus that directly represent the logical flow of information through the I-O behaviour of processes? Notice that a translation of *classical multiplicative* proof nets $\mathcal{R}$ into *intuitionistic multiplicative and additive* linear logic is most naturally matched by a translation in the π-calculus *with finite summations and non-deterministic choice*. However in a process $P + Q$ any interaction involving a subterm of P requires discarding Q (and conversely), hence term rewriting ($\succ$) is not congruent with respect to $+$. Thus the straightforward correspondence between proof-nets and π-calculus terms *modulo structural congruence* and between cut-elimination and process interaction is fact lost: perhaps as expected, the π-calculus representation of

logical computation is modulo some notion of bisimulation.[5] The use of summation and non-deterministic choice in the π-calculus translation is related to the use on *boxes* in proof nets for *multiplicative and additive* (**MALL**) linear logic discussed in [10], Section 6. A technique of eliminating additive boxes by *slicing* is also considered there; we cannot pursue the issue here.

2.8. An anomaly in Milner's encoding of the typed λ-calculus?

Although Theorem 2.15 does clarify the problem of characterizing the computationally consistent translations of proof-nets in $\mathbf{MLL}^{\otimes\wp}$ into the π-calculus, from another point of view it seems that we have looked at the *mirror image* of our problem all along. Indeed (multiplicative) linear logic must be able to represent at least the (linear) λ calculus and we do have encoding by R. Milner whose *input-output* behaviour seems dual to the one given by our analysis.

Consider the encoding of the (linear) λ-calculus discussed in [10], Section 5.5:

$$[\![\lambda x M]\!]_u \ =_{df}\ u(xv)[\![M]\!]_v$$

$$[\![MN]\!]_u \ =_{df}\ (\nu v)\big([\![M]\!]_v \| (\nu x)\overline{v}\langle xu\rangle x(w)[\![N]\!]_w\big)$$
(where x is not free in N);

$$[\![x]\!]_u \ =_{df}\ \overline{x}\langle u\rangle$$

Thinking of this as a translation of the *typed* linear λ-calculus, we have that

- $\lambda x\colon A.M\colon A \multimap B$ is an *output*, corresponding to a consequence of a $\multimap$-introduction, yet in Milner's translation $[\![\lambda x M]\!]_u$ the channel u is a *receiver*;
- $[\![MN]\!]_u$ is an *input* in a cut-free (normal) derivation ending with a $\multimap$-elimination with $M\colon A \multimap B$ an input and $N\colon A$ an output; but in the standard representation *with cut* analysed in [10] M is an *output*. In Milner's translation the channel v in $[\![M]\!]v$ is a *receiver* but the prefix $\overline{v}\langle xu\rangle$ interacting with it is a *sender*. Even when $[\![N]\!]_w$ is an output, as expected from the logical orientation, the prefix $x(w)$ interacting with it is an *input*.

Thus it seems that Milner's translation somehow *reverses the logical orientation*. The explanation of this fact in [10], namely, that the roles of input and output are reversed in the *computational environment* where the processes operate, is a sensible remark, but it underplays the issue. Taking the input-output orientations of the π-calculus terms seriously one could conjecture that *in the π-calculus translation links have dual orientations to that of the terms they represent, as given in* Proposition 2.12.

Definition 2.17. (Dual orientation)
Let $s\mathcal{R}$ be a D-R graph on a proof net $\mathcal{R}$ and consider the Girard trip determined by the switching s, as in Definition 2.11. The *dual orientation δ determined by s* makes the assignment $\delta A = \mathbf{I}$ if the Girard trip on $s\mathcal{R}$ is covariant on A and $\delta A = \mathbf{O}$ otherwise. In particular, the *times* and *par* links are oriented thus:

[5] It may be relevant to notice that Abramsky had propose the following translation of axioms as *bi-directional buffers*:

$$I_{xy} =_{df}\ (x(a)\overline{y}\langle\rangle) + (y(a)\overline{x}\langle a\rangle) \mid x\colon A^{\perp}, y\colon A$$

Abramsky did not translate in a similar way *times* and *par* links, as one ought to in order to implement Chu's construction.

$$\frac{A_{\mathbf{I}} \quad B_{\mathbf{I}}}{(A \otimes B)_{\mathbf{I}}}\ \textit{times 1:} \qquad \frac{A_{\mathbf{I}} \quad B_{\mathbf{O}}}{(A \otimes B)_{\mathbf{O}}}\ \textit{times 2:} \qquad \frac{A_{\mathbf{O}} \quad B_{\mathbf{I}}}{(A \otimes B)_{\mathbf{O}}}\ \textit{times 3:}$$

$$\frac{A_{\mathbf{O}} \quad B_{\mathbf{O}}}{(A \wp B)_{\mathbf{O}}}\ \textit{par 1:} \qquad \frac{A_{\mathbf{O}} \quad B_{\mathbf{I}}}{(A \wp B)_{\mathbf{I}}}\ \textit{par 2:} \qquad \frac{A_{\mathbf{I}} \quad B_{\mathbf{O}}}{A \wp B_{\mathbf{O}}}\ \textit{par 3:}$$

Moreover we have a dual of Theorem 2.13 (Proposition 3.1 below). Thus our conjecture amounts to saying that *in a typed setting Milner's translation is essentially a dualizing operation and may be thought of as taking place in co-intuitionistic logic, rather than in intuitionistic logic*. The claim is hardly extravagant, if we think that CPS transforms in Thelecke's thesis are typed as *tensor-negation* types, which are dual to implication [33].

The point may be clarified by looking again at the representation of *application* in [10], Section 5.5: using the orthogonal negation $(\ )^{\perp}$ of linear logic to express the duality, we would like to type the λ-terms M and N in the π-calculus translation as $[\![M]\!]v\colon (B^{\perp} \diagdown A^{\perp} \vdash)$ and $[\![N]\!]w\colon (A^{\perp} \vdash)$. Thus we have

$$\frac{\dfrac{\vdash I_{uu'} \mid u : \mathbf{B}, u' : B^{\perp} \qquad \vdash [\![N]\!]_w \mid w : \mathbf{A}}{\vdash \mathbf{Times}_v^{u'w}(I_{uu'}, [\![N]\!]_w)_u \| \ u : \mathbf{B}, v : A \otimes B^{\perp}} \qquad \vdash [\![M]\!]_v \mid v : (\mathbf{A}^{\perp} \wp \mathbf{B})}{\vdash \mathbf{Cut}^v\big(\mathbf{Times}_v^{u'w}(I_{uu'}, [\![N]\!]_w)_u, [\![M]\!]_v\big) \mid u : \mathbf{B}}$$

where in representing λ-calculus terms a switching must choose the formulas in boldface and also B in $A^{\perp} \wp B$. Applying the dual orientation, by *par 2* we have $(A^{\perp} \wp B)_{\mathbf{I}} = B_{\mathbf{I}} \diagdown A^{\perp}_{\mathbf{O}} = B^{\perp} \diagdown A^{\perp}$ on the left, and dually by *times 2* $(A \otimes B^{\perp})_{\mathbf{O}} = B^{\perp}_{\mathbf{O}} \diagdown A_{\mathbf{I}} = B^{\perp} \diagdown A^{\perp}$ on the right. Please notice that the variable $v : (A^{\perp} \wp B)_{\mathbf{I}}$ is a *receiver* and that $v : (A \otimes B^{\perp})_{\mathbf{O}}$ is a *sender*, as required. Thus we have the following typing:

$$\frac{\dfrac{u : B^{\perp} \vdash I_{uu'} \mid u' : B^{\perp} \qquad w : A^{\perp} \vdash [\![N]\!]_w \mid}{u : B^{\perp} \vdash \mathbf{Times}_v^{u'w}(I_{uu'}, [\![N]\!]_w)_u \| \ v : B^{\perp} \diagdown A^{\perp}} \qquad v : B^{\perp} \diagdown A^{\perp} \vdash [\![M]\!]_v \mid}{u : B^{\perp} \vdash \mathbf{Cut}^v\big(\mathbf{Times}_v^{u'w}(I_{uu'}, [\![N]\!]_w)_u, [\![M]\!]_v\big) \mid}$$

Such a possible solution of a supposed "anomaly" in Milner's translation is a motivation for the dual linear calculus considered in Part II of this paper.

3. PART II. Multiplicative co-Intuitionistic linear Logic

We consider the fragment of *co-intuitionistic linear logic* **co-IMLL**$^{\diagdown\wp}$ with only the connectives *subtraction* ($\diagdown$) and *par* ($\wp$), the *dual* of Multiplicative Intuitionistic Linear Logic $\mathbf{IMLL}^{\multimap\otimes}$ with *linear implication* ($\multimap$) and *tensor product* ($\otimes$).

3.1. Syntax and meaning of co-IMLL$^{\diagdown\wp}$.

Given a countable sequence of elementary formulas denoted by $\eta_1, \eta_2, \ldots$ the language of **co-IMLL**$^{\diagdown\wp}$ is given by the following grammar:

$$A, B \ := \ \eta \mid A \diagdown B \mid A \wp B$$

Here $\eta_i = \not{\pi} p_i$ expresses the hypothesis that proposition p_i is true. If A and B are hypothetical expressions, then the intended meaning of $A \smallsetminus B$ is *possibly A and not B*; $A \wp B$ is Girard's *multiplicative parallel disjunction* (*A par B*). For co-intutionistic logic as a logic of hypotheses, see [6].

3.1.1. Informal explanation.

As *co-intuitionistic linear logic* may be quite unfamiliar, we sketch an intuitive explanation of its proof theory. We think of co-intuitionistic logic as being about *making hypotheses* [8, 6]. It has a consequence relation of the form

$$H \vdash H_1, \ldots, H_n. \tag{7}$$

Suppose H is a hypothesis: *which (disjunctive sequence of) hypotheses H_1* or ... or *H_n follow from H?* Since the logic is *linear*, commas in the meta-theory stand for Girard's *par* and the structural rules Weakening and Contraction are not allowed.

The main connectives are *subtraction* $A \smallsetminus B$ (*possibly A and not B*) and Girard's *par* $A \wp B$. Sequent-style Natural Deduction inference rules for *subtraction* are as follows.

$$\smallsetminus\textit{-intro}\ \frac{H \vdash \Gamma, C \qquad D \vdash \Delta}{H \vdash \Gamma, C \smallsetminus D, \Delta} \qquad \smallsetminus\textit{-elim}\ \frac{H \vdash \Delta, C \smallsetminus D \qquad C \vdash D, \Upsilon}{H \vdash \Delta, \Upsilon}$$

Notice that in the $\smallsetminus$*-elimination* rule the evidence that D may be derivable from C given by the *right premise* has become *inconsistent with the hypothesis* $C \smallsetminus D$ in the left premise; in the conclusion we drop D and we *set aside* the evidence for the inconsistent alternative. Namely, such evidence is not destroyed, but rather stored somewhere for future use.

If the *left premise* of $\smallsetminus$*-elimination*, deriving $C \smallsetminus D$ or Δ from H, has been obtained by a $\smallsetminus$*-introduction*, this inference has the form

$$\frac{H \vdash \Delta_1, C \qquad D \vdash \Delta_2}{H \vdash \Delta_1, \Delta_2, C \smallsetminus D}.$$

Then the pair of *introduction/elimination* rules can be eliminated: using the *removed evidence* that D with Υ are derivable from C (*right premise* of the $\smallsetminus$*-elim.*) we can conclude that $\Delta_1, \Delta_2, \Upsilon$ are derivable from H. This is, in a nutshell, the principle of normalization (or *cut-elimination*) for subtraction.

