Pre-proceedings

First International Workshop on Linearity

LINEARITY 2009

Coimbra, Portugal

12 September 2009

Satellite event of the
18th EACSL Annual Conference on Computer Science Logic

Editors:

Mário Florido and Ian Mackie

Preface

Linearity has been the key feature in several lines of research in both theoretical and practical approaches to computer science. In the theoretical side all the work stemming from linear logic dealing with proof technology, complexity classes and more recently quantum computation. In the practical side work on program analysis, expressive operational semantics for programming languages, linear programming languages, program transformation, update analysis and efficient implementation techniques. The aim of this workshop is to bring together researchers who are currently developing theory and applications of linear calculi, in order to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in this area. LINEARITY 2009 is a one-day satellite event of CSL 2009, held in Coimbra, Portugal.

Topics of interest include foundational calculus, models, applications to programming languages and systems. This included, but not limited to:

- Linear types: session types, etc
- Linear calculi;
- Functional calculi: lambda-calculus, rho-calculus, term and graph rewriting;
- Object calculi;
- Interaction-based systems: interaction nets, games;
- Concurrent models: process calculi, action graphs;
- Calculi expressing locality, mobility, and active data;
- Quantum computational models;
- Biological or chemical models of computation;

The Programme Committee selected five papers for presentation at LINEARITY 2009. In addition, the programme includes invited talks by Alex Simpson (Edinburgh), Simona Ronchi Della Rocca, (Torino) and Simone Martini (Bologna). The Final Proceedings of LINEARITY 2009 will appear as a volume of *Electronic Proceedings in Theoretical Computer Science*.

We would like to thank all those who contributed to LINEARITY 2009. We are grateful to the external referees for their careful and efficient work in the reviewing process, and in particular the programme committee members:

- Sandra Alves
- Ugo Dal Lago
- Maribel Fernández

- Simon Gay
- Mário Florido (co-chair)
- Martin Hofmann
- Ian Mackie (co-chair)
- Greg Morrisett
- Alan Mycroft
- Luke Ong
- Luca Paolini

Categorical Models for a Semantically Linear λ-calculus

Marco Gaboardi
Dipartimento di Informatica
Università degli Studi di Torino
gaboardi@di.unito.it

Mauro Piccolo
Dipartimento di Informatica
Università degli Studi di Torino
Preuves, Programmes et Systèmes
Université de Paris VII
piccolo@di.unito.it

This paper is about a categorical approach to model a very simple Semantically Linear λ-calculus, named $\mathcal{S}\ell\lambda$-calculus. This is a core calculus underlying the programming language $\mathcal{S}\ell$PCF. In particular, in this work, we introduce the notion of $\mathcal{S}\ell\lambda$ Category, which is able to describe a very large class of sound models of $\mathcal{S}\ell\lambda$-calculus. $\mathcal{S}\ell\lambda$ Category extends in the natural way Benton, Biermann, Hyland and de Paiva's Linear Category, in order to soundly interpret all the construct of $\mathcal{S}\ell\lambda$-calculus. This category is general enough to catch interesting models in Scott Domains and Coherence Spaces.

1 Introduction

$\mathcal{S}\ell\lambda$-calculus - acronym for $\mathcal{S}$emantically ℓinear λ-calculus - is a simple term calculus based on λ-calculus. More specifically, $\mathcal{S}\ell\lambda$-calculus extends and refines simply typed λ-calculus by imposing a linearity discipline on the usage of certain kinds of variables, as well as by adding some programming features to the calculus - like numerals, conditional and fix point operators - to make the calculus expressive enough to program all first-order computable functions.

Semantically Linear λ-calculus was already introduced in [7] (with an additional operator called `which?`, which is not present here) as the term rewriting system on which the programming language $\mathcal{S}\ell$PCF is based [3, 7]. $\mathcal{S}\ell$PCF is based on a syntactical restriction of PCF conceived in order to program only linear functions between Coherence Space. In particular, in [7] we define a *concrete* model of $\mathcal{S}\ell$PCF (and consequently of $\mathcal{S}\ell\lambda$-calculus) in the category **Coh** of Coherence Space and Linear Functions, for which we prove a full abstraction result.

The aim of this paper is to give an *abstract* description of models of $\mathcal{S}\ell\lambda$-calculus. This in order to highlight the properties that a mathematical structure must satisfy to model, by means of its equational theory, the operational theory induced by the reduction rules of the calculus. We give this abstract description in terms of category theory and we show that the obtained notion can be used to build concrete models in different mathematical structures.

We recall that the category **Coh**, as well as many other categories, is a well known concrete instance of Benton, Bierman, Hyland and de Paiva's Linear Categories, introduced in [1] to provide an abstract description of models of Intuitionistic Linear Logic. All these categories are symmetric monoidal closed and they are equipped with a symmetric monoidal comonad ! used to interpret the exponential modality and satisfying certain properties [1]. The idea is to impose enough conditions on the comonad in order to make its induced Kleisli category a Cartesian Closed Category with exponential object $A \Rightarrow B = !A \multimap B$. The original construction does not require this, but it would actually be the case, if the monoidal closed category is also cartesian.

Submitted to:
LINEARITY 2009

In this paper, we introduce the notion of $\mathcal{S}\ell\lambda$ Category which extends in the natural way the definition of Linear Category, in order to be able to interpret all programming constructs of $\mathcal{S}\ell\lambda$-calculus.

We ask that this category admits a morphism acting like a "conditional" and a morphism acting like a "fix-point operator". The latter turns out to be the expected decomposition of a fix-point morphism in a Cartesian Closed Category. Furthermore, to interpret ground values, we require the existence of a distinguished object N with the usual zero and successor and predecessor morphisms satisfying the expected equation. However, since variables of ground type can be freely duplicated and erased, we need to ask that all numeral morphisms behaves properly with respect to the comonad !. For this purpose we ask the existence of !-coalgebra $p : N \to !N$ which is also comonoidal and moreover we ask that all numeral morphisms are both coalgebraic and comonoidal.

The notion of natural number object in a symmetric monoidal closed category is not new and it was introduced by Paré and Román in [8]. Based on this definition Mackie, Román and Abramsky introduced an internal language for autonomous categories with natural number objects in [4]. The main similarity between the definitions of natural number object given in [8, 4] and our definition is the requirement of comonoidality of the natural number object; however their definition does not take into consideration the relationship between the natural number object and the exponential comonad !; in fact there, only a strictly linear language without exponential was analyzed. More details on this matter can be found in [9].

We prove that the proposed categorical model of $\mathcal{S}\ell\lambda$-calculus enjoys soundness with respect to the smallest equivalence containing $\mathcal{S}\ell\lambda$-reduction. This Soundness Theorem relies on three distinct substitution lemmas corresponding to the three kind of substitution in the calculus. We believe that the model is also complete but we leave this issue for future developments.

Moreover, this abstract definition of model for $\mathcal{S}\ell\lambda$-calculus allows us to analyse in a modular way many different concrete examples. In particular we build a non-trivial model of $\mathcal{S}\ell\lambda$-calculus in the category **StrictCPO** of Scott Domains and strict continuous functions. We also study models of $\mathcal{S}\ell\lambda$-calculus in the category **Coh** of Coherence Spaces and Linear Functions. More specifically, we show that the model we define in [7] is equivalent to a particular instance of $\mathcal{S}\ell\lambda$ Category, in the category **Coh**.

2 Semantically Linear λ-calculus

$\mathcal{S}\ell\lambda$-calculus is a term rewriting system very close to λ-calculus, on which the programming language $\mathcal{S}\ell$PCF is based [3, 7]. Truth-values of $\mathcal{S}\ell\lambda$-calculus are encoded as integers (zero encodes "true" while any other numeral stands for "false"). The set of $\mathcal{S}\ell\lambda$-*types* is defined as, $\sigma,\tau ::= \iota \mid (\sigma \multimap \tau)$ where ι is the only *atomic* type (i.e. natural numbers), $\multimap$ is the only *type constructor* and $\sigma,\tau,...$ are meta-variables ranging over types. Let Var^σ,SVar^σ be enumerable disjoint sets of variables of type σ. The set of *ground* variables is Var^ι, the set of *higher-order* variables is $\mathrm{HVar} = \bigcup_{\sigma,\tau} \mathrm{Var}^{\sigma\multimap\tau}$, and the whole set of variables is $\mathrm{Var} = \mathrm{Var}^\iota \cup \mathrm{HVar} \cup \mathrm{SVar}$. Letters $\mathtt{x}^\sigma$ range over variables in Var^σ, letters $\mathtt{y}^\iota,\mathtt{z}^\iota,\ldots$ range over variables in Var^ι, letters $\mathtt{f}^{\sigma\multimap\tau},\mathtt{g}^{\sigma\multimap\tau},\ldots$ range over variables in HVar, while F^σ,$F_1{}^\sigma$,$F_2{}^\sigma$,$\ldots$ range over stable variables, namely variables in SVar^σ. Last, $\varkappa$ will denote any kind of variables. Latin letters $\mathtt{M},\mathtt{N},\mathtt{L},\ldots$ range over terms.

A **basis** Γ is a finite list of variables in Var. We denote with Γ^* (resp Γ^ι) a basis Γ containing

$$\frac{}{\vdash \underline{0} : \iota}\ (z) \qquad \frac{}{\vdash \mathtt{succ} : \iota \multimap \iota}\ (s) \qquad \frac{}{\vdash \mathtt{pred} : \iota \multimap \iota}\ (p) \qquad \frac{\Gamma \cap \Delta = \emptyset \quad \Gamma \vdash \mathtt{M} : \iota \quad \Delta \vdash \mathtt{L} : \iota \quad \Delta \vdash \mathtt{R} : \iota}{\Gamma, \Delta \vdash \ell\mathtt{if}\, \mathtt{M}\, \mathtt{L}\, \mathtt{R} : \iota}\ (\ell\mathtt{if})$$

$$\frac{\Gamma, \varkappa_2^{\sigma_2}, \varkappa_1^{\sigma_1}, \Delta \vdash \mathtt{M} : \tau}{\Gamma, \varkappa_1^{\sigma_1}, \varkappa_2^{\sigma_2}, \Delta \vdash \mathtt{M} : \tau}\ (ex) \qquad \frac{}{\mathtt{x}^{\iota} \vdash \mathtt{x} : \iota}\ (gv) \qquad \frac{\Gamma \vdash \mathtt{M} : \tau}{\Gamma, \mathtt{x}^{\iota} \vdash \mathtt{M} : \tau}\ (gw) \qquad \frac{\Gamma, \mathtt{x}_1{}^{\iota}, \mathtt{x}_2{}^{\iota} \vdash \mathtt{M} : \tau}{\Gamma, \mathtt{x}^{\iota} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \mathtt{x}_2] : \tau}\ (gc)$$

$$\frac{}{\mathtt{f}^{\sigma \multimap \tau} \vdash \mathtt{f} : \sigma \multimap \tau}\ (hv) \qquad \frac{\Gamma \cap \Delta = \emptyset \quad \Gamma \vdash \mathtt{M} : \sigma \multimap \tau \quad \Delta \vdash \mathtt{N} : \sigma}{\Gamma, \Delta \vdash \mathtt{M}\mathtt{N} : \tau}\ (ap) \qquad \frac{\Gamma, \mathtt{x}^{\sigma} \vdash \mathtt{M} : \tau}{\Gamma \vdash \lambda \mathtt{x}^{\sigma}.\mathtt{M} : \sigma \multimap \tau}\ (\lambda)$$

$$\frac{}{F^{\sigma} \vdash F : \sigma}\ (sv) \qquad \frac{\Gamma, F_1^{\sigma}, F_2^{\sigma} \vdash \mathtt{M} : \tau}{\Gamma, F^{\sigma} \vdash \mathtt{M}[F/F_1, F_2] : \tau}\ (sc) \qquad \frac{\Gamma \vdash \mathtt{M} : \tau}{\Gamma, F^{\sigma} \vdash \mathtt{M} : \tau}\ (sw) \qquad \frac{\Gamma^{\iota}, \Delta^{*}, F^{\sigma} \vdash \mathtt{M} : \sigma}{\Gamma^{\iota}, \Delta^{*} \vdash \mu F.\mathtt{M} : \sigma}\ (\mu)$$

Table 1: Type assignment system for $\mathcal{S}\ell\lambda$-calculus

variables in SVar (resp. in GVar). We will denote with Γ, Δ the concatenation of two basis and with $\Gamma \cap \Delta$ the intersection of two basis, defined in the expected way.

Definition 1. *Typed* terms *of $\mathcal{S}\ell\lambda$-calculus are defined by using a type assignment proving judgements of the shape $\Gamma \vdash \mathtt{M} : \sigma$, in Table 1.*

Note that only higher-order variables are subject to syntactical constraints. Except for the $\ell\mathtt{if}$ construction typed by an additive rule doing an implicit contraction, higher-order variables are treated linearly. Ground and stable variables belong to distinct kinds only for sake of simplicity, their free use implies that $\mathcal{S}\ell\lambda$-calculus is not syntactically linear (in the sense of [7]).

Free variables of terms are defined as expected. A term $\mathtt{M}$ is *closed* if and only if $\mathrm{FV}(\mathtt{M}) = \emptyset$, otherwise $\mathtt{M}$ is *open*. Terms are considered up to α-equivalence, namely a bound variable can be renamed provided no free variable is captured. Moreover, $\mathtt{M}[\underline{\mathtt{n}}/\mathtt{y}]$, $\mathtt{M}[\mathtt{N}/\mathtt{f}]$ and $\mathtt{M}[\mathtt{N}/F]$ denote the expected capture-free substitutions.

Definition 2. *We denote $\rightsquigarrow$ the firing (without any context-closure) of one of the following rules:*

$$(\lambda \mathtt{f}^{\sigma \multimap \tau}.\mathtt{M})\mathtt{N} \rightsquigarrow_{\beta} \mathtt{M}[\mathtt{N}/\mathtt{f}] \qquad (\lambda \mathtt{z}^{\iota}.\mathtt{M})\underline{\mathtt{n}} \rightsquigarrow_{\iota} \mathtt{M}[\underline{\mathtt{n}}/\mathtt{z}] \qquad \mu F.\mathtt{M} \rightsquigarrow_{\Upsilon} \mathtt{M}[\mu F.\mathtt{M}/F]$$
$$\mathtt{pred}\,(\mathtt{succ}\,\underline{\mathtt{n}}) \rightsquigarrow_{\delta} \underline{\mathtt{n}} \qquad \ell\mathtt{if}\,\underline{\mathtt{0}}\,\mathtt{L}\,\mathtt{R} \rightsquigarrow_{\delta} \mathtt{L} \qquad \ell\mathtt{if}\,\underline{\mathtt{n+1}}\,\mathtt{L}\,\mathtt{R} \rightsquigarrow_{\delta} \mathtt{R}$$

We call redex each term or sub-term having the shape of a left-hand side of rules defined above. We denote $\rightarrow_{\mathcal{S}\ell}$ the contextual closure of $\rightsquigarrow$. Moreover, we denote $\rightarrow^{*}_{\mathcal{S}\ell}$ and $=_{\mathcal{S}\ell}$ respectively, the reflexive and transitive closure of $\rightarrow_{\mathcal{S}\ell}$. and the reflexive, symmetric and transitive closure of $\rightarrow_{\mathcal{S}\ell}$.

