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

Lionel Vaux
Institut de Mathématiques de Luminy, Marseille, France
vaux@iml.univ-mrs.fr
http://iml.univ-mrs.fr/~vaux

May 30, 2007

We define an extension of Herbelin's $\bar{\lambda}\mu$-calculus, introducing a product operation on contexts (in the sense of lists of arguments, or stacks in environment machines), similar to the convolution product of distributions. This is the computational couterpart of some new semantical constructions, extending models of Ehrhard-Regnier's differential interaction nets, along the lines of Laurent's polarization of linear logic. We demonstrate this correspondence by providing this calculus with a denotational semantics inside a lambda-model in the category of sets and relations.

1 Introduction

Herbelin's $\bar{\lambda}\mu$-calculus [Her95] transposes the Curry-Howard correspondence between classical natural deduction and $\lambda\mu$-calculus to the setting of classical sequent calculus LK (in fact one of its deterministic versions: LKT [DJS95]). In particular, the notion of application, corresponding to the *modus ponens* of natural deduction, is replaced with the notion of cut between a term and a context. More precisely, $\bar{\lambda}\mu$-calculus involves three syntactic categories — terms, contexts and commands — given by the following grammar:

$$\begin{array}{rcll} s,t & ::= & x \mid \lambda x\, s \mid \mu\alpha\, c & \text{(terms)} \\ e,f & ::= & \alpha \mid s \cdot e & \text{(contexts)} \\ c & ::= & \langle s\,,e\rangle & \text{(commands)}\ . \end{array}$$

Reduction is defined by the following two basic rules:

$$\langle \lambda x\, s\,, t\cdot e\rangle \to \langle s\,[t/x]\,, e\rangle \quad \text{and} \quad \langle \mu\alpha\, c\,, e\rangle \to c\,[e/\alpha] \ .$$

In the present paper, we introduce an extension of $\bar{\lambda}\mu$-calculus, featuring a binary operation $*$ on contexts (and the corresponding context unit $\mathbf{1}$), which bears similarities with the convolution product of distributions. For that purpose, we further endow the

set of terms with a structure of commutative monoid, with addition + and neutral $\mathbf{0}$, and give the following three reduction rules:

$$\begin{array}{rcl} \langle \lambda x\, s\,, (t \cdot e) * f \rangle & \to & \langle \lambda x\, \mu\alpha\, \langle s\,[t + x/x]\,, e * \alpha \rangle\,, f \rangle \\ \langle \lambda x\, s\,, \mathbf{1} \rangle & \to & \langle s\,[\mathbf{0}/x]\,, \mathbf{1} \rangle \\ \langle \mu\alpha\, c\,, e \rangle & \to & c\,[e/\alpha] \quad . \end{array}$$

The reader may check that the reduction rules of usual $\bar{\lambda}\mu$-calculus can be simulated in this new setting.

Outline of the Paper. In the remaining of this introduction, we briefly review notions and ideas that led to the definition of convolution $\bar{\lambda}\mu$-calculus: it is the pure calculus associated with an extension of Ehrhard-Regnier's differential nets [ER05], along the lines of Laurent's polarization of linear logic proof nets [Lau02]. In section 2 we introduce the objects of convolution $\bar{\lambda}\mu$-calculus, and the associated notion of reduction, for which we prove the Church-Rosser property. In section 3 we define a reflexive object $\mathcal{D}$ in the category of sets and relations, following [BE04]. Then we introduce a type-system, in which types are elements of $\mathcal{D}$, along the lines of Carvalho's system R [dC06]. We conclude by proving that terms identified by reduction have exactly the same types.

1.1 Classical Logic and Co-structural Rules

Denotational semantics of linear logic gives rise to models of simply typed λ-calculus, through the Curry-Howard isomorphism and well known translations from intuitionistic logic into linear logic. This relationship may be explicited in the syntax by encodings of typed λ-calculus into linear logic proof nets, in which β-reduction of λ-terms is accurately simulated by cut-elimination in proof nets. In fact, one may also establish such a correspondence in an untyped setting: pure λ-terms are succesfully encoded into the weakly typed nets of [Reg92].

An analoguous relationship may be observed in a setting related with classical logic rather than intuitionistic logic. In [Lau02], Olivier Laurent introduced polarized linear logic and polarized proof nets. Polarized linear logic is linear logic where all formulas are polarized, and weakening and contraction are allowed on every negative formula; also, promotion is allowed on any sequent formed only of negative formulas. In [Lau03], Laurent showed how to encode Parigot's $\lambda\mu$-calculus [Par92] into polarized proof nets. Since $\lambda\mu$-calculus lifts the Curry-Howard correspondence from intuitionistic logic to classical logic, this encoding is the counterpart of a translation from classical natural deduction into polarized linear logic. A thorough semantical investigation on the nature of this translation may be found in [LR03].

It also happens that original models of linear logic lead to novel extensions of λ-calculus, using the Curry-Howard correspondence as a tool to draw semantical properties back into the syntax. In [Ehr01] and [Ehr05], Ehrhard introduced models of linear logic in which formulas are interpreted by particular vector spaces, and proofs by linear maps between these spaces. Moreover, morphisms with type $!A \multimap B$ correspond to analytic

functions between A and B. This not only provided a semantics of typed λ-calculus in which λ-terms are interpreted by smooth functions between vector spaces in a very natural way, but also led to the introduction of differential λ-calculus by Ehrhard and Regnier in [ER03].

Differential Nets. Here we briefly outline some features of the differential interaction nets from [ER05], which may be considered as a syntactic presentation of the models from [Ehr01] and [Ehr05]. For the sake of simplicity, we use a weak typing scheme: using usual linear logic connectors and modalities, introduce type o such that $o = !o \multimap o$; we will use types o, $i = o^\perp$, $!o$ and $?i = (!o)^\perp$, to type the wires of interaction nets. We do not consider criteria for well-formedness of nets; we only focus on local, weakly typed reduction rules.

Differential interaction nets extend the interaction nets for multiplicative exponential linear logic from [Laf95] as follows: besides multiplicative cells

par:  and tensor: 

and structural cells

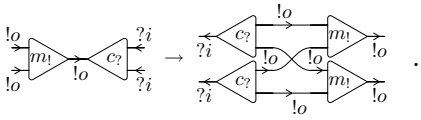
dereliction:  , contraction:  and weakening:  ,

come costructural cells

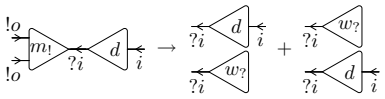
derivative:  , cocontraction:  and coweakening:  .