3.1.2. Sequent calculus for co-IMLL$^{\smallsetminus \wp}$

In Table 5 we give he rules of the sequent calculus for **co-IMLL**$^{\smallsetminus \wp}$ which we call **co-MLJ**$^{\smallsetminus \wp}$. To the *subtractive* fragment there is a Prawitz style Natural Deduction system, to be called **co-MNJ**$^{\smallsetminus}$, which is dual to the familiar *linear implicational* Natural Deduction $\mathbf{MNJ}^{\multimap}$. Using the *"dual orientations"* of definition 2.17, section 2.8, we can state the following dual of theorem 2.13.

Proposition 3.1. (i) Any computationally consistent *dual orientation* given by a Girard trip on a proof-net with conclusions $\Gamma, \mathbf{C}$ in $\mathbf{MLL}^{\otimes \wp}$ corresponds to (a class of) sequent calculus derivations of $C_{\mathbf{I}} \vdash \Gamma_{\mathbf{O}}$ in **co-IMLL**$^{\multimap \otimes}$.

(ii) Conversely, every sequent derivation in **co-IMLL**$^{\multimap \otimes}$ of $C_{\mathbf{I}} \vdash \Gamma_{\mathbf{O}}$ may be associated with a computationally consistent *dual orientation* given by Girard trip on a proof-net with conclusions $\Gamma, \mathbf{C}$ in $\mathbf{MLL}^{\otimes \wp}$.

$$\textit{axiom}\quad A \vdash A \qquad\qquad \textit{cut}\quad \frac{E \vdash \Gamma, A \qquad A \vdash \Delta}{E \vdash \Gamma, \Delta}$$

$$\textit{exchange}\quad \frac{E \vdash \Gamma, B, A, \Delta}{E \vdash \Gamma, A, B, \Delta}$$

$$\diagdown\text{-R}\quad \frac{E \vdash \Gamma, C \qquad D \vdash \Delta}{E \vdash \Gamma, C \diagdown D, \Delta} \qquad\qquad \diagdown\text{-L}\quad \frac{C \vdash D, \Delta}{C \diagdown D \vdash \Delta}$$

$$\wp\text{-R}\quad \frac{E \vdash \Gamma, C_0, C_1}{E \vdash \Gamma, C_0 \wp C_1} \qquad\qquad \wp\text{-L}\quad \frac{C_0 \vdash \Gamma_0 \qquad C_1 \vdash \Gamma_1}{C_0 \wp C_1 \vdash \Gamma_0, \Gamma_1}$$

Table 5. Sequent Calculus $\mathbf{MLJ}^{\diagdown\wp}$

3.2. Example.

Consider the Petri net $\mathbf{N}$ in Figure 1 and the computation resulting from the given initial marking, namely, using resources A and B to fire the first transition and obtain C and then using a resource D to fire the second transition and obtain E.

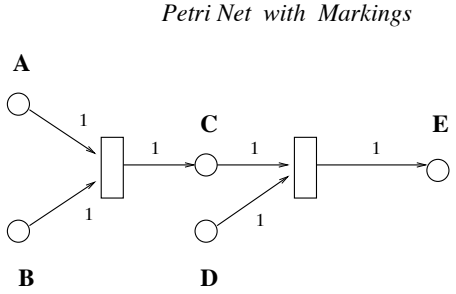


Figure 1. A Petri Net.

Such computation can be represented in Intuitionistic Multiplicative Linear Logic **IMLL** by a (cut free) sequent derivation of

$$\underbrace{(A, A, B, D, D}_{M_1}, \underbrace{A \multimap (B \multimap C)}_{T_1}, \underbrace{C \multimap (D \multimap E)}_{T_2} \vdash \underbrace{E \otimes (A \otimes D)}_{M_2} \tag{8}$$

as shown, e.g., in [26]. There is only one such a derivation in *commutative* **IMLL** (*modulo Exchange*).

By applying Theorem 2.13, (ii) and using the resulting orientation to define the π-calculus translation, we have:

$$\begin{array}{rl}
\vdash V_{vy_2x_2aa_0bdd_0} \mid a : A^{\perp}_{\mathbf{I}}, a_0 : A^{\perp}_{\mathbf{I}}, b : B^{\perp}_{\mathbf{I}}, & d : D^{\perp}_{\mathbf{I}}, d_0 : D^{\perp}_{\mathbf{I}}, y_2 : A_{\mathbf{O}} \otimes (B_{\mathbf{O}} \otimes C^{\perp}_{\mathbf{I}}), \\
x_2 : C_{\mathbf{O}} \otimes (D_{\mathbf{O}} \otimes E^{\perp}_{\mathbf{I}}), & v : E_{\mathbf{O}} \otimes (A_{\mathbf{O}} \otimes D_{\mathbf{O}}) \\
\text{where } V_{vy_2x_2aa_0bdd_0} = & \mathbf{Times}^{e'z}_{v}(S_{y_2x_2abde'}, U_{za_0d_0}) \\
S_{y_2x_2abde'} = \mathbf{Times}^{a'y_1}_{y_2}(I_{aa'}, R_{y_1x_2bde'}) & U_{za_0d_0} = \mathbf{Times}^{a'_0d'_0}_{z}(I_{a_0a'_0}, I_{d_0d'_0}) \\
R_{y_1x_2bde'} = \mathbf{Times}^{b'c}_{y_1}(I_{bb'}, Q_{x_2cde'}) & Q_{x_2cde'} = \mathbf{Times}^{c'x_1}_{x_2}(I_{cc'}, P_{x_1de'}) \\
P_{x_1de'} = & \mathbf{Times}^{d'e}_{x_1}(I_{dd'}, I_{ee'})
\end{array}$$

The same computation can be encoded in an easy **CCS** computation:

$$a\|a\|b\|d\|d\|(\nu xy)\big(\overline{a}.x\|b.\overline{x}.y\|d.\overline{y}.e\big) \quad \succ \quad a\|d\|e \tag{9}$$

But here the use of *guarding prefixes* of **CCS** (or some encoding of them in the π-calculus) is essential, so there is no way to represent the cut-elimination process of **IMLL** in **CCS**: for this purpose, as pointed out above, *contextual rewriting* would be required *within guarded processes*.

If we decorate **IMLL** natural deduction or sequent derivations with the *linear λ-terms* with $\otimes$ and `let` constructs as, e.g., in [11], then we have a full and faithful representation of normalization or cut-elimination in **IMLL** and this allows us to represent *other computation strategies*. For instance, in our simple example we may load the nodes B, D and A in this order and still the computation would go through. Thus we have derivations $\mathcal{D}_{11}$, $\mathcal{D}_{12}$, $\mathcal{D}_{21}$ and $\mathcal{D}_{22}$ decorated with the following terms:

$$\begin{array}{c}
\mathcal{D}_{11} : \\
g : T_2, f : T_1 \vdash \underbrace{\lambda b \lambda d \lambda a . g(fab)d}_{u} : B \multimap (D \multimap (A \multimap E))
\end{array}$$

where $T_1 = A \multimap (B \multimap C)$ and $T_2 = C \multimap (D \multimap E)$ as above,

$$\begin{array}{c}
\mathcal{D}_{12} : \\
y_1 : A \otimes A, b : B, y_2 : D \otimes D, h : B \multimap (D \multimap (A \multimap E)) \vdash \ell : E \otimes (A \otimes D)
\end{array}$$

where $\ell = \texttt{let}\ y_2\ \texttt{be}\ d \otimes d'\ \texttt{in}\ \ell_1$ and $\ell_1 = \texttt{let}\ y_1\ \texttt{be}\ a \otimes a'\ \texttt{in}\ (hbda) \otimes (a' \otimes d')$;

$$\begin{array}{cc}
\mathcal{D}_{21} : & \mathcal{D}_{22} : \\
a : A, a' : A \vdash a \otimes a' : A \otimes A & d : D, d' : D \vdash d \otimes d' : D \otimes D
\end{array}$$

and we obtain a derivation $\mathcal{D}_0$ *with cut* as follows:

$$\dfrac{\mathcal{D}_{21} \qquad \dfrac{\mathcal{D}_{22} \qquad \dfrac{\mathcal{D}_{11} \qquad \mathcal{D}_{12}}{y_1 : A \otimes A, b : B, y_2 : D \otimes D, g : T_2, f : T_1 \vdash \ell[u/h] : E \otimes (A \otimes D)}\, cut_1}{d : D, d' : D, y_1 : A \otimes A, b : B, g : T_2, f : T_1 \vdash \ell[u/h][d \otimes d/y_2] : E \otimes (A \otimes D)}\, cut_2}{d : D, d' : D, a : A, a' : A, b : B, g : T_2, f : T_1 \vdash \ell[u/h][d \otimes d'/y_2][a \otimes a'/y_1] : E \otimes (A \otimes D)}\, cut_3$$

As expected, by eliminating cuts we get

$$d : D, d' : D, b : B, a : A, a' : A, g : T_2, f : T_1 \vdash g(fab)d : E$$

3.2.1. The example in co-Intuitionistic Multiplicative Linear Logic

Here we consider a dual representation of the same Petri Net computation as derivations of the sequent R :

$$\underbrace{E \wp (A \wp D)}_{M_2^\perp} \vdash \underbrace{(E \smallsetminus D) \smallsetminus C}_{T_2^\perp}, \underbrace{(C \smallsetminus B) \smallsetminus A}_{T_1^\perp}, \underbrace{A, A, B, D, D}_{M_1^\perp}$$

The following is a cut free derivation $\mathcal{D}$ of R in the sequent calculus for **co-IMLL**$^{\smallsetminus \wp}$:

$$\dfrac{\dfrac{\dfrac{E \vdash E \qquad D \vdash D}{E \vdash E \smallsetminus D, D}\smallsetminus\text{-R} \qquad \dfrac{\dfrac{C \vdash C \qquad B \vdash B}{C \vdash C \smallsetminus B, B}\smallsetminus\text{-R} \qquad A \vdash A}{C \vdash (C \smallsetminus B) \smallsetminus A, A, B}\smallsetminus\text{-R}}{E \vdash (E \smallsetminus D) \smallsetminus C, T_1, A, B, D}\smallsetminus\text{-R} \qquad \dfrac{A \vdash A \qquad D \vdash D}{A \wp D \vdash A, D}\wp\text{-L}}{E \wp (A \wp D) \vdash T_2^\perp, T_1^\perp, A, A, B, D, D}\smallsetminus\text{-R}$$

If we apply the analogue of Theorem 2.13, (ii) using the *"dual translation"* sketched in section 2.8 we obtain the following π-calculus translation:

$$\begin{array}{rl} \vdash V_{vy_2x_2a'a'_0b'd'd'_0} \mid v : E^\perp_{\mathbf{I}} \otimes (A^\perp_{\mathbf{I}} \otimes D^\perp_{\mathbf{I}}) & x_2 : (E_{\mathbf{O}} \otimes D^\perp_{\mathbf{I}}) \otimes C^\perp_{\mathbf{I}}), \\ y_2 : (C_{\mathbf{O}} \otimes B^\perp_{\mathbf{I}}) \otimes A^\perp_{\mathbf{I}} & a' : A_{\mathbf{O}}, a'_0 : A_{\mathbf{O}}, b' : B_{\mathbf{O}}, d' : D_{\mathbf{O}}, d'_0 : D_{\mathbf{O}} \end{array}$$

$$\begin{array}{rl} \text{where } V_{vy_2x_2a'a'_0b'd'd'_0} = & \mathbf{Times}^{ez}_{v}(S_{y_2x_2a'b'd'e}, U_{za'_0d'_0}) \\ S_{y_2x_2a'b'd'e} = \mathbf{Times}^{y_1a}_{y_2}(R_{y_1x_2b'd'e}, I_{aa'}) & U_{za'_0d'_0} = \mathbf{Times}^{a_0d_0}_{z}(I_{a_0a'_0}, I_{d_0d'_0}) \\ R_{y_1x_2b'd'e} = \mathbf{Times}^{c'b}_{y_1}(Q_{x_2c'd'e}, I_{bb'}) & Q_{x_2c'd'e} = \mathbf{Times}^{x_1c}_{x_2}(I_{cc'}, P_{x_1d'e}) \\ P_{x_1d'e} = & \mathbf{Times}^{e'd}_{x_1}(I_{ee'}, I_{dd'}) \end{array}$$

4. Dual calculus and term assignment for co-IMLL$^{\smallsetminus \wp}$

The calculus of sequents for **IMLL**$^{-\circ\otimes}$ has sequents

$$x_1 : A_1, \ldots x_n : A_n \vdash t : A$$

decorated with linear λ terms with tensor product (Table 2). For the dual logic **co-IMLL**$^{\smallsetminus \wp}$. we propose a *linear calculus of coroutines* assigned to sequents

$$x : C \vdash t_1 : C_1, \ldots, t_n : C_n$$

of **co-MLJ**$^{\smallsetminus \wp}$ (and to the natural deduction derivations of **co-MNJ**$^{\smallsetminus \wp}$).

