Differential linear logic and polarization

Lionel Vaux

Differential Linear Logic and Polarization

Lionel Vaux *
Laboratoire de Mathématiques de l'Université de Savoie,
UMR 5127 CNRS 73376 Le Bourget-du-Lac Cedex, FRANCE
lionel.vaux@univ-savoie.fr

Last modified: 2009-05-06

Abstract

We extend Ehrhard–Regnier's differential linear logic along the lines of Laurent's polarization. We provide a denotational semantics of this new system in the well-known relational model of linear logic, extending canonically the semantics of both differential and polarized linear logics: this justifies our choice of cut elimination rules. Then we show this polarized differential linear logic refines the recently introduced convolution $\bar{\lambda}\mu$-calculus, the same as linear logic decomposes λ-calculus.

1 Introduction

Differential Linear Logic. Differential interaction nets (DIN) were introduced by Ehrhard and Regnier in [1] to provide a notion of proof nets for the finitary fragment of their differential λ-calculus [2]. Both DIN and differential λ-calculus originate in the study of models of linear logic designed after Girard's quantitative semantics of λ-calculus [3], such as Ehrhard's finiteness spaces [4]. The distinctive attribute of these models is that intuitionistic proofs, hence typed λ-terms, are interpreted by power series in particular vector spaces; thus it makes sense to define differentiation on these.

The differential λ-calculus embodies this notion of differentiation, in close correspondence with the linear logic approach to resources of computation: a functional program is linear when uses its argument exactly once. The same as the derivative of a smooth function can be thought of as its best linear approximation, the derivative of an abstraction $\mathrm{D}(\lambda x\, s)\cdot t$ reduces to an abstraction $\lambda x \left(\frac{\partial s}{\partial x}\cdot t\right)$ where $\frac{\partial s}{\partial x}\cdot t$ is obtained by substituting t for exactly one linear occurrence of x in s, where "linear occurrence" means an occurrence which is used exactly once in the head reduction of s. There can be many such occurrences in a term, hence one actually considers the sum of all such terms, which is similar to the well known rule for the derivative of a product: $(f\times g)' = f'\times g + f\times g'$.

Such a differential extension can be reproduced in linear logic: it boils down to the introduction of costructural rules, dual to linear logic structural rules, and a codereliction rule, dual to dereliction. Costructural rules reflect an algebraic structure on exponentials, with a convolution product $\mathrm{m} : !A\otimes !A \multimap !A$ and its unit $\mathrm{u} : \mathbf{1} \multimap !A$. The basis of differentiation in

*Supported by French ANR project CHoCo

these models is that morphisms $f : !A \multimap B$ are power series: then codereliction $\partial : A \multimap !A$ is such that $f \circ \partial : A \multimap B$ is the linear part of f, i.e. its derivative at point 0; together with the convolution product, this defines the derivative at any point. The cut elimination procedure then reflects valid equations in the model.

The system of DIN presented in [1] is not exactly an extension of linear logic: the promotion rule is missing. It is however possible to reintroduce it, together with appropriate cut elimination rules derived from the semantics in finiteness spaces: this defines differential nets (DN), which depart from the interaction net paradigm (see, e.g. [5]). One can naturally introduce a sequent calculus associated with DN, where cut elimination is guided by the reduction of nets: call differential linear logic (DiLL) this system.

Polarization. The notion of polarities in linear logic was made prominent by Andreoli's work on focusing proofs [6] and Girard's deterministic system for classical logic [7]. The latter led to the definition of polarized linear logic (LLP) by Laurent [8]: in the polarized fragment of linear logic, the structural rules can be extended to all negative formulas rather than ?-formulas only. It is well known that the transition from an intuitionistic system to a classical one can be performed by allowing deductions with multiple conclusion formulas. Since negative formulas are the target language of Girard's translation of implicative formulas into linear logic, we understand that LLP corresponds to such a relaxation.

The computational counterpart of classical logic is well established: classical truths type control operators. It is moreover possible to extend the Curry–Howard correspondence to a classical logic setting, while retaining the intuition of proofs with multiple conclusions. For instance, Parigot's $\lambda\mu$-calculus and Herbelin's $\bar{\lambda}\mu$-calculus can be considered as calculi of programs with multiple outputs, controlled by term constructions reflecting polarized structural rules. These enjoy decompositions into LLP, similar to the translation of λ-calculus into linear logic proof nets studied by Danos and Regnier [9, 10].

From a semantical point of view, the idea that polarization canonically extends the structure of exponentials to polarized formulas is also valid. For instance Girard's correlation spaces [7] are coherence spaces equiped with a $\parr$-monoid structure and provide a semantics of LLP [8]: the interpretation of costructural rules on polarized formulas is built from that on their subformulas (basically variables and exponentials). Work by Laurent and Regnier [11] later showed that this construction generalizes: the $\parr$-monoids of a Lafont category [12] form a model of LLP.

Polarized Costructural Rules. In short, DiLL introduces a symmetry on exponential types, with costructural rules, and provides a differential analysis of proofs through a computational notion of derivatives. On the other hand, LLP extends the linear decomposition of intuitionistic logic to classical logic, by relaxing structural rules, i.e. by canonically extending the structure of exponentials to polarized formulas. This motivates the study of the relations entertained by both of these extensions of the Curry–Howard correspondence and its analysis by linear logic.

A first result was provided in [13]: the author introduces a differential $\lambda\mu$-calculus which is a conservative extension of both $\lambda\mu$-calculus and differential λ-calculus, enjoying confluence and strong normalization of typed terms: the definability of such a system witnesses a compatibility between both extensions, and does not involve any new logical interaction. Indeed, although this is not done in [13], one can consider the system obtained as the union of rules of DiLL and LLP, then check that any kind of cut in this system is already covered by

the cut elimination rules of DiLL or LLP: this is the target of a translation of differential $\lambda\mu$-calculus extending naturally that of differential λ-calculus in DiLL and that of $\lambda\mu$-calculus in LLP.

In the present paper we rather investigate the effect of polarization on DN: we consider the system obtained by relaxing not only structural rules to negative formulas but also costructural rules to positive formulas. Again the idea is that polarization should also extend the algebraic structure of exponentials to polarized formulas. In particular, this preserves the symmetry between structural and costructural rules introduced in DiLL.

There are two main guiding lines when designing cut elimination in this system: symmetry and semantics. We consider a model of linear logic which can be extended to both DiLL and LLP: both correlation spaces and finiteness spaces are refinements of the relational model which underlies Girard's coherence semantics. More: in the relational interpretation of linear logic proofs, duality boils down to reversing the orientation of relations. This allows to deduce, in a very natural way, the semantics of polarized costructural rules from that of polarized structural rules: just reverse the corresponding relations.