We remark that $\rightsquigarrow_{\beta}$ formalises a call-by-name parameter passing in case of an higher-order argument. On the other hand, $\rightsquigarrow_{\iota}$ formalises a call-by-value parameter passing, namely the reduction can fire only when the argument is a numeral. As done in [2], it is easy to prove properties as subject-reduction, post-position of δ-rules in a sequence of reductions, the confluence and a standardisation theorem.

3 Categorical model of $\mathcal{S}\ell\lambda$-calculus

In this section we define the categorical model of $\mathcal{S}\ell\lambda$-calculus and we prove its soundness with respect to $=_{\mathcal{S}\ell}$. We assume some familiarity with the notions of monoidal categories, comonoids, comonads, adjunctions and monoidal functors. For an introduction, see [5]. We begin by recalling the definition of Linear Category, given by Benton, Biermann, Hyland and de Paiva, which proposes a categorical notion a model of Intuitionistic Linear Logic.

Definition 3 (Linear Category [1]). *A* Linear Category $\mathcal{L} = \langle \mathbb{L}, !, \delta, \varepsilon, q, d, e\rangle$ *consists of (1) a symmetric monoidal closed category* $\langle \mathbb{L}, \otimes, \multimap, \mathbf{1}\rangle$*; (2) a symmetric monoidal comonad called* exponential comonad $\langle !, \delta, \varepsilon, q_{A,B}, q_{\mathbf{1}}\rangle : \mathbb{L} \to \mathbb{L}$*, such that (i) for every free* $!$*-coalgebra* $\langle !A, \delta_A\rangle$ *there are two distinguished monoidal natural transformations with components* $d_A : !A \to !A \otimes !A$ *and* $e_A : !A \to \mathbf{1}$ *which form a commutative comonoid and are coalgebra morphisms; (ii) whenever* $f : \langle !A, \delta_A\rangle \to \langle !B, \delta_B\rangle$ *is a coalgebra morphism between free coalgebras, then it is also a comonoid morphism.*

We now introduce our categorical model, by defining the notion of $\mathcal{S}\ell\lambda$-category whose morphisms will denote $\mathcal{S}\ell\lambda$-terms. We introduce some notation on the symmetric monoidal closed category $\mathbb{L}$ first. We let $\gamma_{A,B} : A \otimes B \cong B \otimes A$ the tensorial symmetric law. We denote with $\mathrm{curry}(-) : \mathbb{L}(C \otimes A, B) \to \mathbb{L}(C, A \multimap B)$ the isomorphism induced by the canonical adjunction. When $C = A \multimap B$, we denote with $\mathrm{eval} : A \multimap B \otimes A \to B$ the (unique!) morphism such that $\mathrm{curry}(\mathrm{eval}) = id_{A \multimap B}$.

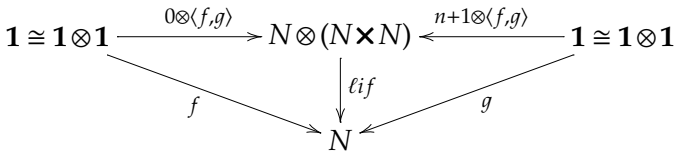
A $\mathcal{S}\ell\lambda$-category is a Linear Category [1], thus every $\mathcal{S}\ell\lambda$-Category is a model of Intuitionistic Linear Logic. Moreover, an $\mathcal{S}\ell\lambda$-category admits an object for natural numbers with additional morphisms allowing to duplicate and weaken occurrences of them and whose other morphisms respects the comonoidal structure induced by the exponential comonad. We also require the existence of a "conditional-like" morphism and a fix-point morphism for every object B in the Kleisli category over the comonad $!$, which is cartesian closed.

Definition 4 ($\mathcal{S}\ell\lambda$ Category). *A* $\mathcal{S}\ell\lambda$ *Category is a linear category* $\mathcal{L} = \langle \mathbb{L}, !, \delta, \varepsilon, q, d, e\rangle$ *such that*

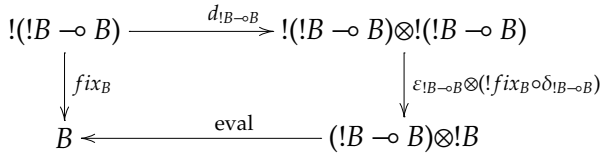
Numerals. *There exists a* $!$*-coalgebra* $\langle N, p\rangle$ *such that*

1. *N is a simple object for numerals with morphisms $0 : \mathbf{1} \to N$, $succ, pred : N \to N$ and all other numerals defined inductively and such that for all $n : \mathbf{1} \to N$ we have $n = pred \circ succ \circ n$.*
2. *there exists two morphisms $w_N : N \to \mathbf{1}$ and $c_N : N \to N \otimes N$ which form a commutative co-monoid and are such that*
   (a) *$0 : \mathbf{1} \to N$ and $succ : N \to N$ are both co-algebras and co-monoid morphisms.*
   (b) *$p : N \to !N$ is a co-monoid morphism.*

Conditional. *$\mathbb{L}$ is cartesian, and we denote with $\times : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ the cartesian product and with $\langle -, -\rangle : \mathbb{L}(C, A) \times \mathbb{L}(C, B) \to \mathbb{L}(C, A \times B)$ the pairing, obtained through the canonical adjunction. Furthermore, there exists a morphism $\ell if : N \otimes (N \times N) \to N$ such that, for all $f, g : \mathbf{1} \to N$*



Fix-Point. *For every object B there exists a morphism $fix_B : !(!B \multimap B) \to B$ such that*



Definition 5 (Categorical $\mathcal{S}\ell\lambda$-model). *A categorical* $\mathcal{S}\ell\lambda$*-model consists of*

- *A* $\mathcal{S}\ell\lambda$ *Category* $\langle \mathcal{L}, N, p, c_N, w_N, \ell if, fix\rangle$*, where* $\mathcal{L} = \langle \mathbb{L}, !, \delta, \varepsilon, q, e, d\rangle$*.*

- *A mapping associating to every $\mathcal{S}\ell\lambda$-type σ, an object $[\![\sigma]\!]$ of $\mathbb{L}$ such that $[\![\iota]\!] = N$ and $[\![\sigma \multimap \tau]\!] = [\![\sigma]\!] \multimap [\![\tau]\!]$.*
- *Given a basis Γ we define $[\![\Gamma]\!]$ by induction as $[\![\emptyset]\!] = \mathbf{1}$, $[\![\mathtt{x}^\sigma, \Delta]\!] = [\![\sigma]\!] \otimes [\![\Delta]\!]$ and $[\![F^\sigma, \Delta]\!] = ![\![\sigma]\!] \otimes [\![\Delta]\!]$. Moreover, given a basis Γ such that $\Gamma^\iota = \mathtt{x}_1^\iota, \ldots, \mathtt{x}_\mathtt{n}^\iota$ (resp. $\Gamma^* = F_1^{\sigma_1}, \ldots, F_n^{\sigma_n}$) we denote with $p_\Gamma = p \otimes \cdots \otimes p$ n-times (resp. $\delta_\Gamma = \delta_{[\![\sigma_1]\!]} \otimes \cdots \otimes \delta_{[\![\sigma_n]\!]}$).*
  Given a term $\mathtt{M}$ such that $\Gamma \vdash \mathtt{M} : \sigma$ we associate a morphism $[\![\Gamma \vdash \mathtt{M} : \sigma]\!] : [\![\Gamma]\!] \to [\![\sigma]\!]$, such that
  1. $[\![\vdash \underline{\mathtt{0}} : \iota]\!] = 0$, $[\![\vdash \mathtt{succ} : \iota \multimap \iota]\!] = \mathrm{curry}(succ)$, $[\![\vdash \mathtt{pred} : \iota \multimap \iota]\!] = \mathrm{curry}(pred)$, $[\![\mathtt{x}^\iota \vdash \mathtt{x} : \iota]\!] = id_N$
  2. $[\![\mathtt{f}^{\sigma\multimap\tau} \vdash \mathtt{f} : \sigma \multimap \tau]\!]^{\mathbb{L}} = id_{[\![\sigma\multimap\tau]\!]}$, $[\![F^\sigma \vdash F : \sigma]\!] = \varepsilon_{[\![\sigma]\!]}$
  3. $[\![\Gamma^\iota, \Delta^* \vdash \mu F.\mathtt{M} : \sigma]\!] = fix_{[\![\sigma]\!]} \circ !\mathrm{curry}([\![\Gamma^\iota, \Delta^*, F^\sigma \vdash \mathtt{M} : \sigma]\!]) \circ q \circ (p_\Gamma \otimes \delta_\Delta)$
  4. $[\![\Gamma \vdash \lambda \mathtt{x}^\sigma.\mathtt{M} : \sigma \multimap \tau]\!] = \mathrm{curry}([\![\Gamma, \mathtt{x}^\sigma \vdash \mathtt{M} : \tau]\!])$
  5. $[\![\Gamma, \varkappa_1^{\sigma_1}, \varkappa_2^{\sigma_2}, \Delta \vdash \mathtt{M} : \tau]\!] = [\![\Gamma, \varkappa_2^{\sigma_2}, \varkappa_1^{\sigma_1}, \Delta \vdash \mathtt{M} : \tau]\!] \circ (id_{[\![\Gamma]\!]^{\mathbb{L}}} \otimes \gamma_{[\![\sigma_1]\!],[\![\sigma_2]\!]} \otimes id_{[\![\Delta]\!]})$
  6. $[\![\Gamma, \Delta \vdash \mathtt{MN} : \tau]\!] = \mathrm{eval} \circ ([\![\Gamma \vdash \mathtt{M} : \sigma \multimap \tau]\!] \otimes [\![\Delta \vdash \mathtt{N} : \sigma]\!])$.
  7. $[\![\Gamma, \Delta \vdash \ell\mathtt{if}\ \mathtt{M}\ \mathtt{L}\ \mathtt{R} : \iota]\!] = \ell if \circ ([\![\Gamma \vdash \mathtt{M} : \iota]\!] \otimes \langle [\![\Delta \vdash \mathtt{L} : \iota]\!], [\![\Delta \vdash \mathtt{R} : \iota]\!] \rangle)$.
  8. $[\![\Gamma, \mathtt{x}^\iota \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \mathtt{x}_2] : \tau]\!] = [\![\Gamma, \mathtt{x}_1{}^\iota, \mathtt{x}_2{}^\iota \vdash \mathtt{M} : \tau]\!] \circ id_{[\![\Gamma]\!]} \otimes c_N$
  9. $[\![\Gamma, F^\sigma \vdash \mathtt{M}[F/F_1, F_2] : \tau]\!] = [\![\Gamma, F_1{}^\sigma, F_2{}^\sigma \vdash \mathtt{M} : \tau]\!] \circ id_{[\![\Gamma]\!]} \otimes d_{[\![\sigma]\!]}$
  10. $[\![\Gamma, \mathtt{x}^\iota \vdash \mathtt{M} : \tau]\!] = [\![\Gamma \vdash \mathtt{M} : \tau]\!] \circ id_{[\![\Gamma]\!]} \otimes w_N$
  11. $[\![\Gamma, F^\sigma \vdash \mathtt{M} : \tau]\!] = [\![\Gamma \vdash \mathtt{M} : \tau]\!] \circ id_{[\![\Gamma]\!]} \otimes e_{[\![\sigma]\!]}$

The following theorem shows that the three kinds of syntactical substitutions are modelled by categorical composition of morphism. Let us observe that the substitution of a ground or higher order variable respectively with a numeral or a term is modelled directly with the composition in $\mathbb{L}$, while the substitution of a stable variable with a term is modelled with the composition in the category of coalgebras.

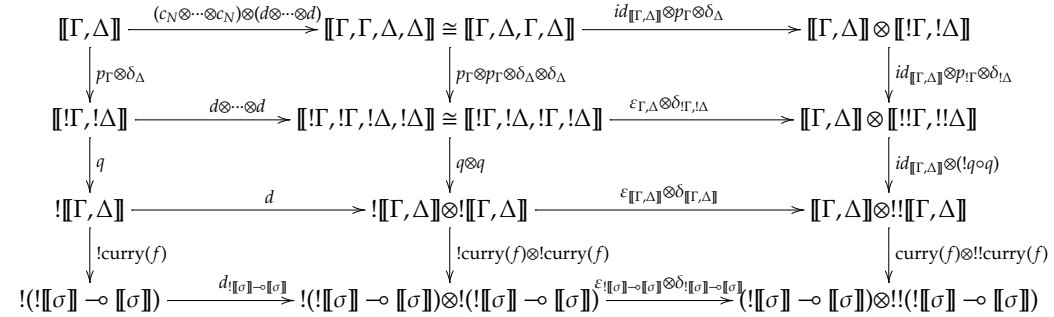
Theorem 1 (Semantical Substitution Lemma)**.**

1. *Let $\mathtt{M}$ be such that $\Gamma, \mathtt{x}^\iota, \Delta \vdash \mathtt{M} : \sigma$. Then $[\![\Gamma, \Delta \vdash \mathtt{M}[\underline{\mathtt{n}}/\mathtt{x}] : \sigma]\!] = [\![\Gamma, \mathtt{x}^\iota, \Delta \vdash \mathtt{M} : \sigma]\!] \circ (id_{[\![\Gamma]\!]} \otimes n \otimes id_{[\![\Delta]\!]})$.*
2. *Let $\mathtt{M}, \mathtt{N}$ be such that $\Gamma, \mathtt{f}^\sigma \vdash \mathtt{M} : \tau$ and $\Delta \vdash \mathtt{N} : \sigma$, with $\Gamma \cap \Delta = \emptyset$. Then $[\![\Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{f}] : \tau]\!] = [\![\Gamma, \mathtt{f}^\sigma \vdash \mathtt{M} : \tau]\!] \circ id_{[\![\Gamma]\!]} \otimes [\![\Delta \vdash \mathtt{N} : \sigma]\!]$.*
3. *Let $\mathtt{M}, \mathtt{N}$ be such that $\Gamma, F^\sigma \vdash \mathtt{M} : \tau$ and $\Delta_1^\iota, \Delta_2^* \vdash \mathtt{N} : \sigma$, with $\Gamma \cap \Delta_1 \cap \Delta_2 = \emptyset$. Then $[\![\Gamma, \Delta_1^\iota, \Delta_2^* \vdash \mathtt{M}[\mathtt{N}/F] : \tau]\!] = [\![\Gamma, F^\sigma \vdash \mathtt{M} : \tau]\!] \circ (id_{[\![\Gamma]\!]} \otimes (![\![\Delta_1^\iota, \Delta_2^* \vdash \mathtt{N} : \sigma]\!] \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2})))$.*

Proof. All the proofs follow by induction on the derivation of the typing judgements. The key point is to show that the transformations induced by the typing rules are natural on the unchanged components of the sequent. More details can be found in [9]. □

Theorem 2 (Soundness)**.** *Let $\mathtt{M}, \mathtt{N}$ such that $\Gamma \vdash \mathtt{M} : \sigma$ and $\Gamma \vdash \mathtt{N} : \sigma$. If $\mathtt{M} =_{\mathcal{S}\ell} \mathtt{N}$ then $[\![\Gamma \vdash \mathtt{M} : \sigma]\!] = [\![\Gamma \vdash \mathtt{N} : \sigma]\!]$*