4.1. From Crolard's classical coroutines to co-intuitionistic ones

Crolard [15] provides a term assignment to the subtraction rules in the framework of Parigot's $\lambda\mu$-calculus, typed in a sequent-style natural deduction system. The $\lambda\mu$-calculus provides a typing system for functional programs with *continuations* and a computational interpretation of classical logic.

In the type system for the $\lambda\mu$ calculus sequents may be written in the form $\Gamma \vdash t : A \mid \Delta$, with contexts $\Gamma = x_1 : C_1, \ldots, x_m : C_m$ and $\Delta = \alpha_1 : D_1, \ldots, \alpha_n : D_n$, where the x_i are *variables* and the α_j are

μ-variables (or *co-names*). In addition to the rules of the simply typed lambda calculus, there are *naming rules*

$$\frac{\Gamma \vdash t : A \mid \alpha : A, \Delta}{\Gamma \vdash [\alpha]t : \bot \mid \alpha : A, \Delta}[\alpha] \qquad \frac{\Gamma \vdash t : \bot \mid \alpha : A, \Delta}{\Gamma \vdash \mu\alpha.t : A \mid \Delta}\mu$$

whose effect is to "change the goal" of a derivation and which allow us to represent the familiar *double negation rule* in Prawitz Natural Deduction.

Crolard extends the $\lambda\mu$ calculus with introduction and elimination rules for subtraction:[6]

$$\frac{\Gamma \vdash t : A \mid \Delta}{\Gamma \vdash \texttt{make-coroutine}(t, \beta) : A \setminus B \mid \beta : B, \Delta} \setminus I$$

$$\frac{\Gamma \vdash t : A \setminus B \mid \Delta \qquad \Gamma,\ x : A \vdash u : B \mid \Delta}{\Gamma \vdash \texttt{resume}\ t\ \texttt{with}\ x \mapsto u : C \mid \Delta} \setminus E$$

The reduction of a redex of the form

$$\frac{\dfrac{\Gamma \vdash t : A \mid \Delta}{\Gamma \vdash \texttt{make-coroutine}(t, \beta) : A \setminus B \mid \beta : B, \Delta} \setminus\text{-I} \qquad \Gamma, x : A \vdash u : B \mid \Delta}{\Gamma \vdash \texttt{resume}\ (\texttt{make-coroutine}(t, \beta))\ \texttt{with}\ x \mapsto u\ : C \mid \beta : B, \Delta} \setminus\text{-E}$$

is as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma, x : A \vdash u\ B \mid \Delta}{\Gamma \vdash u[t/x] : B \mid \Delta}\ \textit{substitution}}{\Gamma \vdash [\beta]u[t/x] : \bot \mid \beta : B, \gamma : C, \Delta'}[\beta]}{\Gamma \vdash \mu\gamma.[\beta]u[t/x] : C \mid \beta : B, \Delta'}\mu$$

Working with the full power of classical logic, if a constructive system of bi-intuitionistic logic is required, then the implication right and subtraction left rules must be restricted; this can be done by considering *relevant dependencies*.[7] Crolard is able to show that the term assignment for such a restricted logic is a calculus of *safe coroutines*, described as terms in which no coroutine can access the local environment of another coroutine.

Crolard's work suggests the possibility of defining co-intuitionistic coroutines directly, independently of the typing system of the $\lambda\mu$-calculus. Since μ-variable abstraction and the μ-rule are devices to change the "actual thread" of computation, the effect of removing such rules is that all "threads" of computation are simultaneously represented in a single-premise multiple-conclusion sequent, but variables y that are temporarily inaccessible in a term N are being replaced by a term $\mathrm{y}(M)$ by the substitution $N[y := \mathrm{y}(M)]$, where M contains a free variable x which is accessible in the current context. This approach was pursued in [6, 5, 8].

[6] Actually in Crolard [15] the introduction rule is given in the more general form of $\setminus$-*introduction* with two sequent premises as in our calculus and more general *continuation contexts* occur in place of β; the above formulation is logically equivalent and suffices for our purpose.

[7] For instance, in the derivation of the right premise $\Gamma, x : A \vdash u : B \mid \Delta$ of a subtraction elimination ($\setminus E$), there should be no relevant dependency between the formula B and the assumptions in Γ, but only between B and A.

4.2. A dual linear calculus for co-IMLL$^{\diagdown\wp}$

We present the grammar and the basic definitions of our dual linear calculus for the fragment of linear co-intuitionistic logic with subtraction and disjunction.

Definition 4.1. We are given a countable set of *free variables* (denoted by x, y, z, ...), and a countable set of *unary functions* (denoted by $\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots$).

(i) Terms are defined by the following grammar:

$$t, u \ := x \mid \mathtt{x}(t) \mid t \wp u \mid \mathtt{casel}(t) \mid \mathtt{caser}(t) \mid \mathtt{mkc}(t, \mathtt{x})$$

(ii) Let t_1, t_2, ... be an enumeration in a given order of all the *terms* freely generated by the above grammar starting with a special symbol $*$ and no variables (a selected variable a would also do the job). Thus we have a fixed *bijection* $t_i \mapsto x_i$ between terms and free variables.

(iii) Moreover, if t is a term and u is a term such that y occurs in u, then $\mathtt{postp}(\mathtt{y} \mapsto u\{y := \mathtt{y}(t)\}, t)$ is a p-term.

We use the abbreviations $(t \to \mathtt{y})$ for $\mathtt{mkc}(t, \mathtt{y})$ and $\xleftarrow{\mathtt{z} \mapsto u} w$ for $\mathtt{postp}(\mathtt{z} \mapsto u, w)$.

Notice that a p-term cannot be a subterm of other terms.

Definition 4.2. (i) The *free variables* $FV(\ell)$ in a term are defined as follows:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\mathtt{x}(t)) &= FV(t) \\ FV(t \wp u) &= FV(t) \cup FV(u) \\ FV(\mathtt{casel}(t)) = FV(\mathtt{caser}(t)) &= FV(t) \\ FV(\mathtt{mkc}\ (t, \mathtt{x})) &= FV(t) \\ FV(\mathtt{postp}(\mathtt{x} \mapsto u, t)) &= FV(u) \cup FV(t). \end{aligned}$$

(ii) A *computational context* $\mathcal{S}_x$ is a set of terms and p-terms containing the free variable x and no other free variable. We may represent a computational context as a list κ of terms and p-terms.

Definition 4.3. Substitution of a term t for a free variable x in a in a list of terms κ is defined as follows:

$$\begin{aligned} x\{x := t\} = t, \quad y\{x := t\} &= y \text{ if } x \neq y; \\ \mathtt{y}(u)\{x := t\} = \mathtt{y}(u\{x := t\}); \quad (r \wp u)\{x := t\} &= (r\{x := t\}) \wp (u\{x := t\}); \\ \mathtt{casel}(r)\{x := t\} = \mathtt{casel}(r\{x := t\}), \quad \mathtt{caser}(r)\{x := t\} &= \mathtt{caser}(r\{x := t\}); \\ \mathtt{mkc}(r, \mathtt{y})\{x := t\} &= \mathtt{mkc}(r\{x := t\}, \mathtt{y}), \\ \mathtt{postp}(\mathtt{y} \mapsto (u), s)\{x := t\} &= \mathtt{postp}(\mathtt{y} \mapsto (u\{x := t\}), s\{x := t\}); \\ ()\{x := t\} = (), \quad (u \cdot \kappa)\{x := t\} &= t\{x := t\} \cdot \kappa\{x := t\}. \end{aligned}$$

Definition 4.4. β-reduction of a *redex* $\mathcal{R}ed$ in a *computational context* $\mathcal{S}_x$ is defined as follows.

(i) If $\mathcal{R}ed$ is a term u of the following form, then the reduction is local and consists of the rewriting $u \rightsquigarrow_\beta u'$ in $\mathcal{S}_x$ as follows:

$$\texttt{casel}\ (t \wp u) \rightsquigarrow_\beta\ t; \qquad \texttt{caser}\ (t \wp u) \rightsquigarrow_\beta\ u.$$

(ii) If $\mathcal{R}ed$ has the form $\overset{\texttt{z}\mapsto u}{\prec\!\!\!\cdot}\ (t \to \texttt{y})$, i.e., $\texttt{postp}(\texttt{z} \mapsto u, \texttt{mkc}(t, \texttt{y}))$, then $\mathcal{S}_x$ has the form

$$\mathcal{S}_x\ =\ \mathcal{R}ed, \quad \overline{\kappa}, \quad \overline{\zeta}_{\texttt{y}}, \quad \overline{\xi}_{\texttt{z}}$$

where $\texttt{z}((t \to \texttt{y}))$ occurs in $\overline{\xi}_{\texttt{z}}$, $\texttt{y}(t)$ occurs in $\overline{\zeta}_{\texttt{y}}$ but $\texttt{z}((t \to \texttt{y}))$ does not and, finally, neither $\texttt{y}(t)$ nor $\texttt{z}((t \to \texttt{y}))$ occur in $\overline{\kappa}$. Using our bijection between the set of terms and the free variables, we can write $y = \texttt{y}(t)$ and $z = \texttt{z}((t \to \texttt{y}))$; then a *reduction* of $\mathcal{R}ed$ transforms the computational context as follows:

$$\mathcal{S}_x \quad \rightsquigarrow \quad \overline{\kappa}, \quad \overline{\zeta}\{y := u\{z := t\}\}, \quad \overline{\xi}\{z := t\}. \tag{10}$$

Thus for $\overline{\zeta} = u_1, \ldots, u_k$ and $\overline{\xi} = r_1, \ldots, r_m$ we have:

$$\overline{\xi}\{z := t\} = r_1\{z := t\}, \ldots, r_m\{z := t\};$$
$$\overline{\zeta}\{y := u\{z := t\}\} = u_1\{y := u\{z := t\}\}, \ldots, u_k\{y := u\{z := t\}\}.$$

4.3. Term assignment to $\mathbf{MLJ}^{\wedge\wp}$

Definition 4.5. (Term assignment)
The assignment of terms of the dual linear calculus to sequent calculus derivation in $\mathbf{MLJ}^{\wedge\wp}$ is given in Table 6.

Remark 4.6. (Binding of variables)
In our calculus the free variable y occurring in a computational context $\mathcal{S}_y$ becomes bound when $\mathcal{S}_y$ is merged with another context $\mathcal{S}_x$ because of the introduction of a term $\texttt{mkc}(t, \texttt{y})$ with t belonging to $\mathcal{S}_x$. Similarly the free variable x occurring in $\mathcal{S}_x \cup \{u\}$ becomes bound when a term $\texttt{postpone}(\texttt{x} \mapsto u, y)$ is introduced, creating a new computational context $\mathcal{S}_y$. Thus the terms `make-coroutine` and `postpone` are *global binders*: their scope is not delimited, contrary to the binding of variables by the λ operator in the λ-calculus or by the (νx) in the π-calculus. Global effects such as "remote binding" and "remote substitutions" evoke features of distributed computation, e.g., in biological computing.