Cells $m_!$ and $u_!$ have the same geometry as tensor and tensor unit respectively: $m_!$ connects two nets together, and $u_!$ is a net by itself.

Recall that in the formalism of interaction nets, typing depends on the orientation of wires: if a wire has type A in one orientation, it has type $A^\perp$ in the reverse orientation. Also, recall that each cell has exactly one principal port (this we put on the point of our triangular cells) together with any number of auxiliary ports. A redex consists of two cells connected by their respective principal ports, in accordance with typing. Among new reduction rules introduced in differential nets, interaction between cells $c_?$ and $w_?$ on the one hand, and $m_!$ and $u_!$ on the other hand, endow exponential types with a structure of bialgebra, mainly characterized by the following interaction rule:

 .

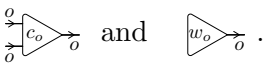
Also, dereliction d interacts with $m_!$ as follows:

 .

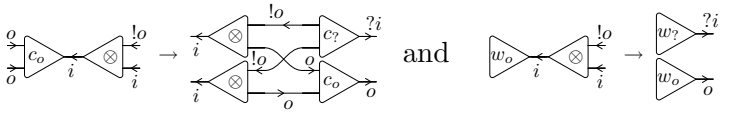
The idea is that d requires one copy of an argument from its principal port, which it feeds to its auxiliary port; this argument is taken nondeterministically from either auxiliary port of $m_!$, hence the sum. The redex between d and $w_!$ reduces to the 0 net (which is the neutral element of sum of nets) following the same intuition. There are of course symmetric rules for interaction between derivative ∂, and $c_?$ and $w_?$, that we do not explicit here.

An extensive discussion of the intuitions behind differential reduction rules may be found in [ER05] and the relationship between sum and nondeterminism is developped in the introduction of [ER03]. Although this is not done in [ER05], one may also introduce promotion boxes in differential interaction nets, along the lines of [Laf95], and provide appropriate reduction rules: this allows to encode differential λ-calculus.

Polarized Nets. Polarized nets are another extension of linear logic nets. Again, we only outline a weakly typed version of the polarized proof nets of [Lau03], which we present in an interaction net flavor. The main feature of polarized nets is that contraction and weakening are generalized to type o:



This accounts for structural rules on output types, which is a characteristic of classical logic. Generalized structural rules give rise to a new redex between tensor and contraction c_o (or weakening w_o) cells:



Polarized proofs nets are well suited to encode classical extensions of λ-calculus: see [Lau03] for an encoding of $\lambda\mu$-calculus and [Lau02, Section 12.2] for an encoding of both deterministic variants of $\bar{\lambda}\mu\tilde{\mu}$-calculus.

Differential Structures and Classical Logic. In [Vau07a], the author introduced the differential $\lambda\mu$-calculus, as an attempt to uncover possible interactions between differential structures and classical logic, in a purely computational setting. The result is a pure calculus which consistently extends both differential λ-calculus and $\lambda\mu$-calculus.

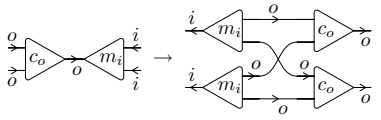
Another possible path for studying how differential and classical constructs interact with each other lies at the level of interaction nets. One may come up with a notion of differential polarized nets, with cells those of differential nets and polarized nets altogether. Then it is easily checked that the union of the reduction rules for differential nets and for polarized nets address all possible redexes.

We do not detail further this construction, but one may verify that differential $\lambda\mu$-calculus enjoys a natural encoding into these differential polarized nets. Hence, although differential $\lambda\mu$-calculus introduces a new reduction rule, which was not present in $\lambda\mu$-calculus nor in differential λ-calculus (namely that associated with the derivative of

a μ-abstraction), one may claim that it is only a side-effect of the sequentiality of λ-calculus. Indeed, in the more parallel syntax of interaction nets, differential cells on the one hand, and generalized structural rules on the other hand, do not interact with each other.

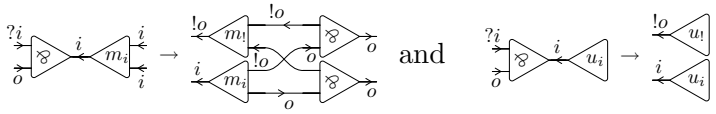
A Convolution Product on Contexts. The convolution $\bar{\lambda}\mu$-calculus defined in this paper, is the pure calculus associated with the following other variant of polarized nets: together with the cells of polarized nets, introduce the abovementioned costructural cells $m_!$ and $u_!$, and also generalized versions of these,

i.e. cocontraction and coweakening on input type i. New redexes arise and we give the following reduction rules: m_i and u_i interact with c_o and w_o the same way as $m_!$ and $u_!$ interact with $c_?$ and $w_?$,

in particular 

;

also, m_i duplicates $\parr$ and u_i erases $\parr$, the same way as c_o and w_o act on $\otimes$:



.

Again, we do not detail this system further: this is still the subject of ongoing work in [Vau07b]. Rather, we explicit how convolution $\bar{\lambda}\mu$-calculus stems from it. In the translation of $\lambda\mu$-calculus into polarized nets, the type of dangling wires (oriented outwards) is only o or $?i$. Type i only occurs in the cuts involved in the translation of application. This suggests that the computational counterpart of costructural cells on type i may be more fruitfully studied in the setting of Herbelin's $\bar{\lambda}\mu$-calculus, where cuts appear explicitly.

One may derive a translation of $\bar{\lambda}\mu$-calculus into polarized nets from that of $\bar{\lambda}\mu\tilde{\mu}_T$-calculus given in [Lau02], in which i types the active wire of the translation of *contexts*. The counterpart of m_i and u_i is then an associative and commutative operation on contexts that we denote by $*$, and its unit $\mathbf{1}$. It is argued in [Ehr01, Section 5.4], that $m_!$ acts as a convolution product on $!o$, with properties similar to those of convolution of distributions; in that analogy, $u_!$ corresponds to the Dirac mass at 0 (see also [Ehr05, Section 3], in paragraph *Algebraic structure*). Since the behaviour of m_i and u_i on type i mimics that of $m_!$ and $u_!$ on type $!o$, we may call $*$ *convolution product on contexts*. We will show later that, although they live in different formalisms, $*$ actually shares a distinctive feature with the convolution product of distributions, which further enforces this designation.