Proof. The proof is by induction on the derivation of $\mathtt{M} =_{\mathcal{S}\ell} \mathtt{N}$. We develop only the case $\mathtt{M} =_{\mathcal{S}\ell} \mathtt{N}$ since $\mathtt{M} \rightsquigarrow_\mathtt{Y} \mathtt{N}$. To help with the notations, in the proofs we will relax a bit the definition of basis and we add types prefixed with a ! to the syntax of types. Thus, given a basis $\Gamma = \varkappa_1^{\sigma_1}, \ldots, \varkappa_n^{\sigma_n}$, we denote with $!\Gamma = \varkappa_1^{!\sigma_1}, \ldots \varkappa_n^{!\sigma_n}$ (where ! is just a syntactical annotation which will be interpreted with the corresponding categorical operator) and we will adapt in the canonical way the interpretation function on the so obtained types and basis. Thus $\mathtt{M} = \mu F.\mathtt{M}_1$ and $\mathtt{N} = \mathtt{M}_1[\mu F.\mathtt{M}_1/F]$. First of all, if we let $f = [\![\Gamma^\iota, \Delta^*, F^\sigma \vdash \mathtt{M}_1 : \sigma]\!]$, let us observe that the following diagram commutes



where the left square on the top commutes since p and δ are co-monoid morphisms, the right square on the top commutes since p and δ are co-algebras (observe that we used both commutative diagrams of the definition of co-algebra) and by bi-functoriality, the left square on the middle commutes since d is a monoidal natural transformation, the right square on the middle commutes since δ and ε are monoidal natural transformations, and finally the two squares on the bottom commutes respectively because being $!\mathrm{curry}(f)$ a co-algebra morphism between free co-algebra, it is also a co-monoid morphism, by naturality of ε and δ and by bi-functoriality. Thus, we have,

$$
\begin{aligned}
&[\![\Gamma^\iota,\Delta^* \vdash \mathtt{M}:\sigma]\!] = fix_{[\![\sigma]\!]}\circ!\mathrm{curry}(f)\circ q\circ(p_\Gamma\otimes\delta_\Delta)\\
&= \mathrm{eval}\circ(\varepsilon_{![\![\sigma]\!]\multimap[\![\sigma]\!]}\otimes(!fix_{[\![\sigma]\!]}\circ\delta_{![\![\sigma]\!]\multimap[\![\sigma]\!]}))\circ d_{![\![\sigma]\!]\multimap[\![\sigma]\!]}\circ!\mathrm{curry}(f)\circ q\circ(p_\Gamma\otimes\delta_\Delta)\\
&= \mathrm{eval}\circ(\mathrm{curry}(f)\otimes(!fix_{[\![\sigma]\!]}\circ!!\mathrm{curry}(f)\circ(!q\circ q)\circ(p_{!\Gamma}\otimes\delta_{!\Delta}))\circ(p_\Gamma\otimes\delta_\Delta))\circ((c_N\otimes\cdots\otimes c_N)\otimes(d\otimes\cdots\otimes d))\\
&= f\circ id_{[\![\Gamma,\Delta]\!]}\otimes(![\![\Gamma^\iota,\Delta^* \vdash \mu F.\mathtt{M}:\sigma]\!]\circ q\circ(p_\Gamma\otimes\delta_\Delta))\circ((c_N\otimes\cdots\otimes c_N)\otimes(d\otimes\cdots\otimes d))\\
&= [\![\Gamma^\iota,\Delta^*,\Gamma^\iota,\Delta^* \vdash \mathtt{N}:\sigma]\!]\circ((c_N\otimes\cdots\otimes c_N)\otimes(d\otimes\cdots\otimes d))
\end{aligned}
$$

where in the second line we used the fix-point law, in the third line we use the commutativity of the above diagram, in the fourth line we use the definition of interpretation and the naturality of q and since the category is monoidal closed and finally in the fifth line we use Theorem 1 point (3.). Then we can conclude by interpretation. □

4 Instances of $\mathcal{S}\ell\lambda$-categories

Scott Domains. Let **StrictCPO** be the category obtained by taking as **objects** *ω-algebraic bounded complete partial orders* (or *Scott domains*) and as **morphisms** *strict continuous functions*, namely those continuous functions that map the bottom element of the source object to the bottom element of the target. This category is monoidal closed, by taking the tensor product $A\otimes B$ to be the smash product $A\wedge B = \{\langle a,b\rangle \mid a\in A\setminus\{\bot\}, b\in B\setminus\{\bot\}\}\cup\{\bot\}$, the unit of the tensor product $\mathbf{1}$ to be the Sierpinsky Domain $\{\bot,\top\}$ with $\bot\leq\top$ and the function space $A\multimap B$ consisting of all strict maps between A and B under the point-wise order. Moreover if we take as exponential comonad !, the lifting constructor $(-)_\bot$, we obtain a linear category; we remind that, given a Scott Domain A, the domain $A_\bot$ is obtained from A by adding a new least element below the bottom of A (for more details see [10]). Observe that the Kleisli category over the comonad $(-)_\bot$ is the usual category of Scott Domains and continuous functions.

We can prove that **StrictCPO** is a $\mathcal{S}\ell\lambda$ Category, by taking N to be the usual flat domain of natural numbers with the coalgebra $p: N\to N_\bot$ such that $p(n)=n$ for all $n\neq\bot$. N is a commutative comonoid, by taking $w_N: N\to\mathbf{1}$ be such that $w_N(n)=\top$ and $c_N: N\to N\otimes N$ be such that $c_N(n)=\langle n,n\rangle$ for all $n\neq\bot$. **StrictCPO** is cartesian, by taking $A\times B$ to be the usual cartesian product of Scott Domains. Thus we can define $\ell if: N\otimes(N\times N)\to N$ to be such that $\ell if(c)=m_1$

if $c = \langle 0, \langle m_1, m_2 \rangle \rangle$, $\ell if(c) = m_2$ if $c = \langle n, \langle m_1, m_2 \rangle \rangle$ and $n \neq 0$ and $\ell if(c) = \perp$ otherwise. Finally, it follows easily by Knaster-Tarsky's Fix Point Theorem that the considered category admits fix point for every object. This model is shown to adequate with respect to the operational semantics of $\mathcal{S}\ell$PCF [9].

Coherence Spaces. A coherence space is a pair $X = \langle |X|, \frown_X \rangle$, consisting of a finite or countable set of tokens $|X|$ called *web* and a binary reflexive symmetric relation on $|X|$ called *coherence relation*. The set of *cliques* of X is given by $Cl(X) = \{x \subseteq |X| \mid a, b \in x \Rightarrow a \frown_X b\}$. This set ordered by inclusion forms a Scott Domain whose set of finite elements is the set $Cl_{fin}(X)$ of finite cliques. Two cliques $x, y \in Cl(X)$ are *compatible* when $x \cup y \in Cl(X)$. A continuous function $f : Cl(X) \to Cl(Y)$ is *stable* when it preserves intersections of compatible cliques. A stable function $f : Cl(X) \to Cl(Y)$ is *linear* when it is strict and preserves unions of compatibles cliques. Given a linear function $f : Cl(X) \to Cl(Y)$, we denote its trace with $\mathrm{tr}(f) = \{(a, b) \mid b \in f(\{a\})\}$. We say that a linear function f is less or equal than g according to the *stable order* when $\mathrm{tr}(f) \subseteq \mathrm{tr}(g)$.

Let **Coh** be the category of coherence spaces as **objects** and linear functions as **morphisms**. Given a coherence space X, we define $!X$ to be the coherence space having as web the set $Cl_{fin}(X)$ and as coherence relation, the compatibility relation between cliques. It is possible to prove that ! is an exponential comonad, thus **Coh** is a Linear Category [6]. Observe that the Kleisli category over the comonad ! is the category of Coherence Space and Stable Functions

The model of $\mathcal{S}\ell$PCF we define in [7] is based on this category, and can be obtained as follows. As in previous section, we take N to be the infinite flat domain of natural numbers, and we define w_N, c_N in an analogous way as before, as well as ℓif and the fix point operator. The !-coalgebra $p : N \to !N$ is such that $\mathrm{tr}(p) = \{(n, \{n\}) \mid n \in \mathbb{N}\} \cup \{(n, \{\emptyset\}) \mid n \in \mathbb{N}\}$.

For this model a full abstraction result for a fragment of $\mathcal{S}\ell$PCF is shown. As future work we propose to extend $\mathcal{S}\ell$PCF with opportune operators in order to establish an universality result with respect to the Coherence Space model.

References

[1] Nick Benton, G. M. Bierman, J. Martin E. Hyland & Valeria de Paiva (1992): *Linear λ-Calculus and Categorical Models Revisited*. In *CSL'92, LNCS* 702. Springer, pp. 61–84.

[2] Gérard Berry, Pierre-Louis Curien & Jean-Jacques Lévy (1985): *Full Abstraction for Sequential Languages: the State of the Art*. In: *Algebraic Semantics*. Cambridge University Press, pp. 89–132.

[3] Marco Gaboardi & Luca Paolini (2007): *Syntactical, Operational and Denotational Linearity*. In: *Workshop LOGIC*. Available at `http://www.unisi.it/eventi/LOGIC/`.

[4] Ian Mackie, Leopoldo Román & Samson Abramsky (1993): *An internal language for autonomous categories*. *Applied Categorical Structures* 1(3), pp. 311–343.

[5] Saunders MacLane (1998): *Categories for the Working Mathematician*. Springer. Second Edition.

[6] Paul-André Melliès (2003): *Categorical Models of Linear Logic Revisited*. PPS Tech. Report.

[7] Luca Paolini & Mauro Piccolo (2008): *Semantically linear programming languages*. In *PPDP'08*. ACM, pp. 97–107.

[8] Robert Paré & Leopoldo Román (1988): *Monoidal categories with natural numbers object*. *Studia Logica* 48(3), pp. 361–376.

[9] Mauro Piccolo (2009): *Linearity and Behind in Denotational Semantics*. Ph.D. thesis, Dipartimento di Informatica, Universitá di Torino/Laboratoire PPS, Université de Paris VII. In preparation.

[10] Alberto Pravato, Simona Ronchi Della Rocca & Luca Roversi (1999): *The call by value λ-calculus: a semantic investigation*. *Mathematical Structures in Computer Science* 9(5), pp. 617–650.

On linear information systems

A. Bucciarelli[†]
† Univeristé Paris Diderot
Paris, France
Preuves Programmes et Systèmes
Antonio.Bucciarelli@pps.jussieu.fr

A. Carraro[○†]
○ Univeristà Ca' Foscari
Venice, Italy
Dipartimento di Informatica
acarraro@dsi.unive.it

T. Ehrhard[†]
Thomas.Ehrhard@pps.jussieu.fr

A. Salibra[○]
salibra@dsi.unive.it

Scott's information systems provide a categorically equivalent, intensional description of Scott domains and continuous functions. Following a well established pattern in denotational semantics, we define a linear version of information systems, providing a model of intuitionistic linear logic (a new-Seely category), with a "set-theoretic" interpretation of exponentials that recovers Scott continuous functions via the co-Kleisli construction. From a domain theoretic point of view, linear information systems are equivalent to prime algebraic Scott domains, which in turn generalize prime algebraic lattices, already known to provide a model of classical linear logic.

1 Introduction

The ccc of Scott domains and continuous functions, which we call **SD**, is the paradigmatic framework for denotational semantics of programming languages. In that area, much effort has been spent in studying more "concrete" structures for representing domains.

At the end of the 70's G. Kahn and G. D. Plotkin [17] developed a theory of concrete domains together with a representation of them in terms of concrete data structures. In the early 80's G. Berry and P.-L. Curien [6] defined a ccc of concrete data structures and sequential algorithms on them. At the same time Scott [20] also developed a representation theory for Scott domains which led him to the definition of information systems; these structures, together with the so-called approximable relations, form the ccc **Inf**, which is equivalent to **SD**.

So it was clear that many categories of "higher-level" structures such as domains had equivalent descriptions in terms of "lower-level" structures, such as concrete data structures and information systems, which are collectively called webs.

At the end of the 80's, J.-Y. Girard [12] discovered linear logic starting from a semantical investigation of the second-order lambda calculus. His seminal work on the semantics of linear logic proofs [13, 14], introduced a category of webs, the coherence spaces, equivalent to the category of coherent qualitative domains and stable maps between them. Coherence spaces form a $*$-autonomous and thus a model of classical Linear Logic (LL).

From the early 90's on, there has been a wealth of categorical models of linear logic, arising from different areas: we mention here S. Abramsky and R. Jagadeesan's games [1], Curien's sequential data structures [10], G. Winskel's event structures [21, 24], and Winskel and Plotkin's bistructures [19] whose associated co-Kleisli category is equivalent to a full-sub-ccc of Berry's category of bidomains [5].

Remarkably, all the above-mentioned models lie outside Scott semantics.

Despite the observation made by M. Barr's in 1979 [2] that the category of complete lattices and linear maps is $*$-autonomous, it was a common belief in the Linear Logic community that the standard

Submitted to:
LINEARITY 2009

Scott semantics could not provide models of classical LL, until 1994, when M. Huth showed [15] that the category **PAL** of prime-algebraic complete lattices and lub-preserving maps is $*$-autonomous and its associated ccc $\mathbf{PAL}^!$ (the co-Kleisli category of the "!" comonad) is a full-sub-ccc of **Cpo**. A few years later, Winskel rediscovered the same model in a semantical investigation of concurrency [22, 23]: indeed he showed that the category **ScottL** whose objects are preordered sets and the morphisms are functions from downward closed to downward closed subsets which preserve arbitrary unions is $*$-autonomous; this category is equivalent to Huth's. T. Ehrhard [11] continues this investigation and shows that the extensional collapse of the category $\mathbf{Rel}^!$, where ! is a comonad based on multi-sets over the category **Rel** of sets and relations, is the category $\mathbf{ScottL}^!$ and that both are new-Seely categories (in the sense of Bierman [7]).

Summing up, there are several categorical models of LL in Scott semantics. In this paper we provide a representation of these models as webs. Our starting point are information systems, of which we provide a linear variant together with linear approximable relations: such data form the category **InfL**, which we prove to be a symmetric monoidal closed category; **InfL** is equivalent to the category **PSD** of prime algebraic Scott domains and has as full-sub-categories **InfLFull** (equivalent to the category **PAL**) and, ultimately, **Rel**.

We define a comonad ! over **InfL**, based on sets rather than multi-sets, which makes **InfL** a new-Seely category and hence a model of intuitionistic MELL: our approach is different from that of [11] in that our comonad is not an endofunctor of **Rel**; we don't need to consider multisets exactly because we work in the bigger category **InfL**. We also notice that **InfLFull** is the largest $*$-autonomous full-subcategory of **InfL** and that $\mathbf{InfL}^!$ is a full sub-ccc of **Inf**.

2 The category of linear informations systems

Let A be a set. We adopt the following conventions: letters $\alpha,\beta,\gamma,\ldots$ are used for elements of A; letters $a,b,c,\ldots$ are used for elements of $\mathscr{P}_{\mathrm{f}}(A)$; letters $x,y,z,\ldots$ are used for arbitrary elements of $\mathscr{P}(A)$.