The reflexive object introduced in [14] is well suited for this study: it allows to interpret both DiLL and LLP in a pure (i.e. untyped) setting, so that exponential structural and costructural rules are exchanged by symmetry, and polarized structural rules are given by a $\parr$-monoid structure on the object. It is then easy to derive the computational behaviour of polarized costructural rules from this semantics. The system presented in the current paper can be seen as the end result of this course of thought.

In [15], the author introduced convolution $\bar{\lambda}\mu$-calculus based on similar ideas: interpret Herbelin's $\bar{\lambda}\mu$-calculus into the object of [14] through LLP, then investigate the computational counterpart of the monoid operation modelling polarized costructural rules, when applied to the denotations of contexts, which are dual to terms.

Organization of the paper. In section 2, we introduce the system of polarized differential nets (PDN), together with typing and reduction rules. Then, in section 3, we validate this new system by providing a denotational semantics on a particular object of the relational model of linear logic. This canonically extends the relational semantics of both DN and LLP. Section 4 briefly reviews sequentialization of PDN. Last, section 5 explicits the translation of the convolution $\bar{\lambda}\mu$-calculus in PDN, as hinted in [15]. The end of the paper proposes a quick glimpse at how to bring differentiation back into that setting.

2 Polarized Differential Nets

Polarized differential nets (PDN) are formal finite sums of simple nets, which are particular multiport interaction nets, such as studied by Mazza [16] following Lafont [17]. The cells of simple PDN are actually those of DN, i.e. DIN plus promotion boxes. Mainly, PDN differ from DN when considering typing, which is relaxed by polarization, and cut elimination, which involves new rules. An example simple PDN is given in Figure 1, with a pure typing: this is the translation of convolution $\bar{\lambda}\mu$-calculus closed term $\lambda x\, \mu\alpha\, \langle x\, ,\, (x \cdot \alpha) * \alpha \rangle$ (see below).

Nets. We call *signature* a set Σ of symbols, where each symbol $\alpha \in \Sigma$ is given an arity $\mathsf{a}(\alpha) \in \mathbf{N}$. A *simple net* on signature Σ is a circuit built up from a finite number of *cells*,

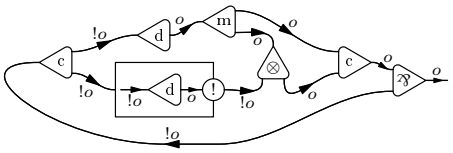


Figure 1: An example of simple net

each given a symbol in Σ, which are connected by finitely many *wires*, so that every cell c is connected to $\mathsf{a}(\alpha_c)+1$ wires, where α_c is the symbol of c. We allow wires with dangling ends, and also loop wires. The ends of wires are called *ports* (a loop is a wire whose ports are equal). Hence each port is either a cell port, or a loop port, or a *free port* (of a dangling wire). The placement of connexion points matters: cell ports are not interchangeable. If c is a cell, we write $c^0, \ldots, c^{\mathsf{a}(\alpha_c)}$ for its ports. Port c^0, which is always present, is the *principal* port of c; the possible other ones are called *auxiliary* ports. In general, cells are depicted by triangles, with their respective principal port put on the tip of the triangle, and the auxiliary ones on the opposite edge.

The *interface* of a simple net is the set of its free ports. A *net* μ is a multiset $[\mu_1, \ldots, \mu_n]$ of simple nets sharing the same interface, which we also consider to be the interface of μ. If μ and μ' are nets with the same interface, we denote additively their multiset union $\mu + \mu'$. We also denote by $\mathbf{0}$ the empty multiset of simple nets, whatever the underlying interface. This should not be confused with the empty simple net ε, the interface of which is empty.

We will consider cells of a special kind: a box is a cell with symbol $\mu^!$, where μ is a net whose interface matches the ports of the box-cell. A box $\mu^!$ is depicted as a rectangle containing μ, where we distinguish the principal port by a circled exclamation mark. Let Σ be a signature. We define the signature $\Sigma^!$ by induction on the depth of boxes: $\Sigma^! = \bigcup \Sigma^{(n)}$, where $\Sigma^{(0)} = \Sigma$, and if $\Sigma^{(n)}$ is defined, we set $\Sigma^{(n+1)} = \Sigma^{(n)} \cup \{\mu^!;\ \mu^! \text{ is a box symbol with } \mu \text{ a net on } \Sigma^{(n)}\}$. Notice that boxes may contain sums, since box symbols are not necessarily simple nets.

Definition 2.1 *The signature Δ_0 of PDN of depth 0 is that of Ehrhard–Regnier's DIN [1]: binary symbols tensor $\otimes$, par $\parr$, contraction* c *and cocontraction* m*; unary symbols dereliction* d *and codereliction ∂; and nullary symbols weakening* w *and coweakening* u*. Then the signature of all PDN is $\Delta = \Delta_0^!$.*

Typing. The polarized formulas of multiplicative exponential linear logic are given by the following mutually inductive grammars:

$$\begin{array}{rrcl} \text{negative:} & M, N & ::= & X \mid M \parr N \mid ?P \\ \text{positive:} & P, Q & ::= & X^\perp \mid P \otimes Q \mid !N \end{array}$$

with negation defined by De Morgan duality: $X^{\perp\perp} = X$, $(M \parr N)^\perp = N^\perp \otimes M^\perp$ and $(?P)^\perp = !P^\perp$. Recall that the linear logic formulas used in the decomposition of minimal implicative natural deduction (i.e. simply typed λ-calculus) through Girard's translation $A \Rightarrow B = !A \multimap B$ are the intuitionistic and exponential ones, organized as follows:

$$\begin{array}{rrcll} \text{negative:} & A, B & ::= & X \mid ?A^\perp \parr B & \qquad \text{why-not:} \quad ?A^\perp \\ \text{positive:} & A^\perp, B^\perp & ::= & X^\perp \mid A^\perp \otimes !B & \qquad \text{of-course:} \quad !A \end{array}$$

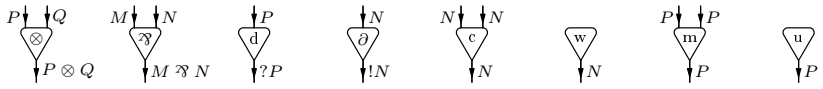


Figure 2: Typing of the cells of simple PDN

These are special cases of polarized formulas. Pure types were introduced by Danos [9] and Regnier [10] in order to interpret pure λ-calculus in linear logic. They are variable free intuitionistic and exponential formulas obtained from an additional constant o, subject to the equation $o = o \Rightarrow o$, which allows to type all pure λ-terms. This translates to $o = !o \multimap o = ?o^\perp \parr o$. We obtain four possible formulas: o itself (the type of terms), $i = o^\perp$ (its dual, the type of contexts), $!o$ (the type of arguments) and $?i$ (the type of free variables). In a pure setting, o (resp. i, $!o$, $?i$) is the only negative (resp. positive, of-course, why-not) formula: it can be considered as the archetypal one.