But variable binding requires an appropriate definition of α-equivalence, to ensure proper renaming of variable and substitutions avoiding "capture of free variables". We define α-equivalence by induction on the recursive definition of a computational context given by the term assignment in Table 6.

Definition 4.7. (α-equivalence)
Let $\mathcal{S}^*_x$ and $\mathcal{S}^*_{x'}$ be computational contexts. We say that $\mathcal{S}^*_x$ and $\mathcal{S}^*_{x'}$ are α-equivalent (written $\mathcal{S}^*_x \equiv \mathcal{S}^*_y$) if the following conditions are satisfied. We focus on the cases of `make-coroutine` and `postpone`, the others being familiar.

Table 6. Labelled sequent calculus **co-MLJ**$^{\diagdown\wp}$

Labelled Sequent Calculus co-MLJ$^{\diagdown\wp}$

identity rules

logical axiom:

$$x\colon C \vdash \pi \mid x\colon C$$

In what follows let $\mathcal{S}_x = \pi_0 \mid \overline{\ell}$ and $\mathcal{S}_y = \pi_1 \mid \overline{\kappa}$:

cut:

$$\frac{x\colon E \vdash \mathcal{S}_x\colon \Upsilon_0\ ,\ t\colon C \qquad y\colon C \Rightarrow \mathcal{S}_y\colon \Upsilon_1}{x\colon E \vdash \mathcal{S}_x\colon \Upsilon_0, \mathcal{S}_y\{y := t\}\colon \Upsilon_1}$$

logical rules

$\diagdown$ *right:*

$$\frac{x\colon E \vdash \mathcal{S}_x\colon \Upsilon_0,\ t\colon C \qquad y\colon D \Rightarrow \mathcal{S}_y\colon \Upsilon_1}{E \vdash \mathcal{S}_x\colon \Upsilon_0, \mathtt{mkc}(t, \mathtt{y})\colon C \diagdown D, \mathcal{S}_y\{y := \mathtt{y}(t)\}\colon \Upsilon_1}$$

$\diagdown$ *left*:

$$\frac{x\colon C \vdash \mathcal{S}_x\colon \Upsilon, u\colon D}{y\colon C \diagdown D \vdash \mathtt{postp}(\mathtt{x} \mapsto u, y)\{x := \mathtt{x}(y)\}\colon,\ \mathcal{S}_x\{x := \mathtt{x}(y)\}\colon \Upsilon}$$

$\wp$ *right*

$$\frac{x : C \vdash \pi_1 \mid \overline{\ell} : \Upsilon, t_0 : D_0, t_1 : D_1}{y : C \vdash \pi_1 \mid \overline{\ell} : \Upsilon, t_0 \wp t_1 : D_0 \wp D_1}$$

$\wp$ *left:*

$$\frac{x : D_0 \vdash \mathcal{S}_x : \Upsilon_0 \qquad\qquad y : D_1 \vdash \mathcal{S}_y : \Upsilon_1}{z\colon D_0 \wp D_1 \vdash \mathcal{S}_x\{x := \mathtt{casel}(z)\}\colon \Upsilon_0, \mathcal{S}_y\{y := \mathtt{caser}(z)\}\colon \Upsilon_1}$$

1. If $\mathcal{S}^*_x = \{x\}$, then $\mathcal{S}^*_x \equiv \mathcal{S}^*_{x'}$ iff $\mathcal{S}^*_{x'} = \{x'\}$ and $x = x'$;
2. if $\mathcal{S}^*_x = \mathcal{S}_x \cup \{\mathtt{mkc}(t, \mathtt{y})\} \cup \mathtt{S}_y\{y := \mathtt{y}(t)\}$, then $\mathcal{S}^*_x \equiv \mathcal{S}^*_{x'}$ iff $\mathcal{S}^*_{x'} = \mathcal{S}_{x'} \cup \{\mathtt{mkc}(t', \mathtt{y}')\} \cup \mathtt{S}_{y'}\{y := \mathtt{y}'(t')\}$ and $\mathcal{S}_x \cup \{t\} \equiv \mathcal{S}_{x'} \cup t'$ and, moreover, for all variables z except for a finite number $\mathtt{S}_y\{y := z\} \equiv \mathtt{S}_{y'}\{y' := z\}$;
3. if $\mathcal{S}^*_x = (\{\mathtt{postpone}(y \mapsto u, x)\} \cup \mathcal{S}_y)\{y := \mathtt{y}(x)\}$, then $\mathcal{S}^*_x \equiv \mathcal{S}^*_{x'}$ iff $\mathcal{S}^*_{x'} = (\{\mathtt{postpone}(y' \mapsto u', x')\} \cup \mathcal{S}_{y'})\{y' := \mathtt{y}'(x')\}$ and for all variables z except for a finite number $(\mathcal{S}_y \cup \{u\})\{y := z\} \equiv (\mathcal{S}_{y'} \cup \{u'\})\{y' := z\}$.

Using definition (4.7), it is possible to define capture avoiding substitutions in the standard way (cfr. [23]). We avoid an explicit use of α-equivalence through a bijection between terms and free variables, as in Definition 4.1 (ii): the terms given by a term assignment encode the structure of a derivation tree, thus the bijection will not yield undesirable identification of free variables.

4.4. Dualities

In the following proposition we use orthogonality $(\ )^{\perp}$ as a meta-theoretical symbol for the duality between formulas and derivationss in $\mathbf{MLJ}^{-\circ\otimes}$ and in **co**-$\mathbf{MLJ}^{\diagdown\wp}$:

Proposition 4.8. (i) Given a labelled sequent calculus derivation d in $\mathbf{MLJ}^{-\circ\otimes}$ of $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$, there is a sequent derivation $d^{\perp}$ in **co**-$\mathbf{MLJ}^{\diagdown\wp}$ of $x : A^{\perp} \vdash \pi, u_1 : A_1^{\perp}, \ldots, u_n : A_n^{\perp}$, for some sequence of terms $u_1, \ldots, u_n$ and a sequence of p-terms π of the dual linear calculus of co-routines, and conversely;

(ii) If a $\mathbf{MLJ}^{-\circ\otimes}$ derivation d reduces to d_0, then the **co**-$\mathbf{MLJ}^{\diagdown\wp}$ derivation $d^{\perp}$ reduces in one step to $d_0^{\perp}$, and conversely.

The proposition is proved by a straightforward induction on the length of the given derivation (part (i)) and on the length of the reduction sequence (part (ii)). It is understood that "a step" of reduction in the dual linear calculus must be seen as "macro" instruction for several steps of rewriting, which may nevertheless be seen as a unit.

4.5. Labelled Prawitz' trees

As trees in Prawitz style Natural Deduction $\mathbf{MNJ}^{-\circ}$ can be decorated with linear λ terms, so we can draw Prawitz trees for **co**-$\mathbf{MNJ}^{\diagdown}$ derivations and decorate them with terms of the dual linear calculus of coroutines, as in the example in Appendix (section 7.1). In the linear case the correspondence between the sequent calculus **co**-$\mathbf{MLJ}^{\diagdown}$ and natural deduction **co**-$\mathbf{MNJ}^{\diagdown}$ is straightforward.

5. PART III. Membrane computing: λP systems as executors

In this section we will show the parallel between our dual linear calculus and λP systems. These are a variant of P systems, a computational framework aimed at representing structured environments inspired by cell biology, first introduced by Păun in 1998 [29] and subsequently specialized into many different types. More specifically, P systems model initial data as sets of objects (considered with their respective multiplicity) and evolution rules, similar to chemical reactions, inside a hierarchical set of membranes (containers); a computation is defined as a maximally parallel application of all the evolution rules, until no further transformation is possible. λP systems extend this model by defining additional basilar operations on membranes and allowing some membrane transfers inside the hierarchy.

5.1. Definitions

First, we will give here the basic definitions needed to understand the structure and workings of a λP system, referring the reader to [30] and [14] for further clarifications.[8]

[8]Some particulars of the definitions given here, such as the definition of rule in a membrane, have been simplified for ease of comprehension.

Definition 5.1. (Multiset)
A multiset M is a set of items $a, b, c \ldots$ counted with their respective multiplicities $m_0, m_1, m_2, \ldots$. Its conventional representation is a listing comprising every item immediately followed by its multiplicity:

$$M = a^{m_0} b^{m_1} c^{m_2} \ldots$$

Definition 5.2. (Membrane)
A membrane is a unit containing:

- a multiset of objects initially present immediately inside the membrane;
- other membranes inside, if any;
- a set of rules in the form $A_1^{a_1} A_2^{a_2} \ldots A_m^{a_m} \to (B_1^{b_1}, t_1)(B_2^{b_2}, t_2) \ldots (B_n^{b_n}, t_n)$, where each t_i must be one of in, out or here; they express transformations that, for every i in $[1; m]$, take away a_i instances of A_i from the membrane and, for every j in $[1; n]$, insert b_i instances of B_i in a non-deterministically chosen membrane inside the current one (if t_i is in), outside the current membrane (if t_i is out) or in the current membrane (if t_i is here). If t_i is here, the ordered pair $(B_i^{b_i}, t_i)$ can be written in the short form $B_i^{b_i}$.

Definition 5.3. (Membrane structure)
A membrane structure is a hierarchically arranged set of membranes, each denoted by a label, contained in an external membrane. The space delimited by a membrane and any membranes contained into it (if any) is called *region*.

In the rest of this paper, we will denote a membrane with label α with $[^{\alpha}]$. A membrane α is said to be a *child* of β if α is an inner membrane of β; if α is a child of β and there is no inner membrane of β containing α, α is called an *immediate child* of β. If we consider a membrane α and one of its immediate children β, we may denote the label of β with $\textcircled{\beta}$ (the circle intuitively means that β is visible for an hypothetical observer placed inside α).

Definition 5.4. (Basic operations)
We define three basic operations (*immediate inclusion*, *innermost inclusion* and *surrounding*) as follows.

1. The immediate inclusion $\textcircled{\alpha} \to [^{\alpha}]_{\mathrm{in}[^{\beta}]}$ takes a membrane α and all its inner structure and includes it in a membrane β at the same level, such that α becomes an immediate child of β;
2. the innermost inclusion $\textcircled{\alpha} \to [^{\alpha}]_{\mathrm{in}^*[^{\beta}]}$ takes a membrane α and all its inner structure and includes it in the innermost membrane of β;
3. the surrounding operation $\textcircled{\alpha} \to [^{\beta}[^{\alpha}]]$ surrounds α with a membrane β.

Additionally, we define their respective inverses as follows:

1. the inverse of the immediate inclusion is $\textcircled{\alpha} \to [^{\alpha}]_{\mathrm{out}[^{\beta}]}$ (it takes the membrane α, which must be an immediate child of β, and moves it outside β, so that α and β are at the same level);
2. the inverse of the innermost inclusion is $\textcircled{\alpha} \to [^{\alpha}]_{\mathrm{out}^*[^{\beta}]}$ (this is the same operation described above, with the only difference that α needs to be just a child of β);

3. the inverse of the surrounding operation applied to a membrane α is $\delta_{\text{in}[^{\beta}]}$, where with δ, following the usual notation for P systems, we refer to the dissolution of the β membrane itself.

These are, generally speaking, the fundamental operations performed in λP systems; while building the λP system that will act as an executor of our dual linear calculus, we shall use the immediate inclusion, the surrounding operation and its inverse extensively.