Definition 1. *A* linear information system *(LIS, for short) is a triple* $\mathscr{A}=(A,\mathrm{Con},\vdash)$*, where* $\mathrm{Con}\subseteq\mathscr{P}_{\mathrm{f}}(A)$ *contains all singletons and* $\vdash\subseteq A\times A$ *satisfies the axioms listed below.*

(IS1) *if* $a\in\mathrm{Con}$ *and* $\forall\beta\in b.\exists\alpha\in a.\ \alpha\vdash\beta$*, then* $b\in\mathrm{Con}$

(IS2) $\alpha\vdash\alpha$

(IS3) *if* $\alpha\vdash\beta\vdash\gamma$*, then* $\alpha\vdash\gamma$

The set A is called the *web* of $\mathscr{A}$ and its elements are called *tokens*. Let $\mathscr{A}$, $\mathscr{B}$ be two LISs. A relation $R\subseteq A\times B$ is *linear approximable* if for all $\alpha,\alpha'\in A$ and all $\beta,\beta'\in B$:

(AR1) if $a\in\mathrm{Con}_A$ and $\forall\beta\in b.\exists\alpha\in a.\ (\alpha,\beta)\in R$, then $b\in\mathrm{Con}_B$

(AR2) if $\alpha'\vdash_A\alpha\ R\ \beta\vdash_B\beta'$, then $(\alpha',\beta')\in R$

Linear approximable relations compose as usual: $S\circ R=R;S=\{(\alpha,\gamma)\in A\times C:\exists\beta\in B.\ \alpha\ R\ \beta\ S\ \gamma\}$.

We call **InfL** the category with LISs as objects and linear approximable relations as morphisms. We reserve the name **InfLFull** for the full-subcategory of **InfL** whose objects are exactly those LISs $\mathscr{A}$ for which $\mathrm{Con}_A=\mathscr{P}_{\mathrm{f}}(A)$. It is not difficult to see that the category **Rel** of sets and relations is a full-subcategory of **InfLFull**, consisting of exactly those LISs $\mathscr{A}$ for which $\mathrm{Con}_A=\mathscr{P}_{\mathrm{f}}(A)$ and $\vdash_A=\{(\alpha,\alpha):\alpha\in A\}$.

2.1 The cartesian structure of InfL

Definition 2. *Let $\mathscr{A}_1$, $\mathscr{A}_2$ be LISs. Define the LIS $\mathscr{A}_1 \& \mathscr{A}_2 = (A_1 \& A_2, \mathrm{Con}, \vdash)$ as follows:*

- $A_1 \& A_2 = A_1 \uplus A_2$
- $\{(i_1,\gamma_1),\dots,(i_m,\gamma_m)\} \in \mathrm{Con}$ *iff* $\{\gamma_j : j \in [1,m],\ i_j = 1\} \in \mathrm{Con}_{A_1}$ *and* $\{\gamma_j : j \in [1,m],\ i_j = 2\} \in \mathrm{Con}_{A_2}$
- $(i,\gamma) \vdash (j,\gamma')$ *iff* $i = j$ *and* $\gamma \vdash_{A_i} \gamma'$

The projections $\pi_i \in \mathbf{InfL}(\mathscr{A}_1 \& \mathscr{A}_2, \mathscr{A}_i)$ are given by $\pi_i = \{((i,\gamma),\gamma') : \gamma \vdash_{A_i} \gamma'\}$, $i = 1,2$. For $R \in \mathbf{InfL}(\mathscr{C}, \mathscr{A}_1)$ and $S \in \mathbf{InfL}(\mathscr{C}, \mathscr{A}_2)$, the pairing $\langle R,S\rangle \in \mathbf{InfL}(\mathscr{C}, \mathscr{A}_1 \& \mathscr{A}_2)$ is given by $\langle R,S\rangle = \{(\gamma,(1,\alpha)) : (\gamma,\alpha) \in R\} \cup \{(\gamma,(2,\beta)) : (\gamma,\beta) \in S\}$. Define $\top = (\emptyset,\emptyset,\emptyset)$. The LIS $\top$ is the terminal object of **InfL**, since for any LIS $\mathscr{A}$, the only morphism $R \in \mathbf{InfL}(\mathscr{A}, \top)$ is $\emptyset$.

2.2 The monoidal closed structure of InfL

Definition 3. *Let $\mathscr{A}$, $\mathscr{B}$ be LISs. Define the LIS $\mathscr{A} \otimes \mathscr{B} = (A \otimes B, \mathrm{Con}, \vdash)$ as follows:*

- $A \otimes B = A \times B$
- $\{(\alpha_1,\beta_1),\dots,(\alpha_m,\beta_m)\} \in \mathrm{Con}$ *iff* $\{\alpha_1,\dots,\alpha_m\} \in \mathrm{Con}_A$ *and* $\{\beta_1,\dots,\beta_m\} \in \mathrm{Con}_B$
- $(\alpha,\beta) \vdash (\alpha',\beta')$ *iff* $\alpha \vdash_A \alpha'$ *and* $\beta \vdash_B \beta'$

For $R \in \mathbf{InfL}(\mathscr{A},\mathscr{C})$ and $S \in \mathbf{InfL}(\mathscr{B},\mathscr{D})$, $R \otimes S \in \mathbf{InfL}(\mathscr{A} \otimes \mathscr{B}, \mathscr{C} \otimes \mathscr{D})$ is given by $R \otimes S = \{((\alpha,\beta),(\gamma,\delta)) \in (A \otimes B) \times (C \otimes D) : (\alpha,\gamma) \in R \text{ and } (\beta,\delta) \in S\}$. It is easy to check that $\_ \otimes \_ : \mathbf{InfL} \times \mathbf{InfL} \to \mathbf{InfL}$ is a bifunctor and that it is a symmetric tensor product, with natural isomorphisms

- $\phi^{\otimes}_{\mathscr{A},\mathscr{B},\mathscr{C}} : \mathscr{A} \otimes (\mathscr{B} \otimes \mathscr{C}) \to (\mathscr{A} \otimes \mathscr{B}) \otimes \mathscr{C}$ given by

$$\phi^{\otimes}_{\mathscr{A},\mathscr{B},\mathscr{C}} = \{((\alpha,(\beta,\gamma)),((\alpha',\beta'),\gamma')) : \alpha \vdash_A \alpha',\ \beta \vdash_B \beta',\ \gamma \vdash_C \gamma'\}$$

- $\sigma^{\otimes}_{\mathscr{A},\mathscr{B}} : \mathscr{A} \otimes \mathscr{B} \to \mathscr{B} \otimes \mathscr{A}$ given by $\sigma^{\otimes}_{\mathscr{A},\mathscr{B}} = \{((\alpha,\beta),(\beta',\alpha')) : \alpha \vdash_A \alpha',\ \beta \vdash_B \beta'\}$
- $\rho^{\otimes}_{\mathscr{A}} : \mathscr{A} \otimes \mathbf{1} \to \mathscr{A}$ given by $\rho^{\otimes}_{\mathscr{A}} = \{((\alpha,*),\alpha') : \alpha \vdash_A \alpha'\}$
- $\lambda^{\otimes}_{\mathscr{A}} : \mathbf{1} \otimes \mathscr{A} \to \mathscr{A}$ given by $\lambda^{\otimes}_{\mathscr{A}} = \{((*,\alpha),\alpha') : \alpha \vdash_A \alpha'\}$

Define $\mathbf{1} = (\{*\},\{\emptyset,\{*\}\},\{(*,*)\})$. The LIS $\mathbf{1}$ is the unit of the tensor product.
The above data make **InfL** a symmetric monoidal category.

Definition 4. *Let $\mathscr{A}$, $\mathscr{B}$ be LISs. Define the LIS $\mathscr{A} \multimap \mathscr{B} = (A \multimap B, \mathrm{Con}, \vdash)$ as follows:*

- $A \multimap B = A \times B$
- $\{(\alpha_1,\beta_1),\dots,(\alpha_m,\beta_m)\} \in \mathrm{Con}$ *iff for all* $J \subseteq [1,m]$, $\{\alpha_j : j \in J\} \in \mathrm{Con}_A$ *implies* $\{\beta_j : j \in J\} \in \mathrm{Con}_B$
- $(\alpha,\beta) \vdash (\alpha',\beta')$ *iff* $\alpha' \vdash_A \alpha$ *and* $\beta \vdash_B \beta'$

Define a natural isomorphism $\mathrm{cur}_l : \mathbf{InfL}(\mathscr{C} \otimes \mathscr{A}, \mathscr{B}) \to \mathbf{InfL}(\mathscr{C}, \mathscr{A} \multimap \mathscr{B})$, (the linear currying) as follows: $\mathrm{cur}_l = \{(((\gamma,\alpha),\beta),(\gamma',(\alpha',\beta'))) : \alpha' \vdash_A \alpha,\ \beta \vdash_B \beta',\ \gamma' \vdash_C \gamma\}$. Define also the (linear) evaluation morphism $\mathrm{ev}_l : \mathscr{A} \otimes (\mathscr{A} \multimap \mathscr{B}) \to \mathscr{B}$ as $\mathrm{ev}_l = \{((\alpha,(\alpha',\beta)),\beta') : \alpha \vdash_A \alpha',\ \beta \vdash_B \beta'\}$.

Te above data make **InfL** a symmetric monoidal closed category, and thus a model of intuitionistic MLL proofs. In such a model, the connectives $\parr$ and $\otimes$ coincide, and thus the respective units $\bot$ and $\mathbf{1}$ are equal. So we take $\bot = (\{*\},\{\emptyset,\{*\}\},\{(*,*)\})$.

We now briefly discuss the issue of duality in the category **InfL**. Let $\mathscr{A}$ be a LIS and consider the LIS $\mathscr{A} \multimap \bot$; an explicit description of the latter object is as follows: $A \multimap \bot = A \times \{*\}$, $\mathrm{Con}_{A\multimap\bot} = \mathscr{P}_{\mathrm{f}}(A \times \{*\})$, and $(\alpha,*) \vdash_{A\multimap\bot} (\beta,*)$ iff $\beta \vdash_A \alpha$.

Therefore $\bot$ is not a dualizing object in **InfL**, but it is so in **InfLFull**, where the family of arrows $\partial_{\mathscr{A}} : \mathscr{A} \to (\mathscr{A} \multimap \top) \multimap \top$ defined by $\partial_{\mathscr{A}} = \{(\alpha,((\alpha',*),*)) : \alpha \vdash_A \alpha'\}$, for each LIS $\mathscr{A}$, is a natural isomorphism. In other words **InfLFull** is the largest $*$-autonomous full-subcategory of **InfL**.

2.3 InfL is a new-Seely category

In this section we define a comonad ! over **InfL** and prove that it gives a symmetric strong monoidal functor. Finally we prove that **InfL** is a new-Seely category and thus a model of intuitionistic MELL proofs, in which the exponential modality ! has a set-theoretic interpretation. For categorical notions we refer to Melliès [18].

Definition 5. *Let $\mathscr{A}$ be an IS. Define the IS $!\mathscr{A} = (!A, \mathrm{Con}, \vdash)$ as follows:*

- $!A = \mathrm{Con}_A$
- $\{a_1,\ldots,a_k\} \in \mathrm{Con}$ *iff* $\cup_{i=1}^k a_i \in \mathrm{Con}_A$
- $a \vdash b$ *iff* $\forall \beta \in b. \exists \alpha \in a.\ \alpha \vdash_A \beta$

Let $R \in \mathbf{InfL}(\mathscr{A},\mathscr{B})$. Define $!R \in \mathbf{InfL}(!\mathscr{A},!\mathscr{B})$ as follows: $!R = \{(a,b) \in !A \times !B : \forall \beta \in b. \exists \alpha \in a.\ (\alpha,\beta) \in R\}$. It is an easy matter to verify that $!(\_) : \mathbf{InfL} \to \mathbf{InfL}$ is a functor. Moreover ! is a comonad with digging $\delta :! \Rightarrow !!$ defined by $\delta_{\mathscr{A}} = \{(b,Y) \in !A \times !!A : b \vdash_{!A} \cup Y\}$ and dereliction $\varepsilon :! \Rightarrow \mathsf{id}_{\mathbf{InfL}}$ defined by $\varepsilon_{\mathscr{A}} = \{(b,\beta) \in !A \times A : b \vdash_{!A} \{\beta\}\}$.

The forthcoming lemma shows that ! is a symmetric strong monoidal functor. Before proving this, we shall explicit the symmetric monoidal structure $(\mathbf{InfL}, \&, \top)$ involved in the proof:

- $\phi^{\&}_{\mathscr{A},\mathscr{B},\mathscr{C}} : \mathscr{A}\&(\mathscr{B}\&\mathscr{C}) \to (\mathscr{A}\&\mathscr{B})\&\mathscr{C}$ given by
$$\begin{aligned}\phi^{\&}_{\mathscr{A},\mathscr{B},\mathscr{C}} \quad = \quad & \big\{\big((1,\alpha),(1,(1,\alpha'))\big) : \alpha \vdash_A \alpha'\big\} \cup \big\{\big((2,(1,\beta)),(1,(2,\beta'))\big) : \beta \vdash_B \beta'\big\} \cup \\ & \cup \big\{\big((2,(2,\gamma)),(2,\gamma')\big) : \gamma \vdash_C \gamma'\big\}\end{aligned}$$
- $\sigma^{\&}_{\mathscr{A},\mathscr{B}} : \mathscr{A}\&\mathscr{B} \to \mathscr{B}\&\mathscr{A}$ given by $\sigma^{\&}_{\mathscr{A},\mathscr{B}} = \big\{\big((1,\alpha),(2,\alpha')\big) : \alpha \vdash_A \alpha'\big\} \cup \big\{\big((2,\beta),(1,\beta')\big) : \beta \vdash_B \beta'\big\}$
- $\rho^{\&}_{\mathscr{A}} : \mathscr{A}\&\top \to \mathscr{A}$ given by $\rho^{\&}_{\mathscr{A}} = \big\{\big((1,\alpha),\alpha'\big) : \alpha \vdash_A \alpha'\big\}$
- $\lambda^{\&}_{\mathscr{A}} : \top\&\mathscr{A} \to \mathscr{A}$ given by $\lambda^{\&}_{\mathscr{A}} = \big\{\big((2,\alpha),\alpha'\big) : \alpha \vdash_A \alpha'\big\}$

Lemma 6. *The functor $! : (\mathbf{InfL}, \&, \top) \to (\mathbf{InfL}, \otimes, \mathbf{1})$ is symmetric strong monoidal.*

Proof. We give the natural isomorphisms $\mathsf{m}_{\mathscr{A},\mathscr{B}} :!\mathscr{A} \otimes !\mathscr{B} \cong !(\mathscr{A}\&\mathscr{B})$ and $\mathsf{n} : \mathbf{1} \cong !\top$ making ! a symmetric strong monoidal functor. Define $\mathsf{n} : \mathbf{1} \to !\top$ as $\mathsf{n} = \{(*,\{\emptyset\})\}$ and $\mathsf{m}_{\mathscr{A},\mathscr{B}} :!\mathscr{A} \otimes !\mathscr{B} \to !(\mathscr{A}\&\mathscr{B})$ as $\mathsf{m}_{\mathscr{A},\mathscr{B}} = \big\{\big((a,b),\{(1,\alpha') : \alpha' \in a'\} \cup \{(2,\beta') : \beta' \in b'\}\big) : a,a' \in !A,\ b,b' \in !B,\ a \vdash_{!A} a',\ b \vdash_{!B} b'\big\}$.

We now proceed by verifying the commutation of the required diagrams. First observe that both $\mathsf{m}_{\mathscr{A},\mathscr{B}} \otimes \mathsf{id}_{!\mathscr{C}}; \mathsf{m}_{\mathscr{A}\&\mathscr{B},\mathscr{C}}; !\phi^{\&}_{\mathscr{A},\mathscr{B},\mathscr{C}}$ and $\phi^{\otimes}_{\mathscr{A},\mathscr{B},\mathscr{C}}; \mathsf{id}_{!\mathscr{A}} \otimes \mathsf{m}_{\mathscr{B},\mathscr{C}}; \mathsf{m}_{\mathscr{A},\mathscr{B}\&\mathscr{C}}$ are equal to the set of all pairs

$$\Big(((a,b),c),\{(1,(1,\alpha_1)),\ldots,(1,(1,\alpha_{n_1})),(2,(1,\beta_1)),\ldots,(2,(1,\beta_{n_2})),(2,(2,\gamma_1)),\ldots,(2,(2,\gamma_{n_3}))\}\Big)$$

where $a \vdash_{!A} \{\alpha_1,\ldots,\alpha_{n_1}\}$, $b \vdash_{!B} \{\beta_1,\ldots,\beta_{n_2}\}$, and $c \vdash_{!C} \{\gamma_1,\ldots,\gamma_{n_3}\}$.