Definition 2.2 *A* typing *of a simple PDN is the assignment of a type to each oriented wire, such that reversing the orientation of the wire negates the type, and respecting some typing constraints on symbols. The typing rules for the cells of simple PDN of depth* 0 *are given in Figure 2, where types are polarized formulas. If $I = p_1, \ldots, p_k$ is an ordered interface of simple net μ, i.e. a list of its free ports, and if $\Gamma = \gamma_1, \ldots, \gamma_k$ is a list of types, then we write $\mu \vdash_I \Gamma$ if there is a typing of μ such that the* outgoing *type at port p_i is γ_i. We extend typing to all PDN, by the following rules for sums and boxes: if $\mu = \sum_{j=1}^m \mu_j$ is a net with ordered interface I, then $\mu \vdash_I \Gamma$ as soon as, for all j, $\mu_j \vdash_I \Gamma$; if moreover $I = \{0, \ldots, n\}$, $\Gamma = N, N_1, \ldots, N_n$ and $\mu \vdash_I \Gamma$, then for a box b of symbol $\mu^!$, we have $b \vdash_{b^0, \ldots, b^n} !N, N_1, \ldots, N_n$ (this is the promotion rule of LLP).*

Notice the four polarized typing rules for structural and costructural cells. From this polarized typing, one straightforwardly deduces an intuitionistic (resp. pure) type system for PDN.

Cut elimination. Among the ports of each cell, some are called *active*: these correspond to active formulas in the associated deduction rule. The only active port of a non-box cell is the principal one; by contrast, all the ports of a box are active. A *cut* in a net is a wire between active ports of distinct cells: a *redex* is the data of two cells c, d and indices of active ports i of c and j of d, such that $\left[c^i, d^j\right]$ is a wire. We denote such a redex by $\left\langle c^i \, , d^j \right\rangle$. For each typable redex $\left\langle c^i \, , d^j \right\rangle$, Figure 3 gives a reduced net which depends only on the symbols α_c and α_d, and the port indices i and j, so that the free ports of the reduced net are assigned to the free ports of the redex, i.e. the ports of c and d minus c^i and d^j.

Definition 2.3 *We first define reduction at depth* 0. *If μ is a simple net, $\left\langle c^i \, , d^j \right\rangle$ is a redex of μ and $\sum_{k=1}^n \nu_k$ is the reduct of $\left\langle c^i \, , d^j \right\rangle$ given in Figure 3, then we write $\mu \to_0 \sum_{k=1}^n \mu'_k$, where each μ'_k is the simple net obtained by removing wire $\left[c^i, d^j\right]$ and cells c and d from μ, then plugging ν_k instead. This is extended to sums as follows: $\mu \to_0 \mu'$ as soon as $\mu = \sum_{i=0}^n \mu_i$ and $\mu' = \sum_{i=0}^n \mu'_i$, where each μ_i is simple, $\mu_0 \to_0 \mu'_0$ and, for $i \in \{1, \ldots, n\}$, $\mu_i \to_0 \mu'_i$ or $\mu'_i = \mu_i$.*

We now define reduction at any depth. Assume $\to_n$ is defined. Then if μ is a simple net, $\mu \to_{n+1} \mu'$ if $\mu \to_0 \mu'$, or if there are nets ν and ν' such that $\nu \to_n \nu'$ and μ' is obtained

by replacing a box with symbol $\nu^!$ in μ with a box of symbol $\nu'^!$. We extend $\rightarrow_{n+1}$ on sums similarly to $\rightarrow_0$. We finally set $\mu \rightarrow \mu'$ if $\mu \rightarrow_n \mu'$ for some n.

We provided annotations for the reduction rules of Figure 3, organized as follows. Groups m and e are the cut elimination rules for multiplicative exponential linear logic. Groups m and r correspond to the reduction of DIN in [1]; if we add d, we obtain the reduction rules of DN, suitable to encode differential λ-calculus. Groups m, e and p define the cut elimination procedure of LLP. This is actually a local version of the reduction presented in [8]: group p and rules $e_{2,3,4}$ decompose in many steps the reductions of positive trees versus structural rules and auxiliary ports of boxes. The only new reduction rules in PDN are those of group p'.

It is easily checked that the left part of Figure 3, i.e. groups m, r, p' and p except p_3, define a confluent and terminating system. As first noticed by Tranquilli [5], however, even local confluence of the system including d_1 is only verified up-to some structural reductions (similar to the structural equivalence to be introduced in Definition 5.3): this is because rule d_1 forces which passive port of the convolution product m receives the linear argument provided by ∂.

A full study of the proof theoretic properties of PDN (including confluence and strong normalization) is left for future work: although similar questions for DN receive a partial answer in [5], much remains to be settled. In the following, we concentrate on a semantical justification of our choice of cut-elimination (section 3), remarks on sequentialization properties (section 4) and the computational expressivity of polarized costructural rules (section 5).

3 Relational Semantics

Following [14], we construct an object in the usual multiset based relational model of linear logic (the category of sets and relations, where multiplicatives are interpreted by cartesian products, and exponential modalities by the free commutative monoid construction, i.e. finite multisets) which is an extensional reflexive object in the co-Kleisli category associated with the ! modality.

If X is a set, denote by $\mathcal{M}_{\mathrm{fin}}(X)$ the set of all finite multisets of elements in X, and by $\mathcal{M}_{\mathrm{fin}}(X)^{(\omega)}$ the set of all sequences $\xi = (\xi(i))_{i\in\omega}$ of multisets in $\mathcal{M}_{\mathrm{fin}}(X)$ such that $\xi(i) = []$ for almost all $i \in \omega$. We define an increasing family $(\mathcal{D}_n)_{n\in\mathbf{N}}$ of sets by induction on n: $\mathcal{D}_0 = \emptyset$ and $\mathcal{D}_{n+1} = \mathcal{M}_{\mathrm{fin}}(\mathcal{D}_n)^{(\omega)}$. Then we set $\mathcal{D} = \bigcup_{n\in\mathbf{N}} \mathcal{D}_n$.