We are now ready to introduce λP rules, evolution rules that expand the general notion of rule in P systems by letting certain membranes taking part in a reaction and allowing direct transfers to the innermost or outermost membranes (with regards to the one to which the rule is applied).

Definition 5.5. (λP rule)
Let $\mathcal{O}$ denote the set of objects that can appear in the system. A λP rule is a rule of the form $X_1, X_2, \ldots, X_n \to Y^1_{\text{tar}_{Z_1}}, Y^2_{\text{tar}_{Z_2}}, \ldots, Y^m_{\text{tar}_{Z_m}}$, where:

- $X_i \in \mathcal{O}$ or $X_i = \textcircled{Z}$ ($Z \in \Lambda$, where Λ is the set of membrane labels as defined below);
- $Y^i \in \mathcal{O}$ or $Y^i = [^{Z}]$ or $Y^i = [^{Z}[^{U}]]$ ($Z, U \in \Lambda$);
- $\text{tar} \in \{\text{in}, \text{in}^*, \text{out}, \text{out}^*, \text{here}\}$, where with in^* and out^* we denote a transfer to the innermost or outermost membrane inside, or outside, the one to which the rule is applied.[9]

Definition 5.6. (λP system)
A λP system is a construct

$$\Pi = (\Lambda \cup \mathcal{O}, \mu(V), R, f)$$

where:

- Λ is the set of membrane labels;
- $\mathcal{O}$ is the set of objects that can appear in the membranes (for our convenience, we will always assume that δ belongs to $\mathcal{O}$);
- $\mu(V)$ is the initial membrane structure (V is the set of membranes);
- R is a set of λP rules;
- f is a function $V \mapsto \Lambda$.

We will now introduce a *tree representation* of membrane structures in a λP system, as well as the definition of *initial objects* in a membrane. The first one is a commonly used, intuitive way to visualize the hierarchy and nesting of membranes; the second definition, normally employed in basic P systems, needs to be given for λP systems, as the notion of *initial configurations* ([14], 2.2) is not sufficiently expressive to represent the exact location of all the objects (only more general sets comprising *all* the elements in a subtree can be derived; we will need a finer degree of control while performing some translations later in this paper).

[9] In the paper by Colson *et al.* ([14]), the authors use stay instead of here, although with the same meaning; we have chosen the latter form, as it is consistent with Păun ([29]).

Definition 5.7. (Tree representation of a λP system — see e.g. [30], Section 6)
The membrane structure of a λP system Π can be represented by a tree μ as follows:

1. there is a bijection between the nodes in the tree and the membranes in the λP system;
2. every node in the tree is labelled with the label of the corresponding membrane;
3. a node N_2 is a child of N_1 if, and only if, the membrane corresponding to N_2 is inside the membrane corresponding to N_1.

Definition 5.8. (Initial objects)
For each membrane M in a λP system Π, we define the set of initial objects $\mathcal{O}_M$ as the multiset containing all the objects present in M before the computation starts, counted with their multiplicities.

As in standard P systems, to perform a computation all the evolution rules are applied, whenever possible, in parallel and in a non-deterministic order. A computation is successful if and only if it halts (no more evolution rules can be applied) and if the output region still exists.

5.2. Translating the dual linear calculus

We shall show that the dual linear calculus can be simulated by a λP system by giving a constructive way to build an initial system representing every initial set of terms and by modelling β-reductions so that they respect Definition 4.4, translating each rule in the labelled sequent calculus $\mathbf{MLJ}^{\diagdown\wp}$ to an operation in a λP system.

Definition 5.9. (Term translation)
Terms in the dual linear calculus can be translated to a membrane structure as follows.

- For each term of type x we build a membrane structure $[^{x}x]$ and add a new object x to the set of objects $\mathcal{O}$.
- For each term of type $\mathtt{x}(t)$ we build a membrane structure $[^{\mathtt{x}(t)}M]$, where M is the membrane structure for t.
- For each term of type $t \wp u$ we build a membrane structure $[^{t\wp u}M_tM_u]$, where M_t and M_u are the membrane structures for t and u.
- For each term of type $\mathtt{casel}(t)$ we build a membrane structure $[^{\mathtt{casel}(t)}M]$, where M is the membrane structure for t. The same rule applies to terms of type $\mathtt{caser}(t)$.
- For each term of type $\mathtt{mkc}(t, \mathtt{x})$ we build a membrane structure $[^{\mathtt{mkc}(t,\mathtt{x})}M_tM_{\mathtt{x}}]$, where M_t and $M_{\mathtt{x}}$ are the membrane structures for t and $\mathtt{x}$.
- For each term of type $\mathtt{postp}(\mathtt{x} \mapsto u, t)$ we build a membrane structure $[^{\mathtt{postp}(\mathtt{x}\mapsto u,t)}M_xM_uM_t]$, where M_x, M_u and M_t are the membrane structures for x, u and t.

In every built membrane structure, we treat the label as a unique identifier that keeps track of the original term type. This will allow us to identify the exact rules to insert to perform substitutions and β-reductions.

Note that these assignments preserve the sets of free variables defined in Definition 4.2 and that terms that are not simple variables are created by enclosing them.

Definition 5.10. (Simulation of substitutions)
Substitutions defined in Definition 4.3 can be modelled in λP systems with *immediate inclusions* and is defined by inspection of each case. If M is the membrane representing the term where we need to perform the substitution and t is the term corresponding to it:

- if t is of the type $()\{x := t\}$, nothing is to be done;
- if t is of the type $x\{x := t\}$, we add the rule $X \to T$, where X and T are the labels of the membranes representing the terms x and t;
- if t is of the type $y\{x := t\}$ and $x \neq y$, we add the same rules as in the previous case;
- in every other case, we just apply the procedure recursively to the membranes representing the correct subterm (the recursive step can not be applied to the submembrane representing the x term in a membrane modelling $\mathtt{mkc}(t, \mathtt{x})$ or in one modelling $\mathtt{postp}(\mathtt{x} \mapsto u, t)$, as capture of variables may occur).

Lemma 5.11. (Initial computational context model)
Let $\mathcal{S}_x$ be a computational context. Then $\mathcal{S}_x$ can be modelled by a λP system $\Pi_{\mathcal{S}_x}$.

Proof:
We first start with an "empty" system

$$\Pi = (\Lambda \cup \mathcal{O}, \mu(V), R, f)$$

where $\Lambda = \{S\}$, $\mathcal{O} = \{\delta\}$, $\mu(V)$ is a membrane structure consisting only of a skin membrane labelled S, $R = \emptyset$ and f is the function associating the skin membrane with its label. Consider then the list of terms and p-terms κ representing $\mathcal{S}_x$. For each term t in κ, build the membrane system corresponding to t (this process is guaranteed to terminate as the term construction is inductive), according to Definition 5.9, adding all labels and modifying f accordingly. In the end, put all the membrane systems obtained at the precedent step inside the skin membrane S. □

Definition 5.12. (Simulation of β-reductions)
β-reductions defined in Definition 4.4 can be modelled in a λP system as follows. Consider a computational context model $\Pi_{\mathcal{S}_x}$. For each membrane M in $\Pi_{\mathcal{S}_x}$:

- if M has a label Ⓜ of the type $\mathtt{casel}(t)$ and the membrane corresponding to t is one modelling a $t_1 ⅋ t_2$ term, let us call P the membrane corresponding to t, T_L the one corresponding to t_1 and T_R the one corresponding to t_2; put a λP rule $P \to \delta_{\mathrm{in}[M]}$ in the parent membrane of P and $T_R \to \delta$ in P;
- if M has a label of the type $\mathtt{caser}(t)$ and the membrane corresponding to t is one modelling a $t_1 ⅋ t_2$ term, follow the same rule as above, replacing T_R with T_L in the last λP rule;
- if M has a label of the type $\mathtt{postp}(\mathtt{x} \mapsto u, z)$ and Z is a membrane representing a term of the type $\mathtt{mkc}(t, \mathtt{y})$:
  1. apply the bijection described in definition 4.4;

2. perform the substitution $u\{z := t\}$ in the membrane corresponding to u (as described before);
3. for every child of the skin membrane S that contains a child membrane structure representing the term $\mathtt{y}(t)$ (let m be their total number), perform the substitution $y := u\{z := t\}$ (by copying the membrane representing the term $\mathtt{y}(t)$ inside the skin membrane and performing a recursive substitution, this can be easily achieved); we also record the fact that this step has been completed by putting a new "trigger" object (let us call it S_1) inside the parent membrane of M (let it be S_F) when the copy of the term inside the skin membrane has been performed;
4. for every child of the skin membrane S that contains a child membrane structure representing the term $\mathtt{z}((t \to \mathtt{y}))$ (let n be their total number), perform the substitution $y := u\{z := t\}$ with the same technique described as above, this time using a new object S_2 as the trigger;
5. add the reaction $S_1 S_2 AM \to A$, where A is an enumeration of the immediate child membranes of S_F (excluding M itself), so that the `postp` term itself is deleted only when both substitutions have been started.

We have thus proved that:

Theorem 5.13. (Existence of a λP system)
The dual linear calculus defined in Section 4.2 can be executed by a λP system.

The "low-level" execution of the dual linear calculus in a λP system we have just shown seems quite complicated at first, if not frightening. That happens because we needed to introduce a single membrane for every term, so as to perform substitutions and β-reductions correctly or, in other words, to represent and model the "language structure" of the dual calculus.

There is, however, another possibility to express these concepts in a more intuitive way. Recalling the tree representation of λP systems given in Definition 5.7, it becomes intuitive to associate the basic operations described in Definition 5.4 with subtree moves: for instance, the operation 1 moves the subtree corresponding to α and all its inner membranes so that the root of the subtree becomes a child of the node corresponding to β. Entire terms can be represented as subtrees and substitutions and β-reductions can be (mostly) modelled as fundamental operations (substitutions become immediate inclusions with subtree cloning, `casel`/`caser` operations become subtree prunings and inverse surroundings, while β-reductions of `postp`/`mkc` terms are combinations of these tree operations).

5.3. Going backwards: representing λP systems

A more interesting problem is checking whether the *inverse* of Theorem 5.13 holds, as it would be shown that our dual linear calculus can simulate a particular distributed computing model. This is indeed true, at least under the assumption that the maximum multiplicity of every object inside each membrane at any given time is upper bounded; however, we shall not perform (as we did in the previous case) a direct translation from an arbitrary λP system to its representation in the dual linear calculus, but give a method to transform it into a Petri net. The latter will be then encoded into a co-$\mathbf{IMLL}^{\wp}$ computation and finally into the calculus we gave.

Theorem 5.14. (Translation of λP systems to Petri nets)
The structure and computational behaviour of a λP system S, where the maximum multiplicity of every object inside each membrane at any given time is upper bounded, can be represented by a Petri net N.

The general idea of the proof is to build a Petri net using the technique envisioned by Klejin, Koutny and Rozenberg ([22]), extending it conveniently to represent the dynamic evolution of the membrane structure typical of λP systems. More specifically, we need to keep track of the location of each membrane, so that rules that specify a target t_i *relative* to the current membrane (that is, $\mathrm{in}/\mathrm{in}^*$ and $\mathrm{out}/\mathrm{out}^*$) will always execute correctly. Each place will thus represent an ordered triplet $(\textit{membrane}, \textit{object}, \textit{membrane position})$, where *membrane position* is a unique identifier that distinguishes representations of the same membrane put into different places in the membrane structure.

Proof:
The Petri net we will build will be in the form $N = (P, T, W, M_0)$ where, as usual:

- P is a finite set of places and T is a finite set of transitions,
- P and T are disjoint,
- $W\colon (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ is a multiset representing the arcs, and
- M_0 is a multiset of places (the *initial marking* of the net).

Let us call the original λP system we need to translate S. Apply the following algorithm to obtain the required Petri net.