Finally the "units" and the "symmetry" diagrams all commute because

- $\mathsf{id}_{!\mathscr{A}} \otimes \mathsf{n}; \mathsf{m}_{\mathscr{A},\top}; !\rho^{\&}_{\mathscr{A}} = \{((a,*),a') : a \vdash_{!A} a'\} = \rho^{\otimes}_{!\mathscr{A}}$,
- $\mathsf{n} \otimes \mathsf{id}_{!\mathscr{B}}; \mathsf{m}_{\top,\mathscr{B}}; !\lambda^{\&}_{\mathscr{B}} = \{((*,b),b') : b \vdash_{!B} b'\} = \lambda^{\otimes}_{!\mathscr{B}}$,
- both $\sigma^{\otimes}_{!\mathscr{A},!\mathscr{B}}; \mathsf{m}_{\mathscr{B},\mathscr{A}}$ and $\mathsf{m}_{\mathscr{A},\mathscr{B}}; !\sigma^{\&}_{\mathscr{A},\mathscr{B}}$ equal the morphism $\big\{\big((a,b),\{(1,\beta') : \beta' \in b'\} \cup \{(2,\alpha') : \alpha' \in a'\}\big) : b' \in !B,\ a' \in !A,\ b \vdash_{!B} b',\ a \vdash_{!A} a'\big\}$.

□

Proposition 7. **InfL** *is a new-Seely category.*

Proof. By Lemma 6, ! is a symmetric strong monoidal functor; it remains to check the coherence diagram in the definition of new-Seely category. Indeed we have:

$$\begin{aligned}\delta_{\mathscr{A}} \otimes \delta_{\mathscr{B}}; \mathsf{m}_{!\mathscr{A},!\mathscr{B}} &= \big\{\big((a,b),\{(1,a') : a' \in X'\} \cup \{(2,b') : b' \in Y'\}\big) : a \vdash_{!A} \cup X',\ b \vdash_{!B} \cup Y'\big\} \\ &= \mathsf{m}_{\mathscr{A},\mathscr{B}}; \delta_{\mathscr{A}\&\mathscr{B}}; !\langle !\pi_1, !\pi_2\rangle\end{aligned}$$

□

2.4 A representation theorem

Not surprisingly, **InfL** turns out to be equivalent to the category of prime algebraic Scott domains and linear continuous functions. In this section, we outline this equivalence.

Definition 8. *An element p of a Scott domain $\mathscr{D}$ is* prime *if, whenever $B \subseteq_{\mathrm{f}} D$ is upper bounded and $p \leq \vee B$, there exists $b \in B$ such that $p \leq b$. The domain $\mathscr{D}$ itself is* prime algebraic *if every element of D is the least upper bound of the set of prime elements below it. The set of prime elements of $\mathscr{D}$ is denoted by $\mathscr{PR}(D)$.*

A *linear function* between two prime algebraic Scott domains is a Scott continuos function that commutes with all (existing) least upper bounds. The category of prime algebraic Scott domains and linear function is denoted by **PSD**.

A *point* of an information system $\mathscr{A}$ is an element of $\mathbf{InfL}^{!}(\top,\mathscr{A})$. Up to isomorphisms, $\mathbf{InfL}^{!}(\top,\mathscr{A})$ can be described as the set of all the subsets $x \subseteq A$ satisfying the following two properties:

(PT1) if $u \subseteq_{\mathrm{f}} x$ then $u \in \mathrm{Con}$ (x is *finitely consistent*)

(PT2) if $\alpha \in x$ and $\alpha \vdash \alpha'$, then $\alpha' \in x$ (x is *closed w.r.t.* $\vdash$)

We are now able to relate linear information systems to the corresponding categories of domains.

Definition 9. *Given $f,\mathscr{D},\mathscr{E},R,\mathscr{A},\mathscr{B}$ such that $f \in \mathbf{PSD}(\mathscr{D},\mathscr{E})$ and $R \in \mathbf{InfL}(\mathscr{A},\mathscr{B})$, we define:*

- *$\mathscr{A}^{+}$ is the set of points of $\mathscr{A}$ ordered by inclusion.*
- $R^{+}(x) = \{\beta \in B \mid \exists \alpha \in x.\ (\alpha,\beta) \in R\}$
- $\mathscr{D}^{-} = (\mathscr{PR}(D), \mathrm{Con}, \vdash)$ *where $a \in \mathrm{Con}$ iff a is upper bounded and $p \vdash p'$ iff $p' \leq_{\mathscr{D}} p$*
- $f^{-} = \{(p,p') \in \mathscr{PR}(D) \times \mathscr{PR}(E) \mid f(p) \geq_{\mathscr{D}} p'\}$

It is an easy matter so check that $(\_)^{+} : \mathbf{InfL} \to \mathbf{PSD}$ and $,(\_)^{-} : \mathbf{PSD} \to \mathbf{InfL}$ are indeed functors.

Theorem 10. *The functors $(\_)^{+},(\_)^{-}$ define an equivalence between the categories* **InfL** *and* **PSD**.

This equivalence specializes to an equivalence between **InfLFull** and the category **PAL** of prime algebraic lattices and linear continuos functions.

As usual, we denote by $\mathbf{InfL}^{!}$ the co-Kleisli category of the comonad ! over **InfL** and by **Inf** we denote the category of Scott information systems and approximable relations. The next proposition shows that again, following a well-established pattern, the comonad ! allows to recover non-linear approximable relations form linear ones i.e., in view of the categorical equivalence just stated, to recover Scott-continuous functions from linear ones.

Proposition 11. $\mathbf{InfL}^!$ *is a full-sub-ccc of* **Inf**.

Proof. Clearly $\mathbf{InfL}(!\mathscr{A},\mathscr{B}) \cong (!\mathscr{A} \multimap \mathscr{B})^+ \cong (\mathscr{A} \Rightarrow \mathscr{B})^+ \cong \mathbf{Inf}(\mathscr{A},\mathscr{B})$, where $\mathscr{A} \Rightarrow \mathscr{B}$ is the exponential object in the category **Inf** (see [20], for example). □

3 Conclusions and future work

In this paper we defined the category **InfL**, whose objects and arrows result from a linearization of Scott's information systems. We show this category to be symmetric monoidal closed and thus a model of MLL. We moreover prove that **InfL** is a new-Seely category, with a "set-theoretic" interpretation of exponentials via a comonad !; this is made possible by the presence of the entailment relation, which is always non-trivial in objects of the form $!\mathscr{A}$: even if the entailment $\vdash_A$ is the equality we have that $\mathbf{InfL}(!\mathscr{A},\mathscr{A}) \subset \mathscr{P}(!\mathscr{A} \times \mathscr{A})$ and this rules out Ehrhard's counterexample for the naturality of dereliction. In the purely relational model of classical MELL, **Rel**, the use of multisets is needed.

Indeed a comonad based on multi-sets, let's say †, can be defined in our framework too, yelding a different co-Kleisli category. A similar situation arises in the framework of the coherence spaces model of LL. In that case Barreiro and Ehrhard [3] proved that the extensional collapse of the hierarchy of simple types associated to the multi-set interpretation *is* the hierarchy associated to the "set" interpretation. This means in particular that, as models of the simply typed λ-calculus, the former discriminates finerly than the latter, being sensitive, for instance, to the number of occurrences of a variable in a term. It is likely that, in a similar way, $\mathbf{InfL}^!$ is the extensional collapse of $\mathbf{InfL}^\dagger$.

The categories **InfL** and **InfLFull** may be themselves compared using the same paradigm. Trivializing the consitency relation boils down to add points to the underlying domains; is that another instance of extensional collapse situation? In the case of the simple types hierarchy over the booleans, the Scott model is actually the extensional collapse of the lattice-theoretic one [9].

Summing up it appears that, by tuning the linear information systems in different ways, one obtains different frameworks for the interpretation of proofs, whose inter-connections remain to be investigated.

This is, in our opinion, the main advantage of the approach, with respect to the existing descriptions of the Scott continuous models of linear logic [2, 15, 16, 22].

Linear information systems are close to several classes of webbed models of the pure the λ-calculus: they generalize Berline's *preordered sets with coherence* [4], where a set of tokens is consistent if and only if its elements are pairwise coherent. We plan to investigate whether such a generalisation is useful for studying the models of the λ-calulus in **PSD**. Actually, one of our motivations was to settle a representation theory for a larger class of cartesian closed categories, whith $\mathbf{Rel}^!$ as a particular case, in order to provide a tool for investigating "non-standard"[1] models. We get **Rel** as a full subcategory of **InfL**, but the bunch of axioms on information-like structures making $\mathbf{Rel}^!$ an instance of the co-Kleisli construction remains to be found.

Another original motivation for investigating linear information systems, that we are pursuing, was the definition of a framework suitable for the interpretation of Boudol's λ-calculus with resources [8].

[1] Let us call "standard" a model of the λ-calculus that is an instance of one of the "main" semantics: continuous, stable, strongly stable.

References

[1] S. Abramsky & R. Jagadeesan (1992): *Games and Full Completeness for Multiplicative Linear Logic (Extended Abstract)*. In: *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer-Verlag, London, UK, pp. 291–301.

[2] M. Barr (1979): *$*$-autonomous categories*. Number 752 in LNCS. Springer-Verlag.

[3] N. Barreiro & T. Ehrhard (1997). *Anatomy of an extensional collapse*.

[4] C. Berline (2000): *From computation to foundations via functions and application: The λ-calculus and its webbed models*. *Theor. Comput. Sci.* 249(1), pp. 81–161.

[5] G. Berry (1979): *Modèles complétement adequats et stables des lambda-calculs typés*. Thèse de doctorat d' État, Université de Paris VII.

[6] G. Berry & P.-L. Curien (1982): *Sequential algorithms on concrete data structures*. *Theor. Comput. Sci.* 20, pp. 265–321.

[7] G. M. Bierman (1995): *What is a Categorical Model of Intuitionistic Linear Logic?* In: *TLCA '95: Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*. Springer-Verlag, London, UK, pp. 78–93.

[8] G. Boudol (1993): *The Lambda-Calculus with Multiplicities (Abstract)*. In: *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*. Springer-Verlag, London, UK, pp. 1–6.

[9] A. Bucciarelli (1996): *Logical relations and lambda theories*. In: *Advances in Theory and Formal Methods of Computing, proceedings of the 3rd Imperial College Workshop*. pp. 37–48.

[10] P.-L. Curien (1994): *On the Symmetry of Sequentiality*. In: *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*. Springer-Verlag, London, UK, pp. 29–71.

[11] T. Ehrhard (2009). *The Scott model of Linear Logic is the extensional collapse of its relational model*.

[12] J.-Y. Girard (1986): *The system F of variable types, fifteen years later*. *Theor. Comput. Sci.* 45(2), pp. 159–192.

[13] J.-Y. Girard (1987): *Linear logic*. *Theor. Comput. Sci.* 50(1), pp. 1–102.

[14] J.-Y. Girard (1988): *Normal functors, power series and the lambda-calculus*. *Annals of pure and applied logic* 37(2), pp. 129–177.

[15] M. Huth (1994): *Linear Domains and Linear Maps*. In: *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*. Springer-Verlag, London, UK, pp. 438–453.

[16] M. Huth, A. Jung & K. Keimel (1994): *Linear Types, Approximation, and Topology*. In: *LICS*. pp. 110–114.

[17] G. Kahn & G. D. Plotkin (1978): *Domaines Concrets*. Technical Report Rapport 336, IRIA-LABORIA.

[18] P. A. Melliès (2008). *Categorical semantics of linear logic*.

[19] G. D. Plotkin & G. Winskel (1994): *Bistructures, Bidomains and Linear Logic*. In: *ICALP '94: Proceedings of the 21st International Colloquium on Automata, Languages and Programming*. Springer-Verlag, London, UK, pp. 352–363.

[20] D. S. Scott (1982): *Domains for Denotational Semantics*. In: *Proceedings of the 9th Colloquium on Automata, Languages and Programming*. Springer-Verlag, London, UK, pp. 577–613.

[21] G. Winskel (1988): *An introduction to event structures*. In: *REX Workshop*. pp. 364–397.

[22] G. Winskel (1999): *A Linear Metalanguage for Concurrency*. In: *AMAST '98: Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology*. Springer-Verlag, London, UK, pp. 42–58.

[23] G. Winskel (2004): *Linearity and non linearity in distributed computation*. In: *T. Ehrhard and J.-Y. Girard and P. Ruet and P. Scott, ed. Linear Logic in Computer Science, London Mathematical Society Lecture Note Series* 316. Cambridge University Press.

[24] G.-Q. Zhang (1992): *A Monoidal Closed Category of Event Structures*. In: *Proceedings of the 7th International Conference on Mathematical Foundations of Programming Semantics*. Springer-Verlag, London, UK, pp. 426–435.

Uniqueness Typing for Resource Management in Message-Passing Concurrency (Extended Abstract)

Adrian Francalanza
University of Malta
Adrian.Francalanza@um.edu.mt

Edsko de Vries* Matthew Hennessy*
Trinity College Dublin, Ireland
{Edsko.de.Vries,Matthew.Hennessy}@cs.tcd.ie

We view channels as the main form of resources in a message-passing programming paradigm. These channels need to be carefully managed in settings where resources are scarce. To study this problem, we extend the pi-calculus with primitives for channel allocation and deallocation and allow channels to be reused to communicate values of different types. Inevitably, the added expressivity increases the possibilities for runtime errors. We define a substructural type system which combines uniqueness typing and affine typing to reject these ill-behaved programs.

1 Introduction

Message-passing concurrency is a programming paradigm whereby shared memory is prohibited and process interaction is limited to explicit message communication. This concurrency paradigm forms the basis for a number of process calculi such as the pi-calculus [11] and has been adopted by programming languages such as the actor based language Erlang [3].

Message-passing concurrency often abstracts away from resource management and programs written at this abstraction level exhibit poor resource awareness. In this paper we study ways of improving this shortcoming. Specifically, we develop a statically typed extension of the pi-calculus in which resources, i.e. channels, can be reused at varying types and unwanted resources can be safely deallocated.

Idiomatic pi-calculus processes are often characterized by wasteful use-once-throw-away channels [11, 10]. Consider the following two pi-calculus process definitions

$$\text{TIMESRV} \triangleq \mathsf{rec}X.\mathit{getTime}?x.x!\langle \mathit{time}\rangle.X \qquad \text{DATESRV} \triangleq \mathsf{rec}X.\mathit{getDate}?x.x!\langle \mathit{date}\rangle.X$$

TIMESRV defines a server that repeatedly waits on a channel named *getTime* to dynamically receive a channel name, represented by the bound variable x, and then replies with the current time on x. DATESRV is a similar service which returns the current date. An idiomatic pi-calculus client definition is

$$\text{CLIENT}_0 \triangleq (\nu \mathit{ret}_1)\,\mathit{getTime}!\langle \mathit{ret}_1\rangle.\mathit{ret}_1?y.(\nu \mathit{ret}_2)\,\mathit{getDate}!\langle \mathit{ret}_2\rangle.\mathit{ret}_2?z.P$$

CLIENT_0 uses two distinct channels ret_1 and ret_2 as return channels to query the time and date servers, and then continues as process P with the values obtained. The return channels are scoped (private) to preclude interference from other clients concurrently querying the servers. From a resource management perspective, there are a number of shortcomings with CLIENT_0.