If $a \in \mathcal{M}_{\mathrm{fin}}(\mathcal{D})$ and $\alpha \in \mathcal{D}$, write $a :: \alpha$ for the sequence β such that $\beta(0) = a$ and $\beta(i+1) = \alpha(i)$ for all $i \in \omega$. We denote by ι the constant sequence such that $\iota(i) = []$ for all $i \in \omega$. For instance, $\mathcal{D}_1 = \{\iota\}$. The mapping $(a, \alpha) \mapsto a :: \alpha$ is clearly a bijection from $\mathcal{M}_{\mathrm{fin}}(\mathcal{D}) \times \mathcal{D}$ to $\mathcal{D}$, and satisfies $\iota = [] :: \iota$. This bijection makes $\mathcal{D}$ an extensional reflexive object in the cartesian closed category defined by the co-Kleisli construction on !, hence an extensional model of pure λ-calculus. It also provides a model of pure DiLL: the finitary semantics of [4] is easily reproduced with base types interpreted by $\mathcal{D}$, pruning the finiteness structure.

We show how this object provides a model of the reduction of pure PDN, first by defining a commutative monoid structure on $\mathcal{D}$, with unit ι: for all $i \in \omega$, set $(\alpha \star \beta)(i) = \alpha(i) + \beta(i)$. Following [11], we obtain a model of pure LLP; we show that this actually extends to a model of PDN. We call *relational type* any couple (γ, i), where i is an orientation bit 0 or 1, and

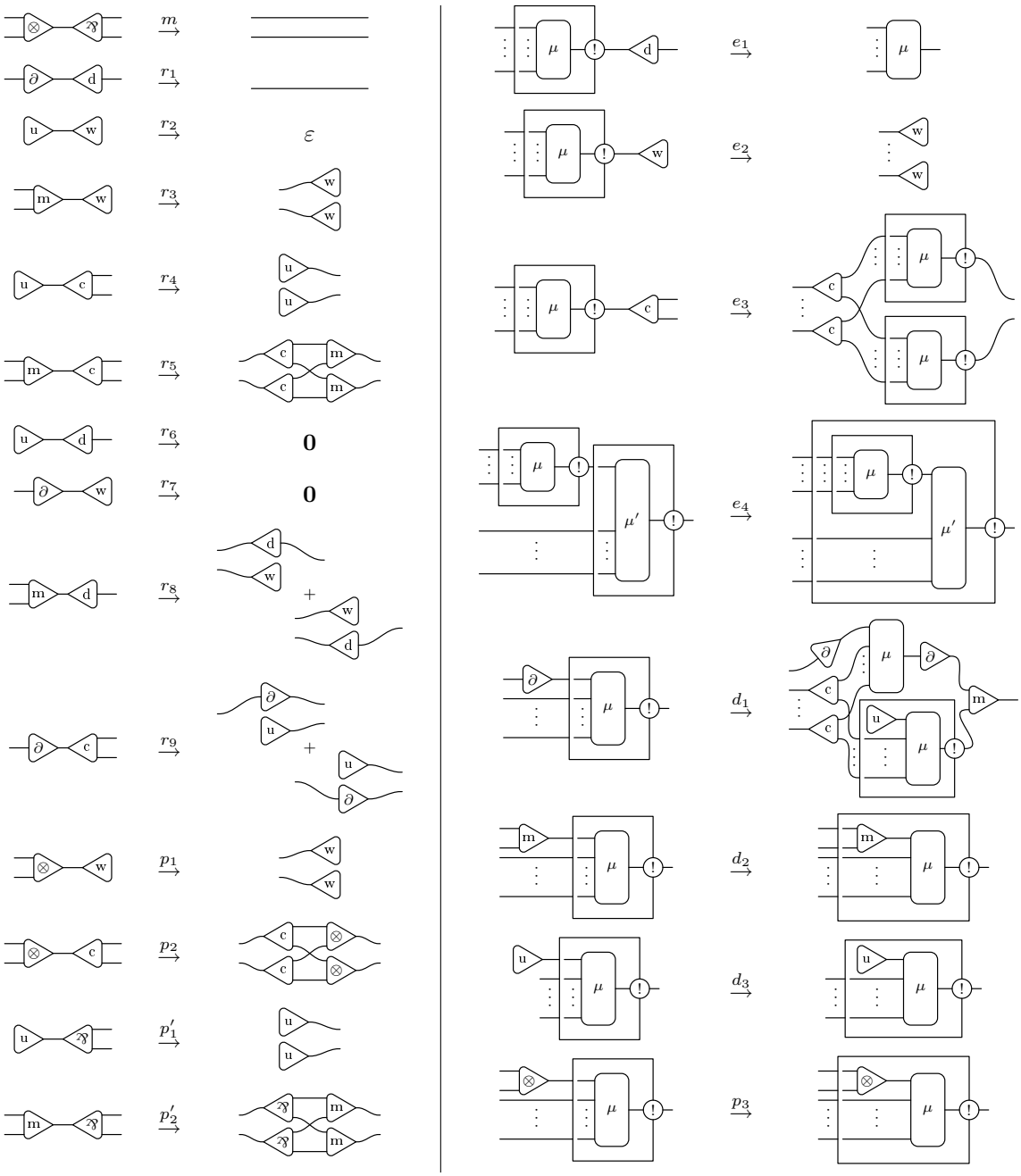


Figure 3: Reduction rules of PDN

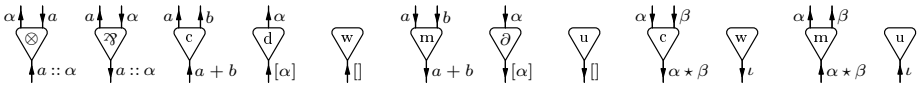


Figure 4: Relational typing of the non-box cells of PDN

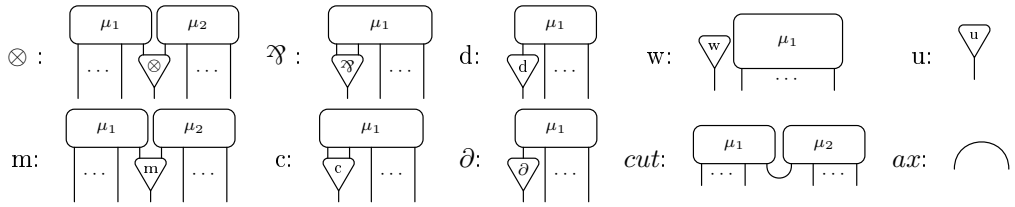


Figure 5: Sequentiality of PDN.

$\gamma \in \mathcal{M}_{\text{fin}}(\mathcal{D}) \cup \mathcal{D}$. We set the duality on types to negate the orientation bit. By convention, when depicting the relational typing of a PDN, we fix the orientation of wires so that the orientation bit is always the same (say 0): on these oriented wires, we only give the value α or a of the type.