From the reduction rules of nets we have outlined, one derives that β-reduction may be generalized so that:

$$\left\langle \lambda x\, s\, , (t \cdot e) * (t' \cdot e') \right\rangle \to \left\langle s\left[t + t'/x\right], e * e' \right\rangle \quad .$$

Recall that nondeterministic choice provides a possible computational interpretation of sum, as described in the introduction of [ER03]. Convolution product of contexts may then be interpreted as a nondeterministic intertwining of lists of arguments.

The reduction step given above, however, only amounts to usual reduction in $\bar{\lambda}\mu$-calculus, together with the identity $(t \cdot e) * (t' \cdot e') = (t + t') \cdot (e * e')$ which is semantically valid (see Remark 15). Moreover, it involves synchronisation between contexts: both must have a term at top-level, for a reduction to occur. In this paper, we use a finer grained notion of reduction, the basic rules of which were given in the beginning of this introduction: this matches cut elimination between $\parr$ and costructural cells more closely. Also, that notion will enable us to demonstrate the link with convolution of distributions in Remark 5.

Relational Semantics. In order to underline the correspondence between convolution $\bar{\lambda}\mu$-calculus and generalized costructural rules on i, we provide it with a denotational semantics in the category of sets and relations. In the standard relational model of linear logic, formulas are interpreted as sets and proofs as relations between these sets: see, e.g., [Ehr05, Appendix A] for precise definitions. In particular, structural rules on type A correspond to relations $d_A \subseteq\, !A \multimap A = \mathcal{M}_{\text{fin}}(A) \times A$, $c_A \subseteq\, !A \multimap (!A \otimes\, !A) = \mathcal{M}_{\text{fin}}(A) \times (\mathcal{M}_{\text{fin}}(A) \times \mathcal{M}_{\text{fin}}(A))$ and $w_A \subseteq\, !A \multimap 1 = \mathcal{M}_{\text{fin}}(A) \times 1$, where $\mathcal{M}_{\text{fin}}(A)$ denotes the set of all finite multisets of elements of A and 1 is a singleton set. One interesting feature of the relational model is that it identifies dual connectors in linear logic. In particular, if $\varphi \subseteq A \multimap B$ in the relational model, then $\varphi^{\perp} \subseteq B \multimap A$ where $\varphi^{\perp} = \{(b, a);\ (a, b) \in \varphi\}$. By setting $\partial_A = d_A^{\perp}$, $m_A = c_A^{\perp}$, and $u_A = w_A^{\perp}$, it turns out that we obtain a model of differential interaction nets. This suggests that we may derive a denotational semantics of convolution $\bar{\lambda}\mu$-calculus from a model of $\lambda\mu$-calculus in the category of sets and relations.

The model we use was introduced by Bucciarelli and Ehrhard in [BE04], as an extensional lambda-model in the category of sets and relations. It is naturally endowed with a monoid structure, which is suitable to provide a denotational semantics of $\lambda\mu$-calculus along the lines of [LR03]: monoid operation and unit model structural rules on o. We also use those monoid laws to handle the denotational semantics of convolution product: the same as for costructural rules, this amounts to reverse the direction of the corresponding relation.

1.2 Related Work

A system of intersection and union types for the $\bar{\lambda}\mu$-calculus is presented [DGL05]. This system bears some similarity with the type system we present in section 3: this is underlined by the fact that all weakly normalizing terms are typable. It comes as no

surprise, since our system is derived from Carvalho's system R, which is related to a system of intersection types for λ-calculus.

One important outcome of [dC06] and our paper is that they provide the aforementioned type systems with a strong grounding into well known denotational semantics of linear logic and its variants.

2 Syntax

In this section, we introduce the syntax of convolution $\bar{\lambda}\mu$-calculus. Like ordinary $\bar{\lambda}\mu$-calculus, it involves three distinct syntactic categories: terms, contexts and commands. We introduce convolution product $*$ as a commutative and associative binary operation on contexts, with unit $\mathbf{1}$.

Similarly to what is done in [ER03] and [Vau07a], each category of objects is endowed with a structure of commutative monoid, and we denote by $+$ and $\mathbf{0}$ the corresponding operation and neutral element. Moreover, all but one syntactic construct are linear, *i.e.* they commute to sums: in particular, $*$ distributes over $+$. In order to implement these high-level, metatheoretical requirements, we first define a basic syntax with a simple equality, then provide extended notations.

Remark 1. Since we only form sums (without coefficients) it is quite clear that our constructions are well defined, and that nothing tricky is hiding behind equality of terms. Recall from [Vau06, Section 4] that the introduction of linear combinations of terms (rather than just sums) may break normalization properties, or even trivialize β-equality [Vau06, Section 2.6], depending on the structure of the set of coefficients.

2.1 Morphology

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

Definition 2. Define sets $\mathcal{T}$ of simple terms and $\mathcal{T}^+$ of terms, set $\mathcal{S}$ of stacks, sets $\mathcal{E}$ of simple contexts and $\mathcal{E}^+$ of contexts, and sets $\mathcal{C}$ of simple commands and $\mathcal{C}^+$ of commands, by the following grammar:

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

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. Equality of terms (resp. commands, contexts) is then given by permutativity of sum and product, together with α-equivalence.

Notations. We call simple object any simple term, simple context or simple command, and object any term, context or command. We may use greek letter θ to denote a simple object and capital Θ to denote an object. In general, if simple object θ and object Θ appear in the same sentence, it should be clear they are of the same kind: Θ is a term, context or command, if θ is a simple term, a simple context or a simple command respectively.

If $\theta_1, \dots, \theta_n$ are simple objects and Θ an object, all of the same kind, we write $\theta_1 + \dots + \theta_n + \Theta$ for $\theta_1 + (\dots + (\theta_n + \Theta)\dots)$. If θ is a simple object, we may also denote by θ the corresponding object $\theta + \mathbf{0}$. Hence, all object Θ may be written $\Theta = \theta_1 + \dots + \theta_n$ or even $\Theta = \sum_{i=1}^{n} \theta_i$. Assume $\Theta = \theta_1 + \dots + \theta_n$ and $\Theta' = \theta'_1 + \dots + \theta'_p$: we write $\Theta + \Theta'$ for $\theta_1 + \dots + \theta_n + \theta'_1 + \dots + \theta'_p$. Up to these conventions, sum $+$ becomes an associative and commutative binary operation on terms, contexts and commands respectively, and object $\mathbf{0}$ is neutral.