First, it makes pragmatic sense to reduce the number of channels used and to *reuse* channel ret_1. CLIENT_1 does this when querying DATESRV on channel *getDate*, even if this reuse entails *strong updates*

*The financial support of SFI is gratefully acknowledged.

$$
\begin{array}{lllllllll}
P,Q ::= & u!\vec{v}.P & \textit{(output)} & | & u?\vec{x}.P & \textit{(input)} & | & \mathsf{nil} \quad \textit{(nil)} & | \quad \mathsf{if}\ u=v\ \mathsf{then}\ P\ \mathsf{else}\ Q \quad \textit{(match)} \\
| & \mathsf{rec}X.P & \textit{(recursion)} & | & X & \textit{(proc. variable)} & | & P \| Q \quad \textit{(parallel)} & | \quad (\nu c{:}s)P \quad \textit{(stateful scoping)} \\
| & \mathsf{alloc}(x).P & \textit{(allocate)} & | & \mathsf{free}\,u.P & \textit{(deallocate)} & & &
\end{array}
$$

Figure 1: Polyadic resource pi-calculus syntax

i.e., values with unrelated types (times and dates) being sent on ret_1. Intuitively, this reuse is safe because CLIENT$_1$ is the only client using ret_1 and TIMESRV does not use ret_1 more than once.

$$\text{CLIENT}_1 \triangleq (\nu ret_1)\, getTime!\langle ret_1\rangle.ret_1?y.\, getDate!\langle ret_1\rangle.ret_1?z.P$$

The second shortcoming is more subtle and stems from the fact that the pi-calculus is ambiguous with respect to the resource semantics of its scoping operator. More precisely, it is unclear whether in CLIENT$_0$ channel ret_2 is allocated before or after the input on ret_1. The relevance of this ambiguity becomes more acute when scoping occurs inside recursive definitions. We can address this problem by introducing an explicit allocation construct, distinct from scoping, which delineates precisely when a channel is allocated; scoping is then reserved only for name-binding bookkeeping.

The third shortcoming we consider is that CLIENT$_0$ does not deallocate the channels when it no longer needs them. If we add both channel allocation and deallocation primitives, we can rewrite the client as CLIENT$_2$:

$$\text{CLIENT}_2 \triangleq \mathsf{alloc}(x).getTime!\langle x\rangle.x?y.getDate!\langle x\rangle.x?z.\mathsf{free}\,x.P$$

Inevitably, the added expressivity of this extended pi-calculus increases the possibilities for runtime errors like value mismatch during communication and usage of channels which have been deallocated.

We define a type system which rejects processes that are unsafe; the type system combines uniqueness typing [4] and affine typing [10], while permitting value coercion across these modes through subtyping. Uniqueness typing gives us *global guarantees*, simplifying local reasoning when typing both strong updates and safe deallocations. Uniqueness typing can be seen as dual to affine typing [7], and we make essential use of this duality to allow uniqueness to be temporarily violated.

2 The Resource Pi-Calculus

Figure 1 shows the syntax for the resource pi-calculus; the language is the standard pi-calculus [1], except that we have added primitives for channel allocation and deallocation, and channel scoping carries the state of the state $s \in \{\top, \bot\}$ for allocated and deallocated channels. Channel allocation is a binder.

Since this language is stateful (channels are allocated or deallocated), the reduction relation is defined over configurations, consisting of a state $\sigma \in \Sigma : \text{CHANS} \rightharpoonup \{\top, \bot\}$ and a process. A tuple $\langle \sigma, P\rangle$ is a configuration whenever $\mathbf{fn}(P) \subseteq \mathbf{dom}(\sigma)$; we denote configurations as $\sigma \triangleright P$.

The reduction relation is defined as the least relation over configurations satisfying the rules in Fig. 2, using the standard pi-calculus structural equivalence relation ($\equiv$). Communication in the resource pi-calculus, RCOM, requires the communicating channel to be allocated. On the other hand, name equality branching, RTHEN and RELSE, permits comparisons on deallocated channels as well. Allocation RALL creates a private allocated channel and substitutes it for the bound variable of the allocation construct in the continuation. Deallocation, RFREE, is the only construct that changes the state of configurations. The contextual rule RREST wraps the state of the private channel as part of the scoping construct.

[1] We distinguish between variables (x, y) and names (c, d) and use identifiers (u, v) to mean either.

Reduction Rules

$$\frac{\sigma(c) = \top}{\sigma \triangleright c!\vec{b} \,\|\, c?\vec{x}.P \longrightarrow \sigma \triangleright P\{\vec{b}/\vec{x}\}}\ \textsc{rCom} \qquad \frac{}{\sigma \triangleright \mathsf{if}\ c=c\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow \sigma \triangleright P}\ \textsc{rThen} \qquad \frac{c \neq d}{\sigma \triangleright \mathsf{if}\ c=d\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow \sigma \triangleright Q}\ \textsc{rElse}$$

$$\frac{}{\sigma \triangleright \mathsf{rec}X.P \longrightarrow \sigma \triangleright P\{\mathsf{rec}X.P/X\}}\ \textsc{rRec} \qquad \frac{c \notin \mathbf{dom}(\sigma)}{\sigma \triangleright \mathsf{alloc}(x).P \longrightarrow \sigma \triangleright (\nu c:\top)\big(P\{c/x\}\big)}\ \textsc{rAll} \qquad \frac{}{\sigma, c:\top \triangleright \mathsf{free}\,c.P \longrightarrow \sigma, c:\bot \triangleright P}\ \textsc{rFree}$$

$$\frac{\sigma \triangleright P \longrightarrow \sigma' \triangleright P'}{\sigma \triangleright P \,\|\, Q \longrightarrow \sigma' \triangleright P' \,\|\, Q}\ \textsc{rPar} \qquad \frac{\sigma, c:s \triangleright P \longrightarrow \sigma', c:s' \triangleright P'}{\sigma \triangleright (\nu c:s)P \longrightarrow \sigma' \triangleright (\nu c:s')P'}\ \textsc{rRest} \qquad \frac{P \equiv P' \quad \sigma \triangleright P' \longrightarrow \sigma' \triangleright Q' \quad Q' \equiv Q}{\sigma \triangleright P \longrightarrow \sigma' \triangleright Q}\ \textsc{rStr}$$

Error Reduction Rules

$$\frac{|\vec{d}| \neq |\vec{x}|}{\sigma \triangleright c!\vec{d}.P \,\|\, c?\vec{x}.Q \xrightarrow{\text{err}}}\ \textsc{esCom1} \qquad \frac{\sigma(c) = \bot}{\sigma \triangleright c!\vec{d}.P \,\|\, c?\vec{x}.Q \xrightarrow{\text{err}}}\ \textsc{esCom2} \qquad \frac{P \equiv P' \quad \sigma \triangleright P' \xrightarrow{\text{err}}}{\sigma \triangleright P \xrightarrow{\text{err}}}\ \textsc{eStr}$$

Figure 2: Reduction Rules and Error Reduction Rules

3 The Type System

The core type of our system is the channel type, $[\overrightarrow{T}]^a$, consisting of an n-ary tuple of types $\overrightarrow{T}$ describing the values carried over the channel and an attribute a which gives usage information about the channel. This attribute can take one of three forms:

- A channel of type $[\overrightarrow{T}]^\omega$ is an *unrestricted* channel; such type assumptions correspond to type assumptions of the form $[\overrightarrow{T}]$ in non-substructural type systems.
- A channel of type $[\overrightarrow{T}]^1$ is *affine*, and comes with an obligation: it can be used at most once.
- A channel of type $[\overrightarrow{T}]^{(\bullet,i)}$ comes with a guarantee that it is *unique* after i actions; we abbreviate the type $[\overrightarrow{T}]^{(\bullet,0)}$ of channels that are unique *now* to $[\overrightarrow{T}]^\bullet$.

Unique channels can be used to describe instances where only one process has access to (owns) a channel. Accordingly, strong update and deallocation is safe for unique channels.

The process typing rules are shown in Figure 3 defining the typing relation $\Gamma \vdash P$. Typing environments, Γ, are unordered lists of tuples of identifiers and types. The relation for configuration typing, $\Gamma \Vdash \sigma \triangleright P$, requires that all type assumptions correspond to allocated channels (TCONF). Additionally, it restricts type environments to partial maps, which is crucial for type soundness[2].

Since the type system is substructural, usage of type assumptions must be carefully controlled; the logical rules do not allow to use an assumption more than once, and operations on the typing environment are described separately by the structural rules. Although the subtyping relation is novel because it combines uniqueness subtyping (SUNQ) with affine subtyping (SAFF) making them *dual* with respect to unrestricted types, the subtyping and weakening structural rules are standard. The other two rules, however, deserve explanation.

[2]Environments that are not partial maps are required further up the type derivation. The reason is rather technical, and is omitted from this extended abstract due to space limitations. It will be explained in the full paper.

Logical rules

$$\frac{\Gamma, u:[\overrightarrow{\mathbf{T}}]^{a-1} \vdash P}{\Gamma, u:[\overrightarrow{\mathbf{T}}]^{a}, \overrightarrow{v:\mathbf{T}} \vdash u!\vec{v}.P}\ \textsc{tOut} \qquad \frac{\Gamma, u:[\overrightarrow{\mathbf{T}}]^{a-1}, \overrightarrow{x:\mathbf{T}} \vdash P \qquad \vec{x} \notin \mathbf{dom}(\Gamma)}{\Gamma, u:[\overrightarrow{\mathbf{T}}]^{a} \vdash u?\vec{x}.P}\ \textsc{tIn}$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}\ \textsc{tIf} \qquad \frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1, \Gamma_2 \vdash P \parallel Q}\ \textsc{tPar}$$

$$\frac{\Gamma^{\omega}, X:\mathbf{proc} \vdash P}{\Gamma^{\omega} \vdash \mathsf{rec}X.P}\ \textsc{tRec} \qquad \frac{}{X:\mathbf{proc} \vdash X}\ \textsc{tVar} \qquad \frac{}{\emptyset \vdash \mathsf{nil}}\ \textsc{tNil} \qquad \frac{\Gamma, c:\mathbf{T} \vdash P}{\Gamma \vdash (\nu c{:}\top)P}\ \textsc{tRst1} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash (\nu c{:}\bot)P}\ \textsc{tRst2}$$

$$\frac{\Gamma, x:[\overrightarrow{\mathbf{T}}]^{\bullet} \vdash P}{\Gamma \vdash \mathsf{alloc}(x).P}\ \textsc{tAll} \qquad \frac{\Gamma \vdash P}{\Gamma, u:[\overrightarrow{\mathbf{T}}]^{\bullet} \vdash \mathsf{free}u.P}\ \textsc{tFree}$$

where Γ^{ω} is an environment containing only unrestricted assumptions.

Structural rules

$$\frac{\Gamma, u:\mathbf{T}_1, u:\mathbf{T}_2 \vdash P \quad \mathbf{T} = \mathbf{T}_1 \circ \mathbf{T}_2}{\Gamma, u:\mathbf{T} \vdash P}\ \textsc{tCon} \qquad \frac{\Gamma \vdash P}{\Gamma, u:\mathbf{T} \vdash P}\ \textsc{tWeak} \qquad \frac{\Gamma, u:\mathbf{T}_2 \vdash P \quad \mathbf{T}_1 \prec_s \mathbf{T}_2}{\Gamma, u:\mathbf{T}_1 \vdash P}\ \textsc{tSub} \qquad \frac{\Gamma, u:[\overrightarrow{\mathbf{T}_2}]^{\bullet} \vdash P}{\Gamma, u:[\overrightarrow{\mathbf{T}_1}]^{\bullet} \vdash P}\ \textsc{tRev}$$

Typing configurations

$$\frac{\forall c \in \mathbf{dom}(\Gamma).\, \sigma(c) = \top \qquad \Gamma \vdash P \qquad \Gamma \text{ is a partial map}}{\Gamma \Vdash \sigma \triangleright P}\ \textsc{tConf}$$

Channel usage

$$\Gamma, c:[\vec{T}]^{a-1} \stackrel{\mathbf{def}}{=} \begin{cases} \Gamma & \text{if } a = 1 \\ \Gamma, c:[\vec{T}]^{\omega} & \text{if } a = \omega \\ \Gamma, c:[\vec{T}]^{(\bullet, i)} & \text{if } a = (\bullet, i+1) \end{cases}$$

Type splitting

$$\frac{}{[\overrightarrow{\mathbf{T}}]^{\omega} = [\overrightarrow{\mathbf{T}}]^{\omega} \circ [\overrightarrow{\mathbf{T}}]^{\omega}}\ \textsc{pUnr} \qquad \frac{}{\mathbf{proc} = \mathbf{proc} \circ \mathbf{proc}}\ \textsc{pProc} \qquad \frac{}{[\overrightarrow{\mathbf{T}}]^{(\bullet,i)} = [\overrightarrow{\mathbf{T}}]^{1} \circ [\overrightarrow{\mathbf{T}}]^{(\bullet,i+1)}}\ \textsc{pUnq}$$

Subtyping

$$\frac{}{(\bullet, i) \prec_s (\bullet, i+1)}\ \textsc{sIndx} \qquad \frac{}{(\bullet, i+1) \prec_s \omega}\ \textsc{sUnq} \qquad \frac{}{\omega \prec_s 1}\ \textsc{sAff} \qquad \frac{a_1 \prec_s a_2}{[\overrightarrow{\mathbf{T}}]^{a_1} \prec_s [\overrightarrow{\mathbf{T}}]^{a_2}}\ \textsc{sTyp}$$

Figure 3: Typing rules

Rule tCon contracts an assumption of the form $u:\mathbf{T}$, as long as $\mathbf{T}$ can be *split* as $\mathbf{T}_1$ and $\mathbf{T}_2$. Unrestricted assumptions can be split arbitrarily (pUnr and pProc); affine assumptions cannot be split at all. An assumption $u:[\vec{T}]^{\bullet}$ about a unique channel can be split into an affine assumption $u:[\vec{T}]^{1}$ and an assumption about a channel that is unique after one action $u:[\vec{T}]^{(\bullet,1)}$—an action using the latter assumption must be coupled with a co-action on the affine channel, and since the affine assumption can only be used once it is sound to assume that the channel is unique again after the action has happened. More generally, a channel which is unique after i actions can be split into an affine assumption and a channel which is unique after $(i+1)$ actions, and splitting is defined in such a way that the number of affine assumptions for a channel never exceeds the index i of the corresponding unique assumption.

In particular, when the index is 0, then no other assumptions about that channel can exist in the typing environment. This means that if a process can be typed using a unique assumption for a channel, no other process has access to that channel. The last structural rule, tRev, makes use of this fact to allow strong updates ("revision") to channels as long as they are unique. Uniqueness information is also used in the logical rule tFree, which typechecks deallocations by removing unique type assumptions; dually, the rule for allocations, tAll, introduces unique assumptions.