1. *(Initial membrane structure)* For each possible $(\textit{membrane}, \textit{object})$ couple initially present in S (by "initially present" we mean that the membrane exists and at least one occurrence of the specified object is present inside that membrane), generate a new place in the Petri net, labelling it with the original membrane label, the name of the object and a unique identifier that serves the sole purpose of discriminating between different locations inside S of the same membrane. We can, for example, recalling the tree representation of λP systems given in Definition 5.7, consider the path $\langle M_1, M_2, \ldots, M_n\rangle$ going from the root to the desired membrane M_n (by design of the λP system, that path exists and is unique) and, if L_i is the label of M_i, use $L_1 \| L_2 \| \ldots \| L_n$ as the identifier (where $\|$ denotes string concatenation).
2. *(Encoding of λP rules and basic operations implementation)* For each λP rule specified in S, let M be the membrane associated to the rule. Then, for each possible position of M in the system, we perform the translation as follows.
   (a) We insert a transition T into the Petri net.
   (b) For each object O on the left hand side of the rule (excluding δ), we draw an arc from the place representing the triplet $(M, O, \textit{current position of } M)$ to T; its weight is equal to the multiplicity of O in the rule.[10]
   (c) For each object O on the right hand side of the rule (excluding δ), we draw an arc from T to the place representing the triplet $(M', O, \textit{current position of } M')$. M' is determined as follows: if the target for O is out or out^*, M' is the outer (respectively, outermost) membrane of M (note that it can be determined with certainty, thanks to the fact that we have kept track

[10] We deliberately ignore "membrane labels" like (A) here, as they are merely a reference to the labels written in the basic operations on the right hand side.

of the dynamic changes of the system); if the target is here, $M = M'$, trivially; if the target is in or in^*, to preserve the indeterminacy with which the inner membrane is chosen, we draw *multiple arcs* to each possible inner (or innermost) membrane. The weight, as before, is equal to the multiplicity of O on the right hand side.

If during this phase a needed place does not exist, it suffices to create it, paying attention to add *all possible transitions that could move objects (or membranes) inside the new one*.

The considerations about the target and the creation of new places if needed are the same throughout the proof and will be implied for brevity in the next cases.

(d) For each $[^A]$ on the right hand side, what we need to do is consider the entire membrane and transfer it if needed.[11] We don't know, in general, the number of objects present inside the membrane (and inside all its children) at the time the transfer is performed; however, using the hypothesis that the multiplicity of any object is always upper bounded, we can simply generate all the possible combinations of multiplicities, add arcs going from the places representing possible objects inside A or its children (whether immediate or not) with weights equal to the specific multiplicity in each combination, create as many copies T_i of T (as well as the arcs connected to T) as the number of combinations and add arcs going from each T_i to the places representing the same objects inside the same membranes, but at the appropriate positions (depending on the target of the λP rule, as we specified in point 2c).

(e) For each $[^B[^A]]$ on the right hand side, we simply create (if needed) new places representing objects into the new membrane B, at the appropriate positions (depending on the target), then transfer $[^A]$ inside B, as in the previous point.

(f) For each δ object (membrane dissolution), we consider the membrane M' specified by the target and transfer the objects inside M' (as well as any inner membrane structure) to its parent.

Note that points 2e and 2f can be seen as special cases of point 2d.

3. *(Initial marking)* The initial marking M_0 of the Petri net is built as follows: for each object X in the $\mathcal{O}_M$ sets in the λP system, we place a number of tokens equal to the multiplicity of X in the place corresponding to $(X, M, \textit{position of } M \textit{ inside the initial membrane structure})$.

□

Corollary 5.15. (Translation of λP systems to the dual linear calculus)
Each computation performed by a λP system S, under the assumptions of Theorem 5.14, can be translated to its equivalent in the dual linear calculus presented in Section 4.2.

Proof:
Thanks to Theorem 5.14, a generic λP system can be translated to an equivalent Petri net N. It now suffices to encode N into a co-$\mathbf{IMLL}^{\wp}_{\searrow}$ computation, as shown in Sections 3.2 and 3.2.1, and to make use of the term assignment given in Table 6 to obtain the final dual linear calculus translation. □

[11] Just like Ⓐ, $[^A]$ may be used in this context only to signal that the specific membrane is considered in the rule: see Rule 2 in Section 3.3 of [21] for an example. For this reason, we need to interpret the notation and decide whether the transfer is required or not before performing the translation.

The translation we proposed has the clear advantage of showing that the intrinsic parallelism and distributed aspect of λP systems (all the applicable rules are executed at the same time in multiple membranes) and of Petri nets (where all suitable transitions, involving different places, are fired concurrently) can be expressed effectively by our dual linear calculus. An important point to be stressed is that we had to make use of the upper bound on the multiplicities to ensure the number of possible transitions is not infinite (in that case, the resulting Petri net would be equivalent to an infinite encoding in co-$\mathbf{IMLL}^{\wp}$); a further step towards a general translation technique could be extending this proof and making use of boundedness analysis algorithms to prune the unused places and transitions.

6. Concluding remarks and future work.

In this paper we reconsidered a version of Milner's *synchronous π-calculus* used in the early days of the subject and Milner and Abramsky's translation of classical multiplicative linear logic into the synchronous π-calculus as documented in [10]. In particular we revisited the efforts to represent the *"logical flow of information"* in Girard's proof nets in the *input-output* behaviour of interacting π-calculus terms *modulo structural congruence*. We saw that such "logical flow of information" has the form of the abstract *game semantics* in the framework of Chu's construction ([4]) and noticed that the most natural π-calculus representation of it would involve the *non-deterministic choice operator* (+) *modulo a suitable bisimulation*. We also noticed that Milner's translation of the *linear λ calculus* into the π-calculus seems to reverse the logical orientation and suggested that in a typed setting the π-calculus translation is best seen as a dualizing operation, i.e., passing to co-intuitionistic linear logic.

For such a logic we presented a *linear calculus of co-routines*: this is a co-intuitionistic version of Tristan Crolard's calculus of co-routines [15], originally proposed in the setting of the classical λμ-calculus. One of our goals here is to investigate the properties of such a calculus and to compare it with other systems of *logics for concurrency*. Thus as a concluding remark we ask the question whether *the linear calculus of co-routines is suitable for representing not only concurrent but also distributed aspects of computation.*

Co-intuitionistic logic is naturally formalized in a *single-premise multiple-conclusions* sequent calculus (or natural deduction) system and our calculus assigns a term to each formula in the succedent of a sequent: this fact suggests an interpretation of our term assignment as a distributed system, where each formula inhabits a distinct location. Moreover the terms for *subtraction* and the behaviour of their redexes present intriguing "geometric" features. A subtraction redex is a *p-terms* of the form `postpone` *the threads* $\mathtt{y} \mapsto u$ *in virtue of* $\mathtt{make\text{-}coroutine}(t, \mathtt{x})$) and are not assigned to a logical type: in our calculus p-terms are *control expressions* corresponding to the pointers marking the discharge of assumptions in ordinary Prawitz-style natural deduction. Thus the reduction of a redex for subtraction appears to involve *"broadcasting remote substitutions"* to terms in different locations.

Clearly these suggestions do not constitute conclusive evidence for claiming that the linear calculus of coroutines is a distributed system. But it may be useful to compare the form and behaviour of various term assignments considered above, in the interpretation where each formula in a multiple-conclusion sequent inhabits a distinct location.

- The λμ-calculus is typed in a sequent style natural deduction system with sequents of the form

$$x_1 \colon A_1, \ldots, x_n \colon A_n \vdash t \colon C \mid \alpha_1 \colon B_1, \ldots, \alpha_m \colon B_m$$

where all computation takes place within the term t: clearly here we have a calculus based on *"central control"*.

- The π-calculus translations of linear derivations can be represented in sequents of the form

$$\vdash V_{a_1 \ldots a_n} \mid a_1 \colon A_1, \ldots, a_n \colon A_n$$

where computation within $V_{a_1 \ldots a_n}$ is concurrent and $a_1, \ldots, a_n$ are *the accessible ports* to it. Here the *hiding operator* (νx) acts globally on the space of computation whose *"ports"* may be distributed.

- The co-intuitionistic linear calculus of coroutines has sequents of the form

$$x \colon A \vdash \pi \mid t_1 \colon C_1, \ldots, t_n \colon C_n$$

where the terms $t_1, \ldots, t_n$ are indeed *"distributed"* to the locations of $A_1, \ldots, A_n$ and the *"control terms"* in π are in a separate area. Reduction of *par*-redexes is local; reduction of *subtraction*-redexes is within "control terms" but has global effects. An explicit *hiding operator* is avoided through the use of functions $\mathtt{x}(t)$ representing dependencies.

The distributed features exhibited by the co-intuitionistic linear calculus of co-routines do not guarantee that it may be applied to issues (such as security) that are relevant to the study of distributed computing as a logic of distributed processes. On the other hand, the π-calculus was not designed specifically for distributed computation. Indeed since the 1990s *distributed versions* of the π-calculus have been provided (see, e.g., [2, 32, 19, 18] and the book [17]) as a theoretical framework for the study of real distributed systems in presence of nodes and link failure, or for the solution of concrete problems, such as safety and control of mobile code. Distributed π-calculi extend the π-calculus with symbols and actions for *locations* and sometimes differ in the treatment of basic operators: in some early papers, e.g., in [2, 32], communication is global, while for others [19, 18] only processes in the same location can interact.

We extended our research in the direction of a paradigmatic example of distributed computing, namely, λP systems of *membrane computing*. The tight connections are known between λP systems and Petri Nets and also between Petri Nets and intuitionistic (and thus co-intuitionistic) multiplicative linear logic: thus our work here is a first investigation into the details of the resulting correlation, to be continued and refined in the future.

Also for future work is reconsidering π-calculus translations of the whole system of linear logic linear with suitable notions of bi-simulation and its applications: here we may exploit recent work by E. Beffara [3] and the new direction of research on *session types* in the framework of a system of functional programming in L. Caires and F. Pfenning ([12]) and in P. Wadler ([34]).

References

[1] Abramsky, S.: Proofs as Processes, *Theor. Comput. Sci.*, **135**(1), 1994, 5–9.

[2] Amadio, R. M.: An Asynchronous Model of Locality, Failure and Process Mobility, *COORDINATION* (D. Garlan, D. L. Métayer, Eds.), 1282, Springer, 1997, ISBN 3-540-63383-9.

[3] Beffara, E.: A Concurrent Model for Linear Logic., *Electr. Notes Theor. Comput. Sci.*, **155**, 2006, 147–168.

[4] Bellin, G.: Chu's Construction: A Proof-theoretic Approach, *Logic for Concurrency and Synchronisation* (R. J. de Queiroz, Ed.), number 18 in Kluwer Trends in Logic, 2003.

[5] Bellin, G.: A Term Assignment for Dual Intuitionistic Logic, *LICS'05-IMLA'05 Workshop, Chicago, IL*, June 2005.

[6] Bellin, G.: Assertions, hypotheses, conjectures and expectations. Rough set semantics and proof-theory, *Advances in Natural Deduction - Proceedings of the Natural Deduction conference in Rio de Janeiro*, 2011.

[7] Bellin, G., Biasi, C.: Towards a logic for pragmatics. Assertions and conjectures, *Journal of Logic and Computation*, **14**(4), 2004, 473–506.

[8] Bellin, G., Biasi, C.: Towards a Logic for Pragmatics. Assertions and Conjectures., *J. Log. Comput.*, **14**(4), 2004, 473–506.

[9] Bellin, G., Hyland, M., Robinson, E., Urban, C.: Categorical proof theory of classical propositional calculus, *Theor. Comput. Sci.*, **364**(2), 2006, 146–165.