Definition 3.1 *The rules of* relational typing *of simple PDN of depth* 0 *are given in Figure 4. This is extended to all PDN of depth* 0 *as follows: if* $\mu = \sum_{i=1}^{n} \mu_i$ *is a sum of simple nets with ordered interface* I*, then* $\mu \vdash_I \Gamma$ *as soon as* $\mu_i \vdash_I \Gamma$ *for some* i *(not necessarily all). It remains to define the typing of boxes. We write* $\mu : (\gamma_1, \ldots, \gamma_l \vdash_I \gamma'_1, \ldots, \gamma'_m)$ *if* $\mu \vdash_I (\gamma'_1, 0), \ldots, (\gamma'_m, 0), (\gamma_1, 1), \ldots, (\gamma_l, 1)$*. Assume* $I = 0, \ldots, l+m$ *and there are typings* $\mu : (a^i_1, \ldots, a^i_l \vdash_I \alpha_i, \beta^i_1, \ldots, \beta^i_m)$ *for* $i \in \{1, \ldots, n\}$*. Then for all box* b *of symbol* $\mu^!$*, we have* $b : (a_1, \ldots, a_l \vdash_I [\alpha_1, \ldots, \alpha_n], \beta_1, \ldots, \beta_m)$*, where* $a_j = \sum_{i=1}^{n} a^i_j$ *and* $\beta_k = \prod_{i=1}^{n} \beta^i_k$*. The* relational semantics *of a PDN is then the set of its input-output typings:* $[\![\mu]\!]_I = \{((\gamma_1, \ldots, \gamma_l), (\gamma'_1, \ldots, \gamma'_m));\ \mu : (\gamma_1, \ldots, \gamma_l \vdash_I \gamma'_1, \ldots, \gamma'_m)\}$.

Theorem 3.2 *The relational semantics of PDN is preserved under reduction.*

Proof One simply inspects the reduction rules and checks that they preserve all possible typings. Then one concludes by contextuality. □

4 Sequentiality

Definition 4.1 *A PDN is* sequential *if it is a sum of simple nets obtained by the rules of Figure 5, plus the formation of boxes, where simple nets* μ_1 *and* μ_2*, and nets inside boxes are inductively supposed to be sequential. It is said to be* weakly sequential *when one moreover allows the formation of the empty PDN* ε *and the juxtaposition (i.e. disjoint union) of simple nets as inductive cases.*

A first sequentiality criterion is provided by the well-know Danos-Regnier switching condition.

Definition 4.2 *Let* μ *be a simple PDN. A* switching *of* μ *is a graph* G *with vertices the ports of* μ *and with edges as follows: every wire of* μ *is an edge in* G*; for each cell* c *in* μ*,*

with symbol ⅋ or c*, there is an edge between* c^0 *and exactly one of the ports* c^i*,* $i > 0$*; for each cell* d *in* μ*, with symbol other than ⅋ or* c*, there is an edge between* d^0 *and every port* d^j*,* $j > 0$*. A PDN* ν *is* correct *if every switching of every simple net* ν *in* μ *is acyclic and, inductively, every PDN inside a box cell of* ν *is correct.*

Of course, the set of correct PDN is stable under cut elimination.

Theorem 4.3 *A PDN is weakly sequential iff it is correct.*

Proof One adapts easily the proof by Danos in [9] for MELL proof structures to the case of PDN. Indeed, this proof is only about the geometry of nets: here m is handled like $\otimes$, ∂ is handled like d and u is handled like a tensor unit.[1] □

One consequence of the polarization property in linear logic, as described by Laurent in [8], is that one can characterize exactly sequential proof structures, based on a simple criterion on so-called correctness graphs. This no longer applies here: the typing rule of codereliction breaks the constraining character of polarization. In particular, we can no longer claim that every typed and sequential net has at most one positive conclusion: this was an essential property of polarization in [10] and [8] (pure or intuitionistic PDN retain this property, however, as first noted by Tranquilli for intuitionistic DN [5]).

5 Convolution $\bar{\lambda}\mu$-calculus

We now recall the definition of the convolution $\bar{\lambda}\mu$-calculus of [15]. Like Herbelin's $\bar{\lambda}\mu$-calculus, it involves three distinct syntactic categories: terms (proofs with an active conclusion), contexts (proofs with an active hypothesis) and commands (cuts between active conclusions of terms and active hypotheses of contexts). It moreover introduces a binary operation on contexts, which is meant to provide a computational counterpart to the polarized costructural rules of PDN. It turns out the obtained reduction rules closely resemble the definition of the convolution product of distributions [18].

5.1 Syntax

Basic Syntax. Fix two denumerably infinite sets $\mathcal{V}$ (set of variables, denoted by x, y, z) and $\mathcal{N}$ (set of names, denoted by α, β, γ).

Definition 5.1 *Define terms, contexts and commands by:*

$$\begin{array}{rcll} s & ::= & x \mid \lambda x\, s \mid \mu\alpha\, c & \textit{(simple terms)} \\ \sigma & ::= & \alpha \mid S \cdot e & \textit{(stacks)} \\ e & ::= & \mathbf{1} \mid \sigma * e & \textit{(simple contexts)} \\ c & ::= & \langle s\,,\, e\rangle & \textit{(simple commands)} \end{array} \qquad \begin{array}{rcll} S & ::= & \mathbf{0} \mid s + S & \textit{(terms)} \\ E & ::= & \mathbf{0} \mid e + E & \textit{(contexts)} \\ C & ::= & \mathbf{0} \mid c + C & \textit{(commands)}\ . \end{array}$$

We consider terms, commands and contexts up to permutativity of sum in the sense that, e.g., $s + (s' + S) = s' + (s + S)$. Also, we consider simple contexts up to permutativity of convolution product: e.g., $\alpha * ((S \cdot e) * e') = (S \cdot e) * (\alpha * e')$. Notice that these identities preserve free and bound variables and names: hence they are compatible with α-conversion.

[1] Although this is not done in the present paper, one can introduce multiplicative units: **1** is positive and $\bot$ is negative. Then, by the polarized typing rules of PDN, **1** (resp. $\bot$) can be seen as a special case of u (resp. w).