A few other logical rules deserve explanation. The rules for input and output (tOUT and tIN) decrement the attribute of the channel, *i.e.*, they *count* usage. This operation is denoted by $\Gamma, c:[\vec{T}]^{a-1}$ and is defined in Figure 3; in words, it states that affine assumptions can only be used once, unrestricted assumptions can be used an arbitrary number of types, and if a channel is unique after $i+1$ actions, then it will be unique after i actions once the action has been performed.

Moreover, in the case of output, tOUT requires separate typing assumptions for each of the channels that are sent. The attributes on these channels are not decremented, because no action has been performed on them; instead, the corresponding assumptions are handed over to some parallel process. Note that if the sending process wants to use any of these channels in its continuation (P), it will have to split the corresponding assumptions first.

Parallel composition, tPAR entails distributing type assumptions amongst processes. Rule tRST1 corresponds to the usual rule for scoping, and adds a typing assumption for allocated channels. However, in the case of deallocated channels, no assumption is added (tRST2).

The type system is sound:

Theorem 1 (Type safety) *If* $\Gamma \Vdash \sigma \triangleright P$ *then* $P \not\twoheadrightarrow^{err}$.

Theorem 2 (Subject reduction) *If* $\Gamma \Vdash \sigma \triangleright P$ *and* $\sigma \triangleright P \rightarrow \sigma' \triangleright P'$ *then there exists a environment* Γ' *such that* $\Gamma' \Vdash \sigma' \triangleright P'$.

The systems $\text{CLIENT}_i \parallel \text{TIMESRV} \parallel \text{DATESRV}$ for $i \in \{0,1,2\}$ can all be successfully typed in our type-system. We recall CLIENT_2 here for convenience and consider how it is typed, assuming x is not free in P:

$$\text{CLIENT}_2 \triangleq \mathsf{alloc}(x).\mathit{getTime}!\langle x\rangle.x?y.\mathit{getDate}!\langle x\rangle.x?z.\mathsf{free}x.P$$

Assuming an environment with $\mathit{getTime}:[[\mathbf{T}_{\text{time}}]^1]^\omega$ and $\mathit{getDate}:[[\mathbf{T}_{\text{date}}]^1]^\omega$, CLIENT_2 types as follows. Rule tALL assigns the unique type $[\mathbf{T}_{\text{time}}]^\bullet$ to variable x and the structural rule tCON then splits this unique assumption in two using pUNQ. Rule tOUT uses the affine assumption for x for the output argument and the unique-after-one assumption to type the continuation. Rule tIN restores the uniqueness of x for the continuation of the input after decrementing the uniqueness index, at which point tREV is applied to change the object type of x from $\mathbf{T}_{\text{time}}$ to $\mathbf{T}_{\text{date}}$. Once again, the pattern of applying tCON, tOUT and tIN repeats, at which point x is unique again and can be safely deallocated by tFREE.

Uniqueness allows us to typecheck a third client variation manifesting (explicit) ownership transfer. Rather than allocating a new channel, CLIENT_3 requests a channel from a heap of channels and returns the channel to the heap when it no longer needs it.

$$\text{CLIENT}_3 \triangleq \mathit{heap}?x.\mathit{getTime}!\langle x\rangle.x?y.\mathit{getDate}!\langle x\rangle.x?z.\mathit{heap}!x.P$$

With the assumption $\mathit{heap}:[[\mathbf{T}]^\bullet]^\omega$ typing the system below is analogous to the previous client typings.

$$\text{CLIENT}_3 \parallel \text{CLIENT}_3 \parallel \text{CLIENT}_3 \parallel \text{TIMESRV} \parallel \text{DATESRV} \parallel (\nu c)\mathit{heap}!\langle c\rangle \qquad (1)$$

In CLIENT_2 and CLIENT_3, substituting $x!().Q$ for P makes the clients unsafe (they perform *resp.* deallocated-channel usage and mismatching communication). Accordingly, both clients would be rejected the type-system because using x in the continuation $x!().Q$ requires a split for the assumption of x, and it is not possible to split any assumption into a unique assumption and any other assumption.

Finally, system (1) can be safely extended with processes such as CLIENT_4 which uses unique channels obtained from the heap in unrestricted fashion. Our type-system accepts CLIENT_4 by applying *subtyping* from unique to unrestricted on the channel x obtained from *heap*.

$$\text{CLIENT}_4 \triangleq \mathit{heap}?x.\mathsf{rec}X.(\mathit{getTime}!\langle x\rangle.x?y.P \parallel X)$$

4 Related work and conclusions

Resources and pi-calculus Resource usage in a pi-calculus extension is studied in [12] but it differs from our work in many respects. For a start, their scoping construct assumes an allocation semantics whereas we tease scoping and allocation apart as separate constructs. The resource reclamation construct in [12] is at a higher level of abstraction than $\mathsf{free}\,c.P$, and acts more like a "resource finalizer" leading to garbage collection. Resource reclamation is implicit in [12], permitting different garbage collection policies for the same program whereas, in the resource pi-calculus, resource reclamation is explicit and fixed for every program. The main difference however concerns the aim of the type system: our type system ensures safe channel deallocation and reuse; the type system in [12] statically determines an upper bound for the number of resources used by a process and does not use substructural typing.

Linearity versus Uniqueness In the absence of subtyping, affine typing and uniqueness typing coincide but, when subtyping is introduced, [7] shows they can be considered as dual. For linear typing the subtyping relation allows to coerce a non-linear assumption into a linear assumption, classically expressed as $!U \to U$, but for uniqueness typing the subtyping relation allows to coerce a unique assumption into a non-unique assumption. Correspondingly, the interpretation is different: linearity (*resp.* affinity) is a *local obligation* that a channel must be used exactly (*resp.* at most) once, while uniqueness is a *global guarantee* that no other processes have access to the channel. Combining both subtyping relations as we have done in this paper appears to be novel. The usefulness of the subtyping relation for affine or linear typing is well-known (e.g., see [9]); subtyping for unique assumptions allows to "forget" the uniqueness guarantee; CLIENT_4 above gave one example scenario in which this might be useful.

Linearity in functional programming Both uniqueness typing [4] and linear typing have been used in functional programming to support strong update, although even some proponents of linear typing agree that the match is not perfect [13, Section 3]. In this context "strong update" usually refers to updating arrays or modelling I/O [1], but Ahmed *et al.* have applied a linear type system to support "strong" (type changing) updates to ML-style references [2] (although in this type system, there is no subtyping relation). This system is therefore quite similar in intent to our type system but does not employ counting to allow for temporary violation of uniqueness; controlled uniqueness violation is essential in the case of message-passing concurrency because for a (unique or non-unique) channel to be useful, at least two processes need access to it.

Linearity in the pi-calculus Linear types for the pi-calculus were introduced by Kobayashi *et al.* [10] but do not employ any subtyping. Still, their system cannot be used as a basis for strong update or channel deallocation; although they allow to split a bidirectional linear ("unique") channel (into a linear input channel and a linear output channel; see their definition of the type combination operator $(+)$, Definition 2.3.1), these pieces are never "collected". The more refined type splitting operation we use in this paper, combined with the type decrement operation (which has no equivalent in their system) is key to make uniqueness useful for strong updates and deallocation. Our system can easily be extended to incorporate modalities but it does not rely on them; in our case, channel modalities are an orthogonal issue.

Fractional permissions and permission accounting Boyland [6] was one of the first to consider splitting permissions into *fractional* permissions which allow linearity or uniqueness to be temporarily violated. Thus, strong update is possible only with a full permission, whereas only passive access is permitted with a "fraction" of a permission. When all the fractions have been reassembled into one whole permission, strong update is once again possible.

Boyland's suggestion has been taken up by Bornat *et al.* [5], who introduce both fractional permissions and "counting" permissions to separation logic. Despite of the fact that their model of concur-

rency is shared-memory, their mechanism of permission splitting and counting is surprisingly similar to our treatment of unique assumptions. Apart from the application model, there are further discrepancies between the two reasoning systems. Whereas their resource-reading of semaphores targets *implicit* ownership-transfer, unique types allow us to reason about explicit ownership-transfer. Though more limited, explicit transfer permits reasoning to be syntax directed and thus more tractable. Moreover, subtyping from unique to unrestricted types provides the flexibility of not counting assumptions whenever this is not required, simplifying reasoning for resources that are not deallocated or strongly updated.

Session types Session types [8] or types with CCS-like usage annotations [9] can be used to describe channels which are used to send objects of different types. However, session types give detailed information on how channels are used, which makes modular typing difficult. For example, the *heap* channel used by CLIENT_3 cannot be given a type without knowing all the processes that use the heap.

Conclusions We have extended ideas from process calculi, substructural logics and permission counting to define a type system for a the pi-calculus extended with primitives for channel allocation and deallocation, where strong update and channel deallocation is deemed safe for unique channels.

References

[1] P. M. Achten & M. J. Plasmeijer (1995): *The ins and outs of Clean I/O*. Journal of Functional Programming 5(1), pp. 81–110.

[2] Amal Ahmed, Matthew Fluet & Greg Morrisett (2005): *A Step-Indexed Model of Substructural State*. In: *Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming (ICFP)*. ACM, pp. 78–91.

[3] Joe Armstrong (2007): *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf.

[4] Erik Barendsen & Sjaak Smetsers (1996): *Uniqueness Typing for Functional Languages with Graph Rewriting Semantics*. Mathematical Structures in Computer Science 6, pp. 579–612.

[5] Richard Bornat, Cristiano Calcagno, Peter O'Hearn & Matthew Parkinson (2005): *Permission accounting in separation logic*. SIGPLAN Not. 40(1), pp. 259–270.

[6] John Boyland (2003): *Checking Interference with Fractional Permissions*. In: R. Cousot, editor: *Static Analysis: 10th International Symposium*, Lecture Notes in Computer Science 2694. Springer, Berlin, Heidelberg, New York, pp. 55–72.

[7] Dana Harrington (2006): *Uniqueness logic*. Theoretical Computer Science 354(1), pp. 24–41.

[8] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *ESOP '98: Proceedings of the 7th European Symposium on Programming*. Springer-Verlag, London, UK, pp. 122–138.

[9] Naoki Kobayashi (2003): *Type Systems for Concurrent Programs*. In: *Formal Methods at the Crossroads: From Panacea to Foundational Support*, LNCS 2757. Springer Berlin / Heidelberg, pp. 439–453.

[10] Naoki Kobayashi, Benjamin C. Pierce & David N. Turner (1999): *Linearity and the pi-calculus*. ACM Trans. Program. Lang. Syst. 21(5), pp. 914–947.

[11] Robin Milner, Joachim Parrow & David Walker (1992): *A calculus of mobile processes, parts I and II*. Inf. Comput. 100(1), pp. 1–40.

[12] David Teller (2004): *Recollecting resources in the pi-calculus*. In: *Proceedings of IFIP TCS 2004*. Kluwer Academic Publishing, pp. 605–618.

[13] Philip Wadler (1991): *Is there a use for linear logic?* In: *Proceedings of the 2nd ACM SIGPLAN symposium on Partial Evaluation and semantics-based program manipulation (PEPM)*. ACM, pp. 255–273.

Resource-bound quantification for graph transformation

Paolo Torrini
University of Leicester
pt95@mcs.le.ac.uk

Reiko Heckel
University of Leicester
reiko@mcs.le.ac.uk

Graph transformation has been used to model concurrent systems in software engineering, as well as in biochemistry and life sciences. The application of a transformation rule can be characterised algebraically as a construction of a double-pushout (DPO) diagram in the category of graphs. We show how Intuionistic Linear Logic (ILL) can be extended with resource-bound quantification, allowing for an implicit handling of the DPO conditions, and how resource logic can be used to reason about graph transformation systems.

1 Introduction

Graphs can be used to model a variety of systems, not just in computer science, but throughout engineering and life sciences. When systems evolve, we are generally interested in the way they change, to predict, support, or react to evolution. Graph transformations combine the idea of graphs, as a universal modelling paradigm, with a rule-based approach to specify the evolution of systems, which can be regarded as a generalisation of term rewriting. There are several formalisations of graph transformation based on algebraic methods — the double-pushout approach (DPO) is one of the most influential [EEPT06] — and several analysis tools based (mainly) on rewriting systems. Logic-based representation and proof-theoretic methods can be worth investigating though, when we are interested in mechanising the verification of abstract properties, and in the formal development of model-based systems.

Intuitionistic linear logic (ILL) has been applied to the modelling of linear resources in programming languages as well as of concurrent systems [CS06, Abr93, Mil92], the latter through semantics based on Petri nets, transition systems, multiset rewriting and process calculi. In this paper we propose a new approach to the representation of graph transformation systems (GTS) based on DPO in a variant of quantified intuitionistic linear logic (QILL). The DPO approach is arguably the most mature of the mathematically-founded approaches to graph transformation, with a rich theory of concurrency comparable to (and inspired by) those of place-transition Petri nets and term rewriting systems.

ILL can provide a formally neat way to handle the creation/deletion of graph components associated with the application of transformation rules in GTS, once we have fixed a textual encoding for graphs and their transformations. Syntactical presentation of DPO-GTS in terms of graph expressions tend to focus on hypergraphs — a generalisation of graphs allowing for edges that connect more than two nodes [CMR94]. Hyperedges can be intuitively associated to predicates defined on nodes. However, a predicate calculus account of graph transformation can be misleading, insofar as the actual content of the theory is less a static relational structure, than a dynamic system of connected components — those that can be either created, deleted or preserved by each transformation. Graph components include the nodes as much as the edge components, each made of an edge with its attached nodes — something that points indeed toward a second-order account.

Submitted to:
LINEARITY 2009

Intuitively, each graph component can be regarded as linear resource. However, there is a clear distinction between components which are used linearly in a graph expression, i.e. the edge components, and those that represent connections between them, and therefore may have multiple occurrences in the expression, i.e. the nodes. This distinction is reflected in one of the conditions that characterise the definition of valid transformation rule in DPO — the dangling condition, which essentially prohibits deleting a node unless the attached edges are deleted, too. The difference between nodes and edges can be taken into account in at least two different ways, using linear logic. One possibility is to consider a node n as resource shared between distinct edge components, say $c_1(n)$ and $c_2(n)$ for example. The logic expression of this idea would be to represent the sharing as $n \multimap c_1(n)\&c_2(n)$. This approach, bearing similarities to the work in [DP08] based on separation logic, has the possible drawback of complicating the representation of parallel edge components.

An alternative approach, that we are pursuing here, is to express parallel composition in a straightforward way — i.e. in terms of tensor product, on one hand, and on the other hand, to distinguish between a node as graph component that can be transformed, i.e. as linear resource, and the node name that may occur any times in a graph expression — hence clearly a non-linear term. This distinction appears to rely on the possibility to express the referential relation between nodes and their names. When we reason about graphs up to isomorphism, assuming node names are hidden (i.e. restricted, in process calculus parlance) makes it necessary to express this relation modulo α-renaming. Moreover, it is important to make sure that hidden node names match quantitatively and qualitatively (by type) the node components. This correspondence is at the basis of the other constraint to the validity of DPO rules — the identification condition. We present an extension of linear logic that allows keeping track of these relations, in order to represent GTS. In the first-order formulation that we have given in [TH], we define a translation of GTS assuming a restriction to transformation rules that allow only for preservation of nodes. Here we introduce a higher-order version of the logic, needed in order to lift that restriction.