Similarly, we identify any stack σ with the simple context $\sigma * \mathbf{1} \in \mathcal{E}$: then we may write any simple context e as $e = \sigma_1 * \dots * \sigma_n$ where the stacks σ_i are names or of shape $S \cdot e'$. With notations similar to those we used for sum, we consider $*$ as an associative and commutative binary operation on simple contexts, with unit $\mathbf{1}$.

Now we extend our syntactic constructs by linearity in order to be able to write $\lambda x\, S$, $\mu\alpha\, C$, $S \cdot E$, $E * F$ and $\langle S\,, E\rangle$ for all $S \in \mathcal{T}^+$, $E, F \in \mathcal{E}^+$ and $C \in \mathcal{C}^+$.

Definition 3. Assume $s_1, \dots, s_n \in \mathcal{T}$, $e_1, \dots, e_p, f_1, \dots, f_q \in \mathcal{E}$, $c_1, \dots, c_r \in \mathcal{C}$ and $S \in \mathcal{T}^+$. Then we write:

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

Notice that the cons of a term and a context is *not* linear in the term: this is the analogue of application not being linear in the argument, in usual 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.

Up to the notations we have just introduced, 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. Then 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:

$$\begin{array}{rcl@{\qquad}rcl}
y\,[T/x] & = & \left\{\begin{array}{ll} T & \text{if } x = y \\ y & \text{otherwise}\end{array}\right. & y\,[E/\alpha] & = & y \\
(\lambda y\, s)\,[T/x] & = & \lambda y\,(s\,[T/x]) & (\lambda y\, s)\,[E/\alpha] & = & \lambda y\,(s\,[E/\alpha]) \\
(\mu\beta\, c)\,[T/x] & = & \mu\beta\,(c\,[T/x]) & (\mu\beta\, c)\,[E/\alpha] & = & \mu\beta\,(c\,[E/\alpha]) \\
\beta\,[T/x] & = & \beta & \beta\,[E/\alpha] & = & \left\{\begin{array}{ll} E & \text{if } \alpha = \beta \\ \beta & \text{otherwise}\end{array}\right. \\
(S \cdot e)\,[T/x] & = & (S\,[T/x]) \cdot (e\,[T/x]) & (S \cdot e)\,[E/\alpha] & = & (S\,[E/\alpha]) \cdot (e\,[E/\alpha]) \\
1\,[T/x] & = & 1 & 1\,[E/\alpha] & = & 1 \\
(\sigma * e)\,[T/x] & = & (\sigma\,[T/x]) * (e\,[T/x]) & (\sigma * e)\,[E/\alpha] & = & (\sigma\,[E/\alpha]) * (e\,[E/\alpha]) \\
\langle s\,, e\rangle\,[T/x] & = & \langle s\,[T/x]\,, e\,[T/x]\rangle & \langle s\,, e\rangle\,[E/\alpha] & = & \langle s\,[E/\alpha]\,, e\,[E/\alpha]\rangle \\
\mathbf{0}\,[T/x] & = & \mathbf{0} & \mathbf{0}\,[E/\alpha] & = & \mathbf{0} \\
(\theta + \Theta)\,[T/x] & = & \theta\,[T/x] + \Theta\,[T/x] & (\theta + \Theta)\,[E/\alpha] & = & \theta\,[E/\alpha] + \Theta\,[E/\alpha]
\end{array}$$

assuming usual conditions to avoid variable and name capture.

2.2 Reduction

Convolution Reduction. We call *simply contextual relation* any triplet r of binary relations respectively on terms, contexts and commands, all denoted r, and such that:

- if s r S' then $\lambda x\, s$ r $\lambda x\, S'$ and $\langle s\,, e\rangle$ r $\langle S'\,, e\rangle$;
- if e r E' then $s \cdot e$ r $s \cdot E'$, $\langle s\,, e\rangle$ r $\langle s\,, E'\rangle$ and $e * f$ r $E' * f$;
- if c r C' then $\mu\alpha\, c$ r $\mu\alpha\, C'$;
- if S r S' then $S \cdot e$ r $S' \cdot e$;
- and if θ_0 r Θ'_0 then for all Θ_1, $\theta_0 + \Theta_1$ r $\Theta'_0 + \Theta_1$.

Definition 4. Reduction $\to_\beta$ is the least simply contextual relation such that:

$$\begin{array}{rcll}
\langle \mu\alpha\, c\,, e\rangle & \to_\beta & c\,[e/\alpha] & (1) \\
\langle \lambda x\, s\,, (S \cdot e) * f\rangle & \to_\beta & \langle \lambda y\, \mu\alpha\, \langle s\,[y + S/x]\,, \alpha * e\rangle\,, f\rangle & (2) \\
\langle \lambda x\, s\,, \mathbf{1}\rangle & \to_\beta & \langle s\,[\mathbf{0}/x]\,, \mathbf{1}\rangle & (3)
\end{array}$$

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

Notice that $\langle \lambda x\, s\,, S \cdot e\rangle \to_\beta^* \langle s\,[S/x]\,, e\rangle$ and, more interestingly,

$$\left\langle \lambda x\, s\,, (S \cdot e) * (S' \cdot e')\right\rangle \to_\beta^* \left\langle s\left[S + S'/x\right], e * e'\right\rangle$$

where $\to_\beta^*$ denotes the reflexive and transitive closure of $\to_\beta$. This enlightens the fact that $\to_\beta$ is a refined version of both usual reduction of $\bar{\lambda}\mu$-calculus and the coarser notion of convolution reduction we first derived from cut elimination in the introduction.

Conversely, $\to_\beta$ may be simulated by that coarse reduction, up-to the following generalization of η-expansion on commands: recalling that the analogue of η-expansion in $\bar{\lambda}\mu$-calculus is $s \leftarrow_\eta \lambda x\, \mu\alpha\, \langle s\,, x \cdot \alpha\rangle$ we set

$$\left\langle s\,, e * e' \right\rangle \leftarrow_{\eta'} \left\langle \lambda x\, \mu\alpha\, \langle s\,, e * (x \cdot \alpha)\rangle\,, e' \right\rangle \quad .$$

This can be thought of as η-expansion w.r.t. only one component of a product. If e' actually holds an argument at top-level, *i.e.* $e' = S \cdot f$, we can get back:

$$\langle \lambda x\, \mu\alpha\, \langle s\,, e * (x \cdot \alpha)\rangle\,, S \cdot f\rangle \to^*_\beta \langle s\,, e * (S \cdot f)\rangle = \left\langle s\,, e * e' \right\rangle$$

which validates $\leftarrow_{\eta'}$ as a notion of η-expansion.