Notations. We call simple object any simple term, simple context or simple command, and object any term, context or command. We allow formation of arbitrary finite sums of objects of the same kind, with the obvious meaning. Thus sum $+$ becomes an associative and commutative binary operation on terms, contexts and commands respectively, and object $\mathbf{0}$ is neutral. Similarly, we allow arbitrary finite convolution products of simple contexts, with unit $\mathbf{1}$. We can then extend our syntactic constructs by linearity:

$$\begin{array}{rclcrcl} \lambda x\,(\sum_{i=1}^n s_i) & = & \sum_{i=1}^n \lambda x\, s_i & \qquad & (\sum_{j=1}^p e_j) * (\sum_{k=1}^q f_k) & = & \sum_{j=1}^p \sum_{k=1}^q e_j * f_k \\ \mu\alpha\,(\sum_{l=1}^r c_l) & = & \sum_{l=1}^r \mu\alpha\, c_l & & \left\langle \sum_{i=1}^n s_i \,,\, \sum_{j=1}^p e_j \right\rangle & = & \sum_{i=1}^n \sum_{j=1}^p \langle s_i \,,\, e_j \rangle \\ S \cdot (\sum_{j=1}^p e_j) & = & \sum_{j=1}^p S \cdot e_j & & & & \end{array} \quad .$$

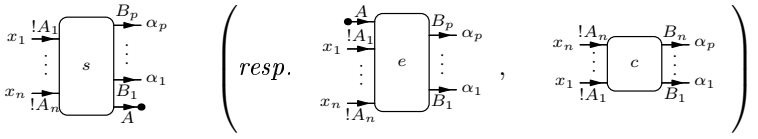
Notice that the cons $S \cdot E$ of term S and context E is *not* linear in the term: this is the analogue of application not being linear in the argument, in ordinary λ-calculus. This definition introduces some overlap of notations: e.g., $\lambda x\, s$ denotes both a simple term in our basic syntax, and the value of $\lambda x\,(s + \mathbf{0})$ in the above definition. This is however harmless since both writings denote the same term.

Hence the set of terms (resp. contexts, commands) is endowed with a structure of commutative monoid. The set of contexts is moreover endowed with a structure of commutative rig (i.e. a commutative ring, without the condition that every element admits an opposite), with addition $+$ and multiplication $*$. Also, λ- and μ-abstractions are linear, cons is linear in the context, and cut is bilinear. Thanks to the notations we have just introduced, the capture avoiding substitution of a term for a variable (resp. of a context for a name) in an object is defined as usual, by induction on objects.

5.2 Translation and reduction

Translation. Before we recall the reduction of convolution $\bar{\lambda}\mu$-calculus from [15], we make explicit the intended translation into PDN, first in a typed setting.

Definition 5.2 *The typing rules for the simple objects of convolution $\bar{\lambda}\mu$-calculus are given in Figure 6, together with their translation in PDN: to each derivation of $\Gamma \vdash s : A \mid \Delta$ (resp. $\Gamma \mid e : A \vdash \Delta$, $c : (\Gamma \vdash \Delta)$), where $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and $\Delta = \alpha_1 : B_1, \ldots, \alpha_p : B_p$, we associate an intutionistic sequential PDN*



For sums of simple objects, we moreover have the following three typing rules:

$$\frac{\{\Gamma \vdash s_i : A \mid \Delta\}_{i=1,\ldots,n}}{\Gamma \vdash \sum_{i=1}^n s_i : A \mid \Delta} \qquad \frac{\{\Gamma \mid e_i : A \vdash \Delta\}_{i=1,\ldots,n}}{\Gamma \mid \sum_{i=1}^n e_i : A \vdash \Delta} \qquad \frac{\{c_i : (\Gamma \vdash \Delta)\}_{i=1,\ldots,n}}{\sum_{i=1}^n c_i : (\Gamma \vdash \Delta)}$$

and the translation of a sum is the sum of the translations. In particular, the object $\mathbf{0}$ lives in all types and is translated by PDN $\mathbf{0}$ with corresponding interface.

From this definition, one easily derives a translation of pure convolution $\bar{\lambda}\mu$-calculus into pure typed sequential PDN. Like the translation of λ-calculus into linear logic, this translation of convolution $\bar{\lambda}\mu$-calculus is meant up-to a structural equivalence on PDN.

$$\frac{}{\Gamma, x : A \vdash x : A \mid \Delta}$$

$$\frac{\Gamma, x : A \vdash s : B \mid \Delta}{\Gamma \vdash \lambda x\, s : A \Rightarrow B \mid \Delta}$$

$$\frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Delta}$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha\, c : A \mid \Delta}$$

$$\frac{\Gamma \vdash s : A \mid \Delta \qquad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid s \cdot e : A \Rightarrow B \vdash \Delta}$$

$$\frac{\Gamma \vdash s : A \mid \Delta \qquad \Gamma \mid e : A \vdash \Delta}{\langle s\,, e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{}{\Gamma \mid \mathbf{1} : A \vdash \Delta}$$

$$\frac{\Gamma \mid \sigma : A \vdash \Delta \qquad \Gamma \mid e : A \vdash \Delta}{\Gamma \mid \sigma * e : A \Rightarrow B \vdash \Delta}$$

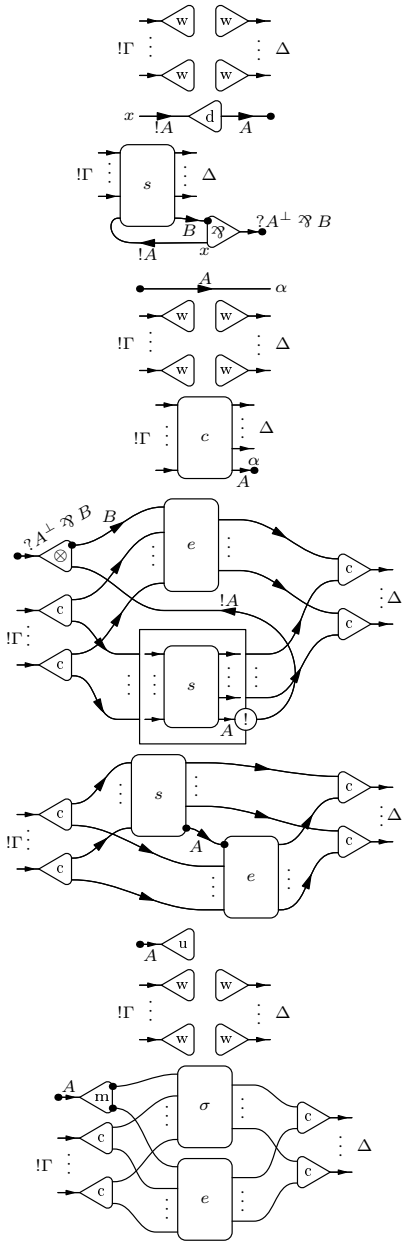


Figure 6: Typing of convolution $\bar{\lambda}\mu$-calculus objects and translation in PDN.