[10] Bellin, G., Scott, P. J.: On the π-Calculus and Linear Logic, *Theor. Comput. Sci.*, **135**(1), 1994, 11–65.

[11] Bierman, G.: *On Intuitionistic Linear Logic*, Technical Report 346, University of Cambridge Computer Laboratory, 1994, PhD Dissertation.

[12] Caires, L., Pfenning, F.: Session Types as Intuitionistic Linear Propositions, *CONCUR* (P. Gastin, F. Laroussinie, Eds.), 6269, Springer, 2010, ISBN 978-3-642-15374-7.

[13] Cockett, R., Seely, R.: Linear Distributive Categories, *Journal of Pure and Applied Algebra*, (114), 1997, 133–173.

[14] Colson, L., Jonoska, N., Margenstern, M.: λ-P Systems and Typed λ-Calculus, *Workshop on Membrane Computing* (G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, Eds.), 3365, Springer, 2004, ISBN 3-540-25080-8.

[15] Crolard, T.: A Formulae-as-Types Interpretation of Subtractive Logic, *J. Log. Comput.*, **14**(4), 2004, 529–570.

[16] Girard, J.-Y.: Linear Logic, *Theor. Comput. Sci.*, **50**, 1987, 1–102.

[17] Hennessy, M.: *A distributed π-calculus*, Cambridge University Press, 2007.

[18] Hennessy, M., Merro, M., Rathke, J.: Towards a behavioural theory of access and mobility control in distributed systems, *Theor. Comput. Sci.*, **322**(3), 2004, 615–669.

[19] Hennessy, M., Riely, J.: Resource Access Control in Systems of Mobile Agents, *Inf. Comput.*, **173**(1), 2002, 82–120.

[20] Hyland, M., de Paiva, V.: Full Intuitionistic Linear Logic, *Ann. Pure Appl. Logic*, **64**(3), 1993, 273–291.

[21] Jonoska, N., Margenstern, M.: Tree Operations in P Systems and λ-Calculus, *Fundam. Inform.*, **59**(1), 2004, 67–90.

[22] Kleijn, J., Koutny, M., Rozenberg, G.: Towards a Petri Net Semantics for Membrane Systems., *Workshop on Membrane Computing* (R. Freund, G. Paun, G. Rozenberg, A. Salomaa, Eds.), 3850, Springer, 2005, ISBN 3-540-30948-9.

[23] Krivine, J.-L.: *Lambda-calculus, types and models.*, Ellis Horwood series in computers and their applications, Masson, 1993, ISBN 978-0-13-062407-9.

[24] Lamarche, F.: Proof Nets for Intuitionistic Linear Logic: Essential Nets, 2008.

[25] Laneve, C., Victor, B.: Solos In Concert, *Mathematical Structures in Computer Science*, **13**(5), 2003, 657–683.

[26] Martí-Oliet, N., Meseguer, J.: From Petri Nets to Linear Logic, *Category Theory and Computer Science, Manchester, UK*, number 389, Springer-Verlag, 1989.

[27] Merro, M.: On the observational theory of the CPS-calculus, *Acta Inf.*, **47**(2), 2010, 111–132.

[28] Milner, R.: The polyadic π-calculus: A tutorial, *Logic and Algebra of Specification* (Bauer, Brauer, Schwichtenberg, Eds.), 94, NATO ASI, Springer, 1993.

[29] Păun, G.: Computing with Membranes, *J. Comput. Syst. Sci.*, **61**(1), 2000, 108–143.

[30] Păun, G.: Introduction to Membrane Computing, in: *Applications of Membrane Computing* (G. Ciobanu, M. J. Pérez-Jiménez, G. Păun, Eds.), Natural Computing Series, Springer, 2006, ISBN 978-3-540-25017-3, page 1–42, Also available at `http://psystems.disco.unimib.it/download/MembIntro2004.pdf`.

[31] Sangiorgi, D., Walker, D.: *The π-Calculus - a theory of mobile processes*, Cambridge University Press, 2001, ISBN 978-0-521-78177-0.

[32] Sewell, P.: Global/Local Subtyping and Capability Inference for a Distributed π-calculus, *ICALP* (K. G. Larsen, S. Skyum, G. Winskel, Eds.), 1443, Springer, 1998, ISBN 3-540-64781-3.

[33] Thielecke, H.: *Categorical Structure of Continuation Passing Style*, Ph.D. Thesis, University of Edinburgh, 1997, Also available as technical report ECS-LFCS-97-376.

[34] Wadler, P.: Propositions as Sessions, Draft paper, available at `http://homepages.inf.ed.ac.uk/wadler/topics/recent.html`.

7. APPENDIX. Examples of computation.

In this appendix we come back to the example in Section 3.2 and in Section 7.1 we give a derivation in Natural Deduction **co**-$\mathbf{MNJ}^{\diagdown}$, labelled with terms of the *linear calculus of coroutines*, which is the dual of the following Prawitz style $\mathbf{MNJ}^{\multimap}$ Natural Deduction derivation of

$$g : T_2, f : T_1, a : A, b : B, d : D \vdash (\lambda d \lambda d \lambda a.g(fab)d)bda : E$$

where $T_1 = A \multimap (B \multimap C)$ and $T_2 = C \multimap (D \multimap E)$.

$$\dfrac{\dfrac{g : T_2 \qquad \dfrac{\dfrac{f : T_1 \qquad a : A}{fa : B \multimap C}(1) \qquad b : B}{fab : C}(2)}{g(fab) : D \multimap E}(3) \qquad d : D}{g(fab)d : E}$$

$$(1)\ \dfrac{g(fab)d : E}{\lambda a.(g(fab)d) : A \multimap E}$$

$$(3)\ \dfrac{\lambda a.(g(fab)d) : A \multimap E}{\lambda d \lambda a.(g(fab)d) : D \multimap (A \multimap E)}$$

$$(2)\ \dfrac{\lambda d \lambda a.(g(fab)d) : D \multimap (A \multimap E) \qquad b : B}{\big(\lambda b.\lambda d \lambda a.(g(fab)d)\big)b : D \multimap (A \multimap E)}$$

$$\dfrac{\big(\lambda b.\lambda d \lambda a.(g(fab)d)\big)b : D \multimap (A \multimap E) \qquad d : D}{\big(\lambda b.\lambda d \lambda a.(g(fab)d)\big)bd : A \multimap E}$$

$$\dfrac{\big(\lambda b.\lambda d \lambda a.(g(fab)d)\big)bd : A \multimap E \qquad a : A}{\big(\lambda b.\lambda d \lambda a.(g(fab)d)\big)bda : E}$$

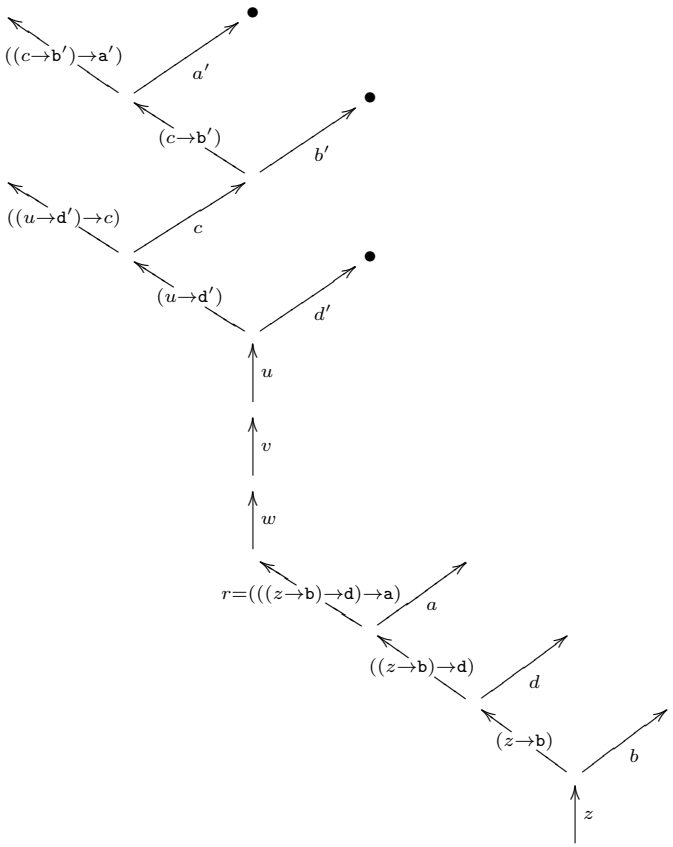
Next in Section 7.2 we translate the **co**-$\mathbf{MNJ}^{\diagdown}$ derivation in our membrane computing model and represent one step of the normalization process.

7.1. Example in the dual linear calculus

For convenience, we still draw trees with the root at the bottom, keeping in mind that here derivations are built from bottom up. We shall use the notation $(t \to \mathtt{a})$ for $\mathtt{mkc}(t, a)$ and $\xrightarrow{e\mapsto u} t$ for $\mathbf{postp}(e \mapsto u, t)$.

$$\mathcal{S}_0 : \underset{\mathcal{R}ed_0}{\overset{\mathtt{w}\mapsto a'}{\leftarrow}} r, \quad \overset{\mathtt{u}\mapsto b'}{\leftarrow} v, \quad \overset{\mathtt{v}\mapsto d'}{\leftarrow} w, \qquad a, \qquad d, \qquad b, \qquad ((c \to b') \to a'),$$

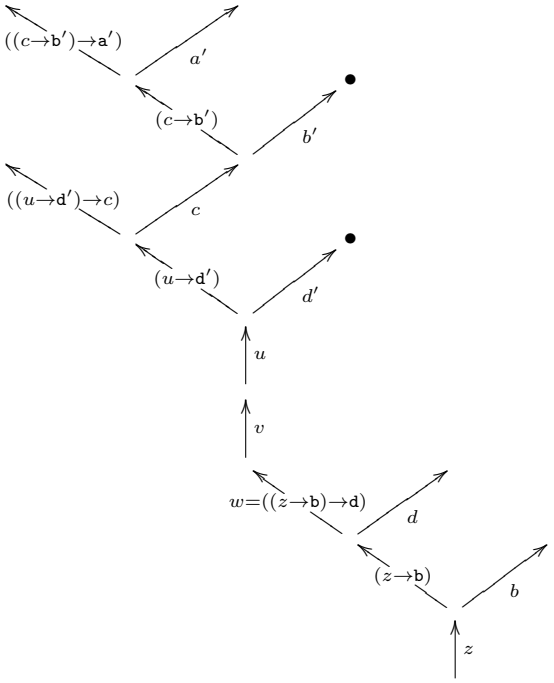
$$((u \to d') \to c)$$



reduces to

$\mathcal{S}_1 :$ $\underset{\mathcal{R}ed_1}{\overset{\mathtt{v}\mapsto d'}{\longleftarrow}} w,$ $\overset{\mathtt{u}\mapsto b'}{\longleftarrow} v,$ $a',$ $d,$ $b,$ $((c \to b') \to a'),$

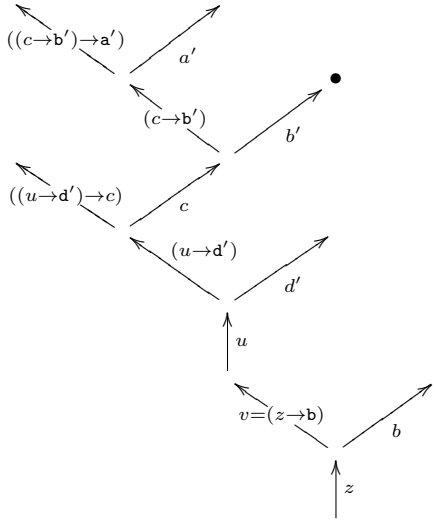
$((u \to d') \to c)$



reduces to

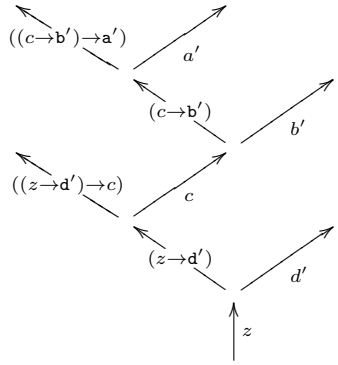
$\mathcal{S}_2$: $\underset{\mathcal{R}ed_2}{\overset{\mathtt{u}\mapsto b'}{\twoheadleftarrow}} v$, a', d', b, $((c \to b') \to a')$,