Remark 5. Recall (e.g., from [Sch66]) that the definition of the convolution product of distributions is as follows: if e and f are distributions with compact domains and φ is a test function, then $e * f$ is such that

$$\langle \lambda z\, \varphi(z)\,, e * f\rangle = \langle \lambda y\, \langle \lambda x\, \varphi(x+y)\,, e\rangle\,, f\rangle \quad .$$

Analoguously, one can check that the following two commands

$$\langle \lambda z\, s\,, (S \cdot e) * (T \cdot f)\rangle \text{ and } \langle \lambda y\, \mu\beta \langle \lambda x\, \mu\alpha \langle s\,[x + y/z]\,, \alpha * \beta\rangle\,, S \cdot e\rangle\,, T \cdot f\rangle$$

are identified by reduction: both reduce to $\langle s\,[S + T/z]\,, e * f\rangle$. The apparent complexity of that last identity has two main causes.

First, in the formalism of distributions and test functions, φ is supposed to be a function with scalar values. The type corresponding to scalars is that of commands, but in $\bar{\lambda}\mu$-calculus, as in λ-calculus, functions *and* values are represented by terms. Hence the μ-abstractions and the innermost cut: these handle the possible remaining arguments. Second, functions are in general considered extensionally. Expansion $\leftarrow_{\eta'}$ may be used to introduce sufficient extensionality:

$$\begin{array}{rcl} \langle \lambda z\, s\,, e * f\rangle & \leftarrow_{\eta'} & \langle \lambda y\, \mu\beta\, \langle \lambda z\, s\,, e * (y \cdot \beta)\rangle\,, f\rangle \\ & \to_\beta & \langle \lambda y\, \mu\beta \langle \lambda x\, \mu\alpha \langle s\,[x + y/z]\,, \alpha * \beta\rangle\,, e\rangle\,, f\rangle \quad . \end{array}$$

Confluence. We prove confluence of reduction using usual Tait-Martin-Löf technique: introduce a parallel extension of one-step reduction, and prove this has the diamond property.

A binary relation r on commutative monoid $\mathcal{A}$ is said to be *linear* if: for all $a_1, \ldots,$ $a_n, b_1, \ldots, b_n \in \mathcal{A}$, if a_i r b_i holds for all i, then $\sum_{i=1}^n a_i$ r $\sum_{i=1}^n b_i$ also holds (in particular 0 r 0). Notice that $\to_\beta$ is not linear: $\mathbf{0} \not\to_\beta \mathbf{0}$. We call *contextual relation* any triplet r of binary relations respectively on terms, contexts and commands, all denoted r, such that each of them is reflexive and linear, and if S r S', E r E', F r F' and C r C', then $\lambda x\, S$ r $\lambda x\, S'$, $\mu\alpha\, C$ r $\mu\alpha\, C'$, $S \cdot E$ r $S' \cdot E'$, $E * F$ r $E' * F'$ and $\langle S\,, E\rangle$ r $\langle S'\,, E'\rangle$.

Definition 6. Parallel reduction $\to_{/\!/}$ is the least contextual relation $\to_{/\!/}$ such that, if $s \to_{/\!/} S'$, $c \to_{/\!/} C'$, $e \to_{/\!/} E'$, and for all $i = 0, \ldots, n$, $S_i \to_{/\!/} S'_i$ and $e_i \to_{/\!/} E'_i$, then:

$$\begin{array}{rcll} \langle \mu\alpha\, c\,, e\rangle & \to_{/\!/} & C'\left[E'/\alpha\right] & (4) \\ \langle \lambda x\, s\,, e * \prod_{i=0}^{n} (S_i \cdot e_i)\rangle & \to_{/\!/} & \left\langle \lambda y\, \mu\alpha \left\langle S'\left[y + \sum_{i=0}^{n} S'_i/x\right], \alpha * \prod_{i=0}^{n} E'_i\right\rangle, E'\right\rangle & (5) \\ \langle \lambda x\, s\,, \prod_{i=1}^{n} (S_i \cdot e_i)\rangle & \to_{/\!/} & \left\langle s\left[\sum_{i=1}^{n} S'_i/x\right], \prod_{i=1}^{n} E'_i\right\rangle. & (6) \end{array}$$

with y a fresh variable α a fresh name in (5).

It should be clear that $\to_\beta \subset \to_{/\!/}$, in the sense that if $\Theta \to_\beta \Theta'$ then $\Theta \to_{/\!/} \Theta'$. In particular, (6) is reminiscent of the coarse version of reduction. Moreover, $\to_{/\!/} \subset \to_\beta^*$ by simple contextuality of $\to_\beta$. The following lemma states the essential property of parallel reduction.

Lemma 7. *If Θ and Θ' are objects, S and $S' \in \mathcal{T}^+$, and E and $E' \in \mathcal{E}^+$, such that $\Theta \to_{/\!/} \Theta'$, $S \to_{/\!/} S'$ and $E \to_{/\!/} E'$, then for every variable x and every name α, the following reductions hold:*

$$\Theta\left[S/x\right] \to_{/\!/} \Theta'\left[S'/x\right] \quad \textit{and} \quad \Theta\left[E/\alpha\right] \to_{/\!/} \Theta'\left[E'/\alpha\right] \ .$$

Proof. This is a simple induction on Θ, using contextuality of $\to_{/\!/}$. □

We now prove that $\to_{/\!/}$ enjoys the diamond property. Assume $s \in \mathcal{T}$ and $e \in \mathcal{E}$, and write $e = (\prod_{i=1}^{n} S_i \cdot e_i) * (\prod_{j=1}^{k} \alpha_j)$; we then define

$$\langle \lambda x\, s\,, e\rangle_0 = \begin{cases} \langle s\left[\sum_{i=1}^{n} S_i/x\right], \prod_{i=1}^{n} e_i\rangle & \text{if } k = 0; \\ \left\langle \lambda y\, \mu\alpha \left\langle s\left[y + \sum_{i=1}^{n} S_i/x\right], \alpha * \prod_{i=1}^{n} e_i\right\rangle, \prod_{j=1}^{k} \alpha_j\right\rangle & \text{if } nk > 0; \\ \langle \lambda x\, s\,, e\rangle & \text{otherwise.} \end{cases}$$

Clearly, $\langle \lambda x\, s\,, e\rangle \to_{/\!/} \langle \lambda x\, s\,, e\rangle_0$, as a particular case of (5) or (6).