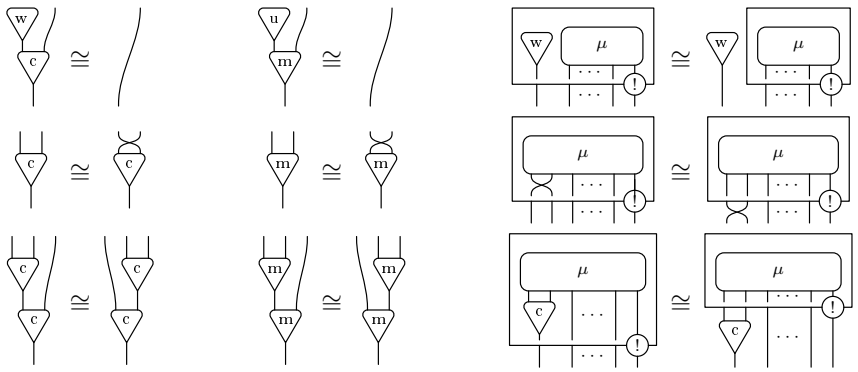


Figure 7: Structural equivalence of PDN

Definition 5.3 *We define the structural equivalence $\cong$ of PDN, as the reflexive, symmetric, transitive and contextual closure of the equations of Figure 7.*

Convolution Reduction. We call *contextual relation* any triple r of binary relations respectively on terms, contexts and commands, each also denoted by r, which are closed under the constructions of Definition 5.1: let $\bullet$ denote a single occurrence of simple object θ in object Θ, then $\theta \; r \; \Theta'$ implies $\Theta \; r \; \Theta\,[\Theta'/\bullet]$.

Definition 5.4 *Reduction $\to$ is the least contextual relation such that:*

$$\langle \mu\alpha\, c\,,\, e\rangle \;\to\; c\,[e/\alpha] \tag{1}$$

$$\langle \lambda x\, s\,,\, (S \cdot e) * f\rangle \;\to\; \langle \lambda y\, \mu\alpha\, \langle s\,[y + S/x]\,,\, \alpha * e\rangle\,,\, f\rangle \tag{2}$$

$$\langle \lambda x\, s\,,\, \mathbf{1}\rangle \;\to\; \langle s\,[\mathbf{0}/x]\,,\, \mathbf{1}\rangle \tag{3}$$

with y a fresh variable and α a fresh name in (2).

In [15], it is proved that this notion of reduction is confluent. It also is proved in [19] that the simply typed objects of convolution $\bar{\lambda}\mu$-calculus are all strongly normalizing: one adapts the proof by Polonowski in [20] for $\bar{\lambda}\mu\tilde{\mu}$-calculus. We now prove it corresponds with cut elimination in PDN.

Lemma 5.1 *The PDN represented in subfigures (a), (b), (c) and (d) of Figure 8 reduce to PDN structurally equal to the translation of respectively $c\,[e/\alpha]$, $c\,[\mathbf{0}/x]$, $c\,[y + z/x]$ and $c\,[T/x]$, where: $c : (\Gamma \vdash \alpha : A, \Delta)$ and $\Gamma \mid e : A \vdash \Delta$ in case (a); $c : (\Gamma, x : A \vdash \Delta)$ in cases (b), (c) and (d); y and z are distinct fresh variables in case (c); $\Gamma \vdash T : A \mid \Delta$ in case (d).*

Theorem 5.5 *The cut elimination procedure of PDN* up-to structural equivalence *simulates the reduction of convolution $\bar{\lambda}\mu$-calculus.*

Proof By context closure, it is sufficient to consider the case of redexes. For $\langle \mu\alpha\, c\,,\, e\rangle \to c\,[e/\alpha]$, case (a) of Lemma 5.1 applies directly. For $\langle \lambda x\, s\,,\, \mathbf{1}\rangle \to \langle s\,[\mathbf{0}/x]\,,\, \mathbf{1}\rangle$, reduce the cut $\langle \mathrm{u}\,,\, ⅋\rangle$, then apply case (b). For $\langle \lambda x\, s\,,\, (S \cdot e) * f\rangle \to \langle \lambda y\, \mu\alpha\, \langle s\,[T + y/x]\,,\, e * \alpha\rangle\,,\, f\rangle$, reduce the cut $\langle \mathrm{m}\,,\, ⅋\rangle$ followed by $\langle \otimes\,,\, ⅋\rangle$; then apply case (c) with fresh variables y and z, followed by case (d) to obtain the PDN associated with $\langle \lambda y\, \mu\alpha\, \langle s\,[y + z/x]\,[T/z]\,,\, e * \alpha\rangle\,,\, f\rangle$. □

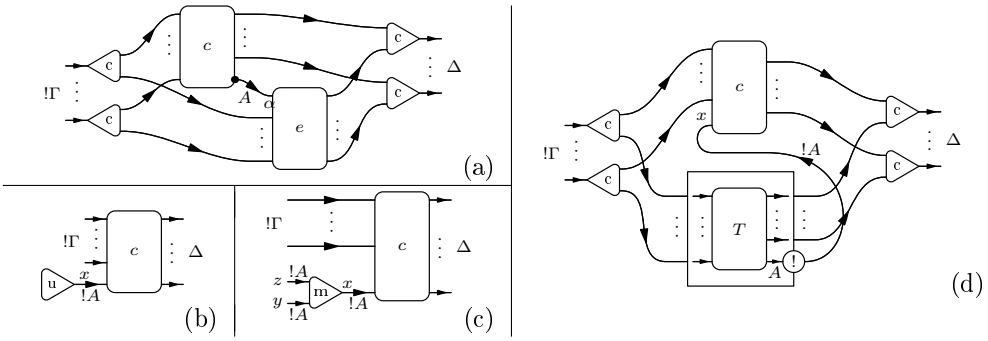


Figure 8: PDN simulating the substitution operations of the convolution $\bar{\lambda}\mu$-calculus

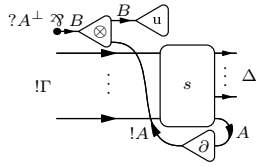
Recall from [15] that the relational model described in section 3 provides a denotational semantics of convolution $\bar{\lambda}\mu$-calculus. The reader will easily check that this semantics can be decomposed as the translation of pure convolution $\bar{\lambda}\mu$-calculus in PDN, followed by the semantics of pure typed PDN in $\mathcal{D}$.

5.3 Towards a differential $\bar{\lambda}\mu$-calculus

Notice that $\langle \lambda x\, s\,,\, (T \cdot e) * (U \cdot f)\rangle \to^* \langle s\,[T + U/x]\,,\, e * f\rangle$. In some sense, context $(T \cdot e) * (U \cdot f)$ simulates $(T + U) \cdot (e * f)$, as is reflected by the fact that these contexts are identified in the relational semantics (see [15]). This suggests to introduce notations $S^! = S \cdot \mathbf{1}$ and $\uparrow e = \mathbf{0} \cdot e$ so that $S^! * \uparrow e$ simulates $S \cdot e$: from now on, we consider a syntax where stacks are restricted to those two shapes. Reduction rule (2) boils down to the elementary rules $\langle \lambda x\, s\,,\, T^! * f\rangle \to \langle \lambda x\, s\,[x + T/x]\,,\, f\rangle$ and $\langle \lambda x\, s\,,\, \uparrow e * f\rangle \to \langle \lambda x\, \mu\alpha\, \langle s\,,\, e * \alpha\rangle\,,\, f\rangle$. In this new syntax, all causality information in stacks is lost, and one only retains a minimal form of sequentiality: contexts of type $A \Rightarrow B$ become bags of arguments of type A and future contexts of type B.