We express the referential relation by annotating the type of a node with the name that refers to the node (using the symbol $\downharpoonright$). We express name hiding by associating the introduction/elimination of hidden non-linear names with the consumption/deallocation of linear resources. This is essentially achieved by $\hat{\exists}$, operationally defined as a resource-bound existential quantifier. $\hat{\exists}$ has a separating character (though in a different sense from the intensional quantifiers in [Pym02]), by implicitly associating each bound variable to a linear resource in the sense of a tensor product. The representation of hiding, in connection to the availability of fresh names, is usually associated with the freshness quantifier of nominal logic [Pit01, CC04]. However, in our case linearity and the assumption that the environment does not contain duplicated labels suffice for the freshness of linear resources. We need to introduce a freshness condition on the instantiating non-linear term when we introduce the binding — which amounts, essentially, to making the introduction rule invertible, in contrast with standard existential quantification.

2 Hypergraphs and their transformations

Graph transformations can be defined on a variety of graph structures. In this paper we prefer typed hypergraphs, their n-ary hyperedges to be presented as predicates in the logic. A hypergraph (V, E, s) consists of a set V of vertices, a set E of hyperedges and a function

$\mathsf{s} : E \to V^*$ assigning each edge a sequence of vertices in V. A morphism of hypergraphs is a pair of functions $\phi_V : V_1 \to V_2$ and $\phi_E : E_1 \to E_2$ that preserve the assignments of nodes, that is, $\phi_V^* \circ \mathsf{s}_1 = \mathsf{s}_2 \circ \phi_E$. By fixing a type hypergraph $TG = (\mathcal{V}, \mathcal{E}, \mathsf{ar})$, we are establishing sets of node types $\mathcal{V}$ and edge types $\mathcal{E}$ as well as defining the arity $\mathsf{ar}(a)$ of each edge type $a \in \mathcal{E}$ as a sequence of node types. A TG-typed hypergraph is a pair $(HG, type)$ of a hypergraph HG and a morphism $type : HG \to TG$. A TG-typed hypergraph morphism $f : (HG_1, type_1) \to (HG_2, type_2)$ is a hypergraph morphism $f : HG_1 \to HG_2$ such that $type_2 \circ f = type_1$.

A *graph transformation rule* is a span of injective hypergraph morphisms $L \xleftarrow{l} K \xrightarrow{r} R$, called a *rule span*. A hypergraph transformation system (GTS) $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$ consists of a type hypergraph TG, a set P of rule names, a function mapping each rule name p to a rule span $\pi(p)$, and an initial TG-typed hypergraph G_0. A *direct transformation* $G \overset{p,m}{\Longrightarrow} H$ is given by a *double-pushout (DPO) diagram* as shown below, where (1), (2) are pushouts and top and bottom are rule spans. For a GTS $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$, a derivation $G_0 \Longrightarrow G_n$ in $\mathcal{G}$ is a sequence of direct transformations $G_0 \overset{r_1}{\Longrightarrow} G_1 \overset{r_2}{\Longrightarrow} \cdots \overset{r_n}{\Longrightarrow} G_n$ using the rules in $\mathcal{G}$. An hypergraph G is *reachable* in $\mathcal{G}$ iff there is a a derivation of G from G_0.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle d} & (2) & \downarrow{\scriptstyle m^*} \\ G & \xleftarrow[g]{} & D & \xrightarrow[h]{} & H \end{array}$$

Intuitively, the left-hand side L contains the structures that must be present for an application of the rule, the right-hand side R those that are present afterwards, and the gluing graph K (the *rule interface*) specifies the "gluing items", i.e., the objects which are read during application, but are not consumed. Operationally speaking, the transformation is performed in two steps. First, we delete all the elements in G that are in the image of $L \setminus l(K)$ leading to the left-hand side pushout (1) and the intermediate graph D. Then, a copy of $L \setminus l(K)$ is added to D, leading to the derived graph H via the pushout (2). The first step (deletion) is only defined if the a built-in application condition, the so-called gluing condition, is satisfied by the match m. This condition, which characterises the existence of pushout (1) above, is usually presented in two parts.

Identification condition: Elements of L that are meant to be deleted are not shared with any other elements, i.e., for all $x \in L \setminus l(K)$, $m(x) = m(y)$ implies $x = y$.

Dangling condition: Nodes that are to be deleted must not be connected to edges in G, unless they already occur in L, i.e., for all $v \in G_V$, if there exists $e \in G_E$ such that $\mathsf{s}(e) = v_1 \ldots v \ldots v_n$, then $e \in m_E(L_E)$.

The first condition guarantees two intuitively separate properties of the approach: First, nodes and edges that are deleted by the rule are treated as resources, i.e., m is injective on $L \setminus l(K)$. Second, there must not be conflicts between deletion and preservation, i.e., $m(L \setminus l(K))$ and $m(l(K)$ are disjoint. The second condition ensures that after the deletion of nodes, the remaining structure is still a graph and does not contain edges short of a node. It is particularly the first condition which makes linear logic so attractive for graph transformation. Crucially, it is also reflected in the notion of concurrency of the approach, where items that are deleted cannot be shared between concurrent transformations.

3 Linear λ-calculus for GTS

We extend ILL with typed quantification, building on a notion of linear λ-calculus that comes to us from an unpublished [Pfe02]. In contrast with [TH], here we do not restrict quantification to be first-order. Our primitive logic formulas are $\alpha = A \mid L(N_1,\ldots,N_n) \mid \mathbf{1} \mid \alpha_1 \otimes \alpha_2 \mid \alpha_1 \multimap \alpha_2 \mid !\alpha_1 \mid \top \mid \alpha_1 \& \alpha_2 \mid \forall x:\beta.\alpha \mid \hat{\exists} x:\beta.\alpha \mid \alpha \mid \alpha \downharpoonright N \mid \alpha = \alpha$, where we assume A to represent a primitive node type, and $L(N_1,\ldots,N_n)$ to represent the type of an edge component. We also define

$$\alpha \hat{\equiv} \beta \;=_{df}\; (\alpha \multimap \beta)\&(\beta \multimap \alpha) \qquad\qquad \alpha \# (x,N) \;=_{df}\; (\alpha[N/x])[x/N] = \alpha$$

Primitive expressions are $M = x \mid u \mid \mathsf{nil} \mid N_1 \otimes N_2 \mid \hat{\varepsilon}(N_1|N_2).N_3 \mid \lambda x.N \mid \hat{\lambda} u.N \mid N_1 \hat{} N_2 \mid N_1 N_2 \mid \langle N_1, N_2 \rangle \mid \langle\rangle \mid \mathsf{fst}\ N \mid \mathsf{snd}\ N$. We define $(\mathsf{let}\ P = N_1\ \mathsf{in}\ N_2) \;=_{df}\; (\lambda P.N_2)N_1$, where $P = x \mid u \mid \mathsf{nil} \mid N_1 \otimes N_2 \mid \hat{\varepsilon}(N_1|N_2).N_3 \mid \langle N_1, N_2 \rangle \mid \langle\rangle \mid \mathsf{fst}\ N \mid \mathsf{snd}\ N$ is a pattern.

We use two-entry sequents of form $\Gamma;\Delta \vdash N :: \alpha$, where Δ is a multiset of typed linear variables, denoted $u :: \alpha, v :: \alpha, \ldots$, Γ is a multiset of typed non-linear variables, denoted $x :: \alpha, y :: \alpha, \ldots$, $\vdash$ represents derivability, and $N :: \alpha$ is a typed expression. We assume that variables can occur at most once in a context. For multisets, we use sequence notation — modulo permutation and associativity, and $\cdot$ for the empty multiset. We consider a subset of linear variables as node names $(m,n,\ldots)$ and another subset as edge components $(c,d,\ldots)$. We consider a subset of non-linear variables as transformation rule names $(p,q,\ldots)$. When we "forget" about proof terms we are left with logic formulas and a consequence relation, which we denote by $\Vdash$. We use a notion of syntactic equality $=$ over types, stronger than linear equivalence $\hat{\equiv}$, to express a freshness constraint in $\hat{\exists} I$. We use $\hat{\lambda}$ to denote linear abstraction (and $\hat{}$ for linear application), in order to distinguish it from the non-linear abstraction (λ) — though this distinction is only meant to improve readability, since the difference between the two is determined by whether the abstraction variable is linear or not. The sequent calculus system is given by the axioms, the structural rules *Copy* (in the form of [Pfe94]), *Cut* and *Cut'*, and the operational rules rules. Cut elimination should be provable, along the lines of [Pfe94] — the only difference being the $\hat{\exists}$ rules — however it might not be straightforward.

3.1 Graphs in QILL

According the main lines of the formalisation we have presented in [TH], graphs can be generally associated to expressions in a sublanguage $G = n \mid c \mid \mathsf{nil} \mid G_1 \otimes G_2 \mid \hat{\varepsilon}(n|x).G$, and can be represented as derivations of form $\Gamma;\Delta \vdash G :: \gamma$, where Γ may contain all the typed non-linear variables that are not bound in the sequent, $\Delta = gc(G)$ contains all and only the typed linear variables that occur in G, and γ (the *graph formula*) is a formula in the $\mathbf{1},\otimes,\hat{\exists}$ fragment of the logic. The elements of Δ can represent the *ground constituents* — i.e. the nodes and the edge components of the graph. Untyped expressions (i.e. labels) represent component identity, whereas graph formulas contain the typing and connectivity information. The notion of graph up to isomorphism turns out to be captured by that of graph formula modulo linear equivalence, corresponding semantically to the set of graph formulas that can be derived from the ground constituent of the graph. Graphs can be represented schematically as $\hat{\exists}\overline{x:A}.L_1\,(\overline{x}_1) \otimes \ldots \otimes L_k\,(\overline{x}_k)$ where $\overline{x:A}$ is a sequence $x_1:A_1,\ldots,x_j:A_j$ of typed variables — this counts as normal form, and it is closed, i.e. $\overline{x}_1,\ldots,\overline{x}_k \subseteq \overline{x}$, whenever all nodes are hidden. As an example, a hypergraph with nodes $n_1:A_1,\ldots,n_4:A_4$ and edge components $c_1:L_1(n_1,n_2), c_2:L_2(n_1,n_3,n_4), c_3:L_3(n_2)$, can be represented with hidden nodes, in normal form, by a formula $\hat{\exists} x_1:A_1,\ldots,x_4:A_4.\ L_1(x_1,x_2) \otimes L_2(x_1,x_3,x_4) \otimes L_3(x_2)$.

3.2 Resource-bound existential quantifier

In order to type hiding, we need an existential-like quantifier (i.e. distributing over the tensor) that, in contrast with standard ones, ensures distinct bound variables in a formula cannot be instantiated with the same node — we will also say that the binder behaves injectively, i.e. that instantiations of multiple bound variables are always injective mappings. With $\hat{\exists}$, the instantiation of two distinct variables requires two linear resources, therefore cannot be derived from the instantiation of one — hence multiple instantiations behave injectively (Obs. 1(1)).

$$\frac{\Gamma;\Delta \vdash M :: \alpha[N_\Delta/x] \quad \Gamma;\cdot \vdash N_\Delta :: \beta \quad \Gamma;\Delta' \vdash n :: \beta \lfloor N_\Delta \quad \Gamma, x :: \beta;\cdot \vdash \mathsf{nil} :: \alpha\#(x, N_\Delta)}{\Gamma;\Delta,\Delta' \vdash \hat{\varepsilon}(n|N_\Delta).M :: \hat{\exists} x : \beta.\alpha}\ \hat{\exists}R$$

$$\frac{\Gamma, x :: \beta;\Delta, n :: \beta \lfloor x, v :: \alpha \vdash N :: \gamma}{\Gamma;\Delta, w :: \hat{\exists} x : \beta.\alpha \vdash \mathsf{let}\ \hat{\varepsilon}(n|x).v = w\ \mathsf{in}\ N :: \gamma}\ \hat{\exists}L$$

$\hat{\exists}$ left introduction is similar to the standard rule, except for the linear premise $n :: \beta \lfloor x$. $\hat{\exists}$ right introduction satisfies two non-standard constraints. The first one — $\Gamma;\Delta' \vdash n :: \beta \lfloor N_\Delta$ — means that N_Δ is the unique name (non-linear resource) for linear resource n. We need to force a dependence of the non-linear term on the linear context Δ — and here we do this by an index — lacking this, we might run into trouble with rule *Cut*. $\lfloor$ represents an injective mapping from linear resources to non-linear ones, ensuring that the node n cannot be associated to two different names, and hence used twice. The mapping is not generally invertible — since N is arbitrary, it can be associated to different linear resources, however this should not be a problem, as long as we do not allow for the same term to be used in different spatial contexts (that is why we need to force dependence on the linear context Δ). We do not give any proper introduction rule for $\lfloor$ (no associated constructor), and therefore $n :: \beta \lfloor N$ can only be introduced by axiom, under the assumption N is well-typed, i.e.

$$\frac{\Gamma;\cdot \vdash N :: \beta}{\Gamma; n :: \beta \lfloor N \vdash n :: \beta \lfloor N}\ \lfloor I$$

Given the definition of #, the second constraint is $\Gamma, x :: \beta;\cdot \vdash \mathsf{nil} :: (\alpha[N/x])[x/N] = \alpha$, which means that N does not occur in $\alpha[N/x]$ other than in place of x, i.e. we use type equality and substitution to formalise the requirement that N does not occur free in $\hat{\exists} x.\alpha$. We do not give rules for type equality here, but they are standard ones. With this constraint, the formula $\hat{\exists} x.\alpha$ is determined by the instance $\alpha[N/x]$ modulo renaming of bound variables — since *all* the occurrences of N must be replaced by x. Therefore applications of $\hat{\exists}R$ force a bijection between unbounded resources (node names) and variables that actually occur in α. Moreover, the rule consumes resources associated to $n :: \beta \lfloor N$, and therefore forces a linear dependence of variables on resources (nodes).

The hiding operator $\hat{\varepsilon}$ can be defined along the lines of the linear interpretation of the existential quantifier [CP02], closely associated to the standard intuitionistic one, i.e.

$$\hat{\varepsilon}(n|N).M :: \hat{\exists} x : \alpha.\beta \ =_{df}\ n \otimes !N \otimes M$$

with the proviso of the non-occurrence of either N or x in α. N depends only on the global context, hence it can be replaced with $!N$. The linearity of $n :: \alpha$ ensures injectivity.

It is not difficult to see that the following properties hold.

Prop. 1 (1) $\nVdash (\hat{\exists} x : \beta.\ \alpha(x,x)) \multimap \hat{\exists} xy : \beta.\ \alpha(x,y)$

the resource associated to x cannot suffice for x and y.

(2) $\nVdash \forall x : \beta.\ \beta \lfloor x \otimes \alpha(x,x) \multimap \hat{\exists} y : \beta.\alpha(y,x)$

y and x should be instantiated with the same term — but this is prevented by the freshness condition in $\hat{\exists}$ introduction

(3) $\nVdash (\hat{\exists} yx : \beta.\ \alpha_1(x) \otimes \alpha_2(x)) \multimap (\hat{\exists} x : \beta.\alpha_1(x)) \otimes \hat{\exists} x : \beta.\alpha_2(x)$

the two bound variables in the consequence require distinct resources and refer to distinct occurrences

Prop. 2 $\hat{\exists}$ satisfies properties of α-renaming, exchange and distribution over $\otimes$, i.e.

$$\Vdash (\hat{\exists} x : \alpha.\beta(x)) \triangleq (\hat{\exists} y : \alpha.\beta(y))$$
$$\Vdash (\hat{\exists} xy : \alpha.\gamma) \triangleq (\hat{\exists} yx.\gamma)$$
$$\Vdash (\hat{\exists} x : \alpha.\beta \otimes \gamma(x