$((u \to d') \to c)$



reduces to

$\mathcal{S}_3$: a', b', d', $((c \to b') \to a')$ $((u \to d') \to c)$



We show here the steps of the computation:

$\mathcal{S}_0$:

$\underset{\mathcal{R}ed_0}{\overset{\mathtt{w}\mapsto a'}{\leftarrow}} r,\ \overset{\mathtt{u}\mapsto b'}{\leftarrow} v,\ \overset{\mathtt{v}\mapsto d'}{\leftarrow} w, \qquad a,\ d,\ b,\ ((c \to b') \to a'),\ ((u \to d') \to c)$

where $r = (((z \to \mathtt{b}) \to \mathtt{d}) \to \mathtt{a}),$ and $b = \mathbf{b}(z),\ d = \mathbf{d}((z \to \mathtt{b})),$
$a = \mathbf{a}(((z \to \mathtt{b}) \to \mathtt{d})),\ w = \mathbf{w}(r),\ v = \mathbf{v}(w),\ u = \mathtt{u}(v),\ d' = \mathbf{d}'(u),$
$c = \mathbf{c}((u \to \mathtt{d}')),\ b' = \mathbf{b}'(c),\ a' = \mathbf{a}(c \to \mathtt{b}').$

Reducing $\mathcal{R}ed_0$:
$\mathcal{S}_1 = \mathcal{S}_0 - \mathcal{R}ed_0\ \{w := ((z \to \mathtt{b}) \to \mathtt{d})\}\ \{a := a'\{w := ((z \to \mathtt{b}) \to \mathtt{d})\}\}$

$\mathcal{S}_1$: $\underset{\mathcal{R}ed_1}{\overset{\mathtt{v}\mapsto d'}{\leftarrow}} w, \overset{\mathtt{u}\mapsto b'}{\leftarrow} v, \qquad a', \quad d, \quad b, \quad ((c \to b') \to a'),\ ((u \to d) \to c)$

where $w = ((z \to \mathtt{b}) \to \mathtt{d})$ and $b = \mathbf{b}(z),\ d = \mathbf{d}((z \to \mathtt{b})),\ v = \mathbf{v}(w),$
$u = \mathtt{u}(v),\ d' = \mathbf{d}'(u),\ c = \mathbf{c}((u \to \mathtt{d}')),\ b' = \mathbf{b}'(c),\ a' = \mathbf{a}((c \to \mathtt{b}')).$

Reducing $\mathcal{R}ed_1$: $\mathcal{S}_2 = \mathcal{S}_1 - \mathcal{R}ed_1\ \{v := (z \to \mathtt{b})\}\ \{d := d'\{v := (z \to \mathtt{b})\}\}.$

$\mathcal{S}_2 : \underset{\mathcal{R}ed_2}{\overset{\mathtt{u}\mapsto b'}{\leftarrow}} v \qquad a', \quad d', \quad b, \quad ((c \to b') \to a')\ ((u \to d) \to c)$

where $v = (z \to \mathtt{b})$ and $b = \mathbf{b}(z),\ u = \mathtt{u}(v),\ d' = \mathbf{d}'(u),$
$c = \mathbf{c}(u \to \mathtt{d}'),\ b' = \mathbf{b}'(c),\ a' = \mathbf{a}((c \to \mathtt{b}')).$

Reducing $\mathcal{R}ed_2$: $\mathcal{S}_3 = \mathcal{S}_2 - \mathcal{R}ed_2 \quad \{u := z\}\ \{b := b'\{u := z\}\}.$

$\mathcal{S}_3 : \qquad a', \quad b', \quad d', \quad ((c \to b') \to a'),\ ((z \to d) \to c)$

where $d' = \mathbf{d}'(z), \quad c = \mathbf{c}((z \to \mathtt{d}')), \quad b' = \mathbf{b}'(c), \quad a' = \mathbf{a}((c \to \mathtt{b}')).$

7.2. Example in a λP system

We will show here, in tree form, the first β-reduction of the derivation we implemented in the dual linear calculus in Subsection 7.1.

We start at first with the computational context $\texttt{postp}(\texttt{w} \mapsto a', r)$, $\texttt{postp}(\texttt{u} \mapsto b', v)$, $\texttt{postp}(\texttt{v} \mapsto d', w)$, $\texttt{mkc}(\texttt{mkc}(c, \texttt{b'}), a')$, $\texttt{mkc}(\texttt{mkc}(u, \texttt{d'}), c)$, with all the additional subterms defined as in Subsection 7.1. In Fig. 2 we build the initial tree (note that, for brevity, some subterms have been omitted and that many substitutions in the parts of the tree that are not affected by the first reduction have not been performed). In Fig. 3 we show the state of the tree after the first `postp` term has been reduced.

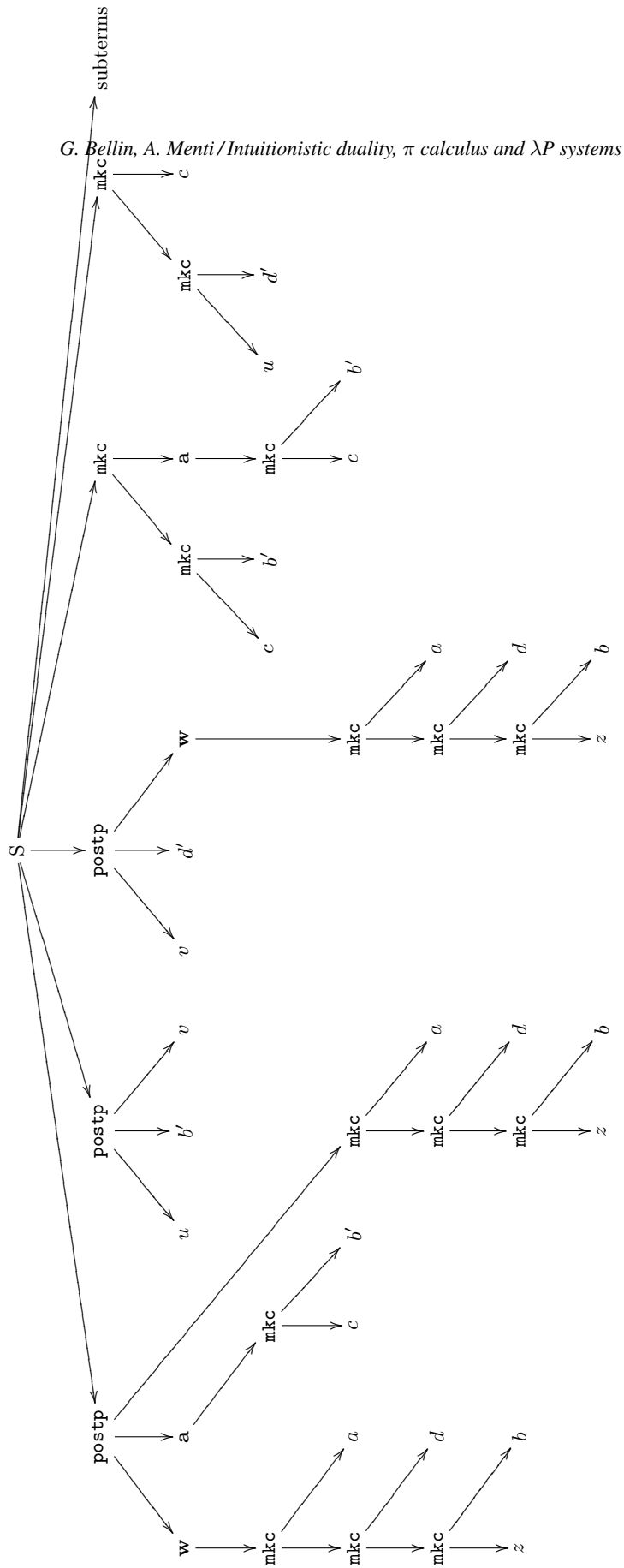


Figure 2. Tree representation of the initial state of the λP system.

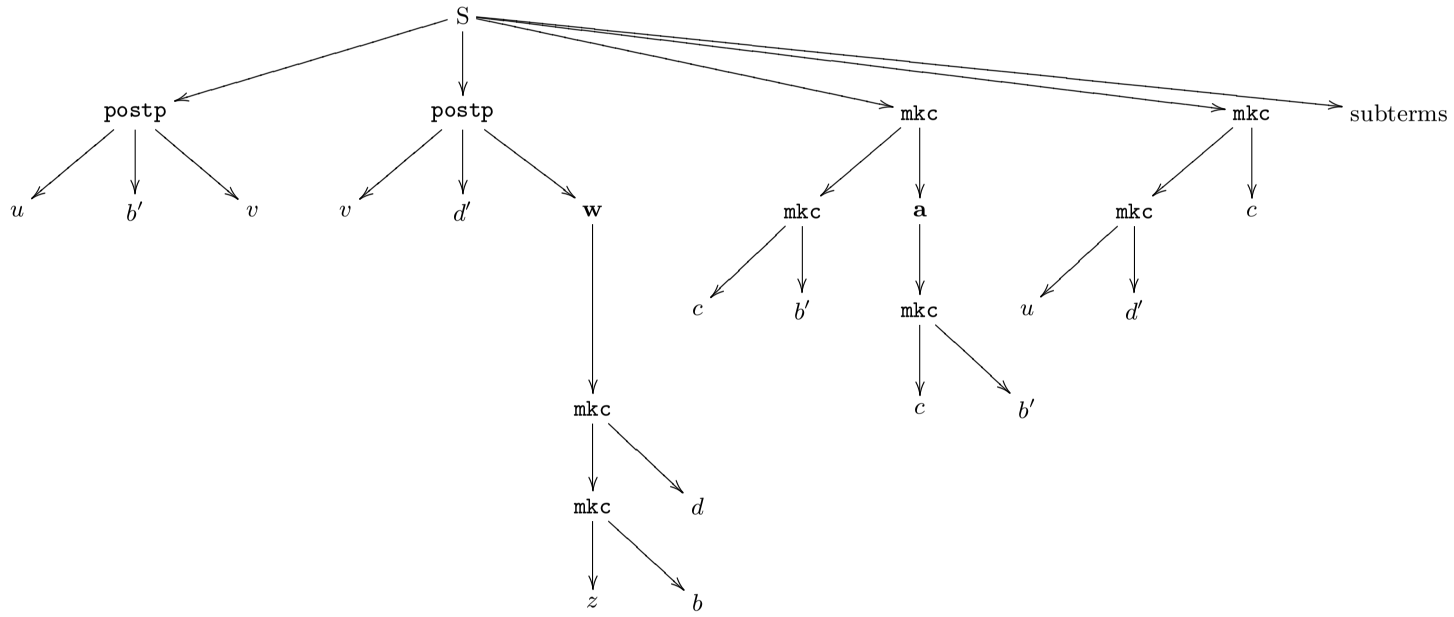


Figure 3. Tree representation of the state of the λP system after the first reduction has been performed. Notice that the `postp` term has been removed and that the substitution in the **w** term has been done. Although it is not shown here for space reasons, the a term has been replaced as well, as described in Subsection 7.1.