So far, we only focused on the computational content of costructural rules. It turns out the breaking down of contexts we have just performed makes the introduction of differentiation in the sense of [2] very natural in this setting. Introduce a new stack construction $[s]$ with typing rule and associated net:

$$\frac{\Gamma \vdash s : A \mid \Delta}{\Gamma \mid [s] : A \Rightarrow B \vdash \Delta} \quad \text{and} \quad$$



.

One then defines a linear variant of substitution $\frac{\partial\,\theta}{\partial\,x} \cdot t$ following [2] so that $\langle \lambda x\, s\,,\, [t] * e\rangle \to \langle \lambda x\, \frac{\partial\, s}{\partial\, x} \cdot t\,,\, e\rangle$, in adequation with cut-elimination in PDN.

One has to pay attention to the fact that case (a) of Lemma 5.1 no longer holds: stacks containing linear arguments can no longer be duplicated nor erased freely. Hence one has to introduce a notion of *named cut*: let θ be any simple object with free name α of type A and e be a simple context of type A, we must define object $\langle \theta\,,\, e\rangle_\alpha$ so that the PDN interpretation of $\langle \mu\alpha\, c\,,\, e\rangle$ reduces (up to $\cong$) to that of $\langle \theta\,,\, e\rangle_\alpha$. This can be done by induction on e, inspecting the possible PDN reductions, and involves a construction similar to the named

derivative of [13]. One then proves that the reduction relation of the obtained pure calculus is confluent and appropriately simulated by cut-elimination in PDN; simply typed objects are moreover strongly normalizing. These results are detailed in [19, Chapter 8]. The obtained system can be seen as a classical sequent calculus version of differential λ-calculus.

Current investigations include: establishing a precise relationship between this calculus and Boudol's resource λ-calculus as studied in [5]; studying the "equation" $S^! = \sum_{i=1}^{n} \frac{1}{n!}[S]^n$, w.r.t. both denotational and operational semantics, following [21, 22]; relating the convolution product with parallel composition as known in concurrency theory; more generally, revealing the expressivity of differential $\bar{\lambda}\mu$-calculus w.r.t. concurrent computing, following recent advances [23] on simulating Milner's π-calculus [24] in DN.

References

[1] Ehrhard, T., Regnier, L.: Differential interaction nets. Theor. Comput. Sci. **364** (2006) 166–195

[2] Ehrhard, T., Regnier, L.: The differential lambda-calculus. Theor. Comput. Sci. **309** (2003) 1–41

[3] Girard, J.Y.: Normal functors, power series and lambda-calculus. Annals of Pure and Applied Logic **37**(2) (1988) 129–177

[4] Ehrhard, T.: Finiteness spaces. Math. Struct. Comput. Sci. **15**(4) (2005) 615–646

[5] Tranquilli, P.: Intuitionistic differential nets and lambda-calculus. To appear in Theor. Comput. Sci (2008)

[6] Andreoli, J.M.: Logic programming with focusing proofs in linear logic. Journal of Logic and Computation **2** (1992) 297–347

[7] Girard, J.Y.: A new constructive logic: Classical Logic. Mathematical Structures in Computer Science **1**(3) (1991) 255–296

[8] Laurent, O.: Étude de la polarisation en logique. PhD thesis, Université Aix-Marseille 2 (2002)

[9] Danos, V.: Une application de la logique linéaire à l'étude des processus de normalisation (principalement du λ-calcul). PhD thesis, Université Paris 7 (1990)

[10] Regnier, L.: Lambda-calcul et réseaux. PhD thesis, Université Paris 7 (1992)

[11] Laurent, O., Regnier, L.: About translations of classical logic into polarized linear logic. In: Proceedings of the eighteenth annual IEEE symposium on Logic In Computer Science, IEEE Computer Society Press (June 2003) 11–20

[12] Bierman, G.M.: What is a categorical model of intuitionistic linear logic? In Dezani, M., ed.: Proceedings of Conference on Typed lambda calculus and Applications. Volume 902 of Lecture Notes in Computer Science., Springer-Verlag (1995)

[13] Vaux, L.: The differential $\lambda\mu$-calculus. Theor. Comput. Sci. **379**(1-2) (2007) 166–209

[14] Bucciarelli, A., Ehrhard, T., Manzonetto, G.: Not enough points is enough. In: Computer Science Logic. Volume 4646 of Lecture Notes in Computer Science., Springer Berlin (2007) 298–312

[15] Vaux, L.: Convolution $\bar{\lambda}\mu$-calculus. In Rocca, S.R.D., ed.: TLCA. Volume 4583 of Lecture Notes in Computer Science., Springer (2007) 381–395

[16] Mazza, D.: Interaction Nets: Semantics and Concurrent Extensions. PhD thesis, Université Aix–Marseille 2, Università degli Studi Roma Tre (2006)

[17] Lafont, Y.: From proof nets to interaction nets. In Girard, J.Y., Lafont, Y., Regnier, L., eds.: Advances in Linear Logic, Cambridge University Press (1995) 225–247

[18] Schwartz, L.: Théorie des distributions. Hermann (1966)

[19] Vaux, L.: λ-calcul différentiel et logique classique: interactions calculatoires. PhD thesis, Université Aix-Marseille 2 (2007)

[20] Polonowski, E.: Substitutions explicites, logique et normalisation. PhD thesis, Université Paris 7 (2004)

[21] Ehrhard, T., Regnier, L.: Uniformity and the Taylor expansion of ordinary lambda-terms. Theor. Comput. Sci. **403** (2008) 347–372

[22] Ehrhard, T., Regnier, L.: Böhm trees, Krivine's machine and the Taylor expansion of lambda-terms. In Beckmann, A., Berger, U., Löwe, B., Tucker, J.V., eds.: CiE. Volume 3988 of Lecture Notes in Computer Science., Springer (2006) 186–197

[23] Ehrhard, T., Laurent, O.: Interpreting a finitary pi-calculus in differential interaction nets. In Caires, L., Vasconcelos, V.T., eds.: Concurrency Theory (CONCUR '07). Volume 4703 of Lecture Notes in Computer Science., Springer (September 2007) 333–348

[24] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. **100**(1) (1992) 1–40