Definition 8. We define full reduction as follows:

$$\begin{array}{rclrclrcl} x\downarrow & = & x & \alpha\downarrow & = & \alpha & \langle x\,, e\rangle\downarrow & = & \langle x\,, e\downarrow\rangle \\ (\lambda x\, s)\downarrow & = & \lambda x\, s\downarrow & (S \cdot e)\downarrow & = & S\downarrow \cdot e\downarrow & \langle \lambda x\, s\,, e\rangle\downarrow & = & \langle \lambda x\, s\downarrow, e\downarrow\rangle_0 \\ (\mu\alpha\, c)\downarrow & = & \mu\alpha\, c\downarrow & \mathbf{1}\downarrow & = & \mathbf{1} & \langle \mu\alpha\, c\,, e\rangle\downarrow & = & c\downarrow\left[e\downarrow/\alpha\right] \\ & & & (\sigma * e)\downarrow & = & \sigma\downarrow * e\downarrow & & & \end{array}$$

and $(\sum_{i=1}^{n} \theta_i)\downarrow = \sum_{i=1}^{n} \theta_i\downarrow$.

Full reduction fires all possible redexes in an object. Then one obtains the diamond property for parallel reduction:

Lemma 9. *If Θ and Θ' are objects such that $\Theta \to_{/\!/} \Theta'$, then $\Theta' \to_{/\!/} \Theta\downarrow$.*

Proof. This result is proved by inspecting all possible cases of reduction $\Theta \to_{/\!/} \Theta'$, using Lemma 7 in redex cases. □

Theorem 10. *Reduction is confluent.*

Proof. This is a corollary of Lemma 9 and the inclusions $\to_\beta \subset \to_{/\!/} \subset \to_\beta^*$. □

3 Relational Semantics

In this section, we adapt the system R of [dC06] to the setting of convolution $\overline{\lambda}\mu$-calculus: we introduce a type system, the types of which are elements of the extensional λ-model described in [BE04].

A Reflexive Object in the Category of Sets and Relations. If X is a set, we denote by $\mathcal{M}_{\mathrm{fin}}(X)$ the set of all finite multisets $[x_1, \ldots, x_n]$ of elements $x_1, \ldots, x_n \in X$ (possibly with repetitions). Also, we write $(\mathcal{M}_{\mathrm{fin}}(X))^{(\omega)}$ for the set of all infinite sequences $a = (a(i))_{i\in\omega}$ of finite multisets of elements of X such that $a(i) = []$ holds for almost all $i \in \omega$.

We define an increasing family $(\mathcal{D}_n)_{n\in\mathbf{N}}$ of sets by: $\mathcal{D}_0 = \emptyset$ and $\mathcal{D}_{n+1} = (\mathcal{M}_{\mathrm{fin}}(\mathcal{D}_n))^{(\omega)}$. Then we write $\mathcal{D} = \bigcup_{n\in\mathbf{N}} \mathcal{D}_n$. If $A \in \mathcal{M}_{\mathrm{fin}}(\mathcal{D})$ and $a \in \mathcal{D}$, we write $A :: a$ for the sequence b such that $b(0) = A$ and $b(i+1) = a(i)$ for all $i \in \omega$. This mapping is clearly a bijection between $\mathcal{D}$ and $\mathcal{M}_{\mathrm{fin}}(\mathcal{D}) \times \mathcal{D}$. We write ι for the sequence in which only the empty multiset occurs: $\iota(i) = []$ for all $i \in \omega$, so that $\iota = [] :: \iota$. Observe that $\mathcal{D}_1 = \{\iota\}$.

Type System. Call types the elements of $\mathcal{D}$. We impose a commutative monoid structure on types as follows. For all $a, b \in \mathcal{D}$, we define $a \star b$ as the sequence such that, for all $i \in \omega$, $(a \star b)(i) = a(i) + b(i)$ where $+$ denotes the union of multisets. Clearly ι is neutral for that associative and commutative operation.

A variable environment is a function $\Gamma : \mathfrak{V} \longrightarrow \mathcal{M}_{\mathrm{fin}}(\mathcal{D})$ such that $\Gamma(x) = []$ for almost all $x \in \mathfrak{V}$. If $x \in \mathfrak{V}$ and $A \in \mathcal{M}_{\mathrm{fin}}(\mathcal{D})$, we write $x : A$ for the variable environment Γ such that: $\Gamma(x) = A$ and, for all $y \neq x$, $\Gamma(y) = []$. If Γ and Γ' are variable environments, we write $\Gamma + \Gamma'$ for the variable environment defined by $(\Gamma + \Gamma')(x) = \Gamma(x) + \Gamma'(x)$. If Γ is a variable environment, we define its support $\mathrm{Supp}(\Gamma) = \{x \in \mathfrak{V}; \Gamma(x) \neq []\}$.

Similarly, a name environment is a function $\Delta : \mathfrak{N} \longrightarrow \mathcal{D}$ such that $\Delta(\alpha) = \iota$ for almost all $\alpha \in \mathfrak{N}$. If $\alpha \in \mathfrak{N}$ and $a \in \mathcal{D}$, we write $\alpha : a$ for the name environment Δ such that: $\Delta(\alpha) = a$ and, for all $\beta \neq \alpha$, $\Delta(\beta) = \iota$. If Δ and Δ' are name environments, we write $\Delta \star \Delta'$ for the name environment defined by $(\Delta \star \Delta')(\alpha) = \Delta(\alpha) \star \Delta'(\alpha)$ and we set $\mathrm{Supp}(\Delta) = \{\alpha \in \mathfrak{N}; \Delta(\alpha) \neq \iota\}$.

Now we introduce type system $R_{\bar{\lambda}\mu*}$ for the objects of convolution $\bar{\lambda}\mu$-calculus. Typing judgements are of form $\Gamma \vdash S : a \mid \Delta$, $\Gamma \mid E : a \vdash \Delta$ or $C : (\Gamma \vdash \Delta)$, where Γ is a variable environment and Δ is a name environment. We may omit Γ (resp. Δ) if it is the constant function with value $[]$ (resp. ι). The rules of system $R_{\bar{\lambda}\mu*}$ are given in Fig. 1.

The reader may refer to [DGL05] and check that the rules of system $R_{\bar{\lambda}\mu*}$, restricted to the objects of ordinary $\bar{\lambda}\mu$-calculus, are quite similar to those of system $\mathcal{M}^{\cap\cup}$. This similarity actually extends to the fact that all weakly normalizing objects are typable in system $R_{\bar{\lambda}\mu*}$, as we will show later. This feature is a characteristic of intersection type systems: this was already prominent in system R.

Example 11. The term $\lambda x\, \mu\alpha\, \langle x\, , x \cdot \alpha \rangle$ (the $\bar{\lambda}\mu$-calculus variant of $\delta = \lambda x\, (x)\, x$) has

$$\frac{}{x : [a] \vdash x : a \mid}\ \mathsf{Var} \qquad \frac{\Gamma + x : A \vdash s : a \mid \Delta \quad \Gamma(x) = []}{\Gamma \vdash \lambda x\, s : A :: a \mid \Delta}\ \mathsf{Abs}$$

$$\frac{c : (\Gamma \vdash \Delta \star \alpha : a) \quad \Delta(\alpha) = \iota}{\Gamma \vdash \mu\alpha\, c : a \mid \Delta}\ \mathsf{Mu} \qquad \frac{}{\mid \alpha : a \vdash \alpha : a}\ \mathsf{Name}$$

$$\frac{\Gamma_0 \mid e : a_0 \vdash \Delta_0 \quad \Gamma_1 \vdash S : a_1 \mid \Delta_1 \quad \cdots \quad \Gamma_n \vdash S : a_n \mid \Delta_n}{\Gamma_0 + \cdots + \Gamma_n \mid S \cdot e : [a_1, \ldots, a_n] :: a_0 \vdash \Delta_0 \star \cdots \star \Delta_n}\ \mathsf{Cons}$$

$$\frac{}{\mid \mathbf{1} : \iota \vdash}\ \mathsf{Unit} \qquad \frac{\Gamma \mid \sigma : a \vdash \Delta \quad \Gamma' \mid e : a' \vdash \Delta'}{\Gamma + \Gamma' \mid \sigma * e : a \star a' \vdash \Delta \star \Delta'}\ \mathsf{Conv}$$

$$\frac{\Gamma \vdash s : a \mid \Delta \quad \Gamma' \mid e : a \vdash \Delta'}{\langle s\,, e\rangle : (\Gamma + \Gamma' \vdash \Delta \star \Delta')}\ \mathsf{Cut} \qquad \frac{\Gamma \vdash \theta_i : a \mid \Delta}{\Gamma \vdash \sum_{i=0}^{n} \theta_i : a \mid \Delta}\ \mathsf{Sum}_i$$

Figure 1: Typing rules for system $R_{\bar{\lambda}\mu *}$

the following typing derivation, recalling that $\iota = [] :: \iota$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\iota] \vdash x : \iota \mid} \quad \dfrac{\dfrac{}{\mid \alpha : \iota \vdash}}{\mid x \cdot \alpha : \iota \vdash}}{\langle x\,, x \cdot \alpha\rangle : (x : [\iota] \vdash)}}{x : [\iota] \vdash \mu\alpha\, \langle x\,, x \cdot \alpha\rangle : \iota \mid}}{\vdash \lambda x\, \mu\alpha\, \langle x\,, x \cdot \alpha\rangle : [\iota] :: \iota \mid}\ .$$

Lemma 12. *Every term, context or command which is in normal form is typable.*

Proof. The result is proved by mutual induction on normal terms, contexts and commands. Among the typing rules in figure 1, only Cut involves some compatibility condition on the types of subobjects. Hence the only interesting induction case is that of simple commands. Simple commands in normal form are those $c = \langle s\,, e\rangle$ such that:

(i) either s is a variable and e is a simple context in normal form;

(ii) or $s = \lambda x\, t$, where t is a simple term in normal form, and $e = \alpha_0 * \cdots * \alpha_n$ is a product of names, with $n > 0$.

In both cases, it is easy to build a typing derivation using the inductive hypothesis and axiom rules (Var or Name). □

Denote by $\mathrm{FV}(\Theta)$ (resp. $\mathrm{FN}(\Theta)$) the set of all variables (resp. names) free in Θ. To simplify some of our next statements, we write $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta, a)$ for:

$$\begin{aligned} \Gamma \vdash S : a \mid \Delta \ &\text{ if } \ \Theta = S \in \mathcal{T}^+ \ ; \\ \Gamma \mid E : a \vdash \Delta \ &\text{ if } \ \Theta = E \in \mathcal{E}^+ \ ; \\ C : (\Gamma \vdash \Delta) \ &\text{ if } \ \Theta = C \in \mathcal{C}^+ \ . \end{aligned}$$

Lemma 13. *If* $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta, a)$, *then* $\mathrm{Supp}(\Gamma) \subseteq \mathrm{FV}(\Theta)$ *and* $\mathrm{Supp}(\Delta) \subseteq \mathrm{FN}(\Theta)$.

Proof. This is easily proved by induction on Θ. □

Denotational Semantics. We define the relational semantics of an object, as the set of all its typings in system $R_{\bar{\lambda}\mu *}$. More precisely:

Definition 14. Assume $S \in \mathcal{T}^+$, $E \in \mathcal{E}^+$ and $C \in \mathcal{C}^+$ are such that $\mathrm{FV}(\Theta) \subseteq \{x_1, \ldots, x_n\}$ and $\mathrm{FN}(\Theta) \subseteq \{\alpha_1, \ldots, \alpha_p\}$, for $\Theta = S, E, C$. We define

$$\begin{array}{rcl} [\![S]\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n} & = & \{(\Gamma(x_1), \ldots, \Gamma(x_n), a, \Delta(\alpha_1), \ldots, \Delta(\alpha_p));\ \Gamma \vdash S : a \mid \Delta\}\ , \\ [\![E]\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n} & = & \{(\Gamma(x_1), \ldots, \Gamma(x_n), a, \Delta(\alpha_1), \ldots, \Delta(\alpha_p));\ \Gamma \mid E : a \vdash \Delta\} \text{ and} \\ [\![C]\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n} & = & \{(\Gamma(x_1), \ldots, \Gamma(x_n), \Delta(\alpha_1), \ldots, \Delta(\alpha_p));\ C : (\Gamma \vdash \Delta)\}\ . \end{array}$$

Remark 15. The reader can easily check that

$$[\![(S \cdot e) * (S' \cdot e')]\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n} = [\![(S + T) \cdot (e * e')]\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n}\ .$$

The following three lemmas are proved by induction on objects.

Lemma 16. *We have* $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta[\mathbf{0}/x], a)$ *if and only if* $x \notin \mathrm{Supp}(\Gamma)$ *and* $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta, a)$.

Lemma 17. *Assume* $x \notin \mathrm{FV}(T)$. *Then the following are equivalent:*

- $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta[x + T/x], a)$;
- *there exist variable environments* $\Gamma', \Gamma_1, \ldots, \Gamma_n$, *and name environments* Δ', $\Delta_1, \ldots, \Delta_n$ *and types* $a_1, \ldots, a_n \in \mathcal{D}$ *such that*
  - $\Gamma = \Gamma' + \Gamma_1 + \cdots + \Gamma_n$ *and* $\Delta = \Delta' \star \Delta_1 \star \cdots \star \Delta_n$;
  - *for all* $i = 1, \ldots, n$, $\Gamma_i \vdash T : a_i \mid \Delta_i$;
  - *and* $R_{\bar{\lambda}\mu *}(\Gamma' + x : [a_1, \ldots, a_n], \Delta', \Theta, a)$.

Lemma 18. *The following statements are equivalent:*

- $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta[E/\alpha], a)$;
- *there exist variable environments* Γ' *and* Γ'', *name environments* Δ' *and* Δ'', *and type* $b \in \mathcal{D}$, *such that*
  - $\alpha \notin \mathrm{Supp}(\Delta')$;
  - $\Gamma = \Gamma' + \Gamma''$ *and* $\Delta = \Delta' \star \Delta''$;
  - $\Gamma'' \mid E : b \vdash \Delta''$;
  - *and* $R_{\bar{\lambda}\mu *}(\Gamma', \Delta' \star \alpha : b, \Theta, a)$.

Theorem 19. *If* $\Theta \to_\beta \Theta'$, *then we have:* $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta, a)$ *iff* $R_{\bar{\lambda}\mu *}(\Gamma, \Delta, \Theta', a)$. *If moreover* $\mathrm{FV}(\Theta) \subseteq \{x_1, \ldots, x_n\}$ *and* $\mathrm{FN}(\Theta) \subseteq \{\alpha_1, \ldots, \alpha_p\}$, *then*

$$[\![\Theta]\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n} = [\![\Theta']\!]^{\alpha_1,\ldots,\alpha_p}_{x_1,\ldots,x_n}\ .$$

Proof. The proof is by induction on Θ, inspecting all possible cases for reduction $\Theta \to_\beta \Theta'$, and using the previous three lemmas in redex cases. □

Hence the relational semantics is preserved by reduction. As a corollary, Lemma 12 implies that every object that has a normal form is typable.

4 Future Work

On Pure Calculi. Although grounded in ideas coming from models of differential λ-calculus, convolution $\bar{\lambda}\mu$-calculus provides no differentiation primitive. Indeed, recall from our introduction that the nets associated with convolution $\bar{\lambda}\mu$-calculus are polarized nets, extended with cocontraction and coweakening on types $!o$ and i. In particular, they do not involve derivative ∂.

One may augment these nets by including ∂ and the associated cut elimination rules, but this needs caution. Uncontrolled use of ∂ breaks one essential property of polarized nets: namely, the occurrence of at most one positive type ($!o$ or i in our setting) among all output wires. That matter is discussed in [Vau07b] in more details.

This remark, however, does not hamper the fact that one may propose differential extensions of convolution $\bar{\lambda}\mu$-calculus. Some first attempts even suggest that the introduction of convolution product of contexts actually simplifies the presentation of a would-be differential $\bar{\lambda}\mu$-calculus.

On Denotational Semantics. In [dC06], Carvalho provides precise results relating the relational semantics of λ-terms with their normalization properties (which are very close to those we expect from an intersection type system); he also provides bounds for the execution time of terms in variants of Krivine's abstract machine, according to the size of their typing derivations in system R. We do not know yet, to which extent these results may accomodate themselves to the setting of Bucciarelli-Ehrhard's model and convolution $\bar{\lambda}\mu$-calculus (or even usual $\lambda\mu$- or $\bar{\lambda}\mu$-calculus for that matter).

Another promising direction for further research is to study more precisely how the categorical constructions of [LR03] may be extended to a setting with costructural rules.

References

[BE04] Antonio Bucciarelli and Thomas Ehrhard. An extensional model of the lambda-calculus in the category of sets and relations. Manuscript, 2004.

[dC06] Daniel de Carvalho. Execution time of lambda-terms via non uniform semantics and intersection types. Research report, 2006.

[DGL05] Daniel J. Dougherty, Silvia Ghilezan, and Pierre Lescanne. Intersection and union types in the lambda-bar-mu-mu-tilde-calculus. *Electr. Notes Theor. Comput. Sci.*, 136:153–172, 2005.

[DJS95] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Sequent calculi for second order logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 211–224. Cambridge University Press, 1995.

[Ehr01] Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Mathematical Structures in Computer Science*, 12:579–623, 2001.

[Ehr05] Thomas Ehrhard. Finiteness spaces. *Mathematical. Structures in Comp. Sci.*, 15(4):615–646, 2005.

[ER03] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309:1–41, 2003.

[ER05] Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Electr. Notes Theor. Comput. Sci.*, 123:35–74, 2005.

[Her95] Hugo Herbelin. *Séquents qu'on calcule*. Phd thesis, Université Paris 7, 1995.

[Laf95] Yves Lafont. From proof nets to interaction nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 225–247. Cambridge University Press, 1995.

[Lau02] Olivier Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.

[Lau03] Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, January 2003.

[LR03] Olivier Laurent and Laurent Regnier. About translations of classical logic into polarized linear logic. In *Proceedings of the 18th annual IEEE symposium on Logic In Comp. Sci.*, pages 11–20. IEEE Computer Society Press, June 2003.

[Par92] Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning*, pages 190–201. Springer-Verlag, 1992.

[Reg92] Laurent Regnier. *Lambda-calcul et réseaux*. PhD thesis, Université Paris 7, 1992.

[Sch66] Laurent Schwartz. *Théorie des distributions*. Hermann, 1966.

[Vau06] Lionel Vaux. λ-calculus in an algebraic setting. Research report, 2006.

[Vau07a] Lionel Vaux. The differential $\lambda\mu$-calculus. To appear in Theoretical Computer Science, `doi:10.1016/j.tcs.2007.02.028`, 2007.

[Vau07b] Lionel Vaux. Polarized proof nets and differential structures. Unpublished manuscript, 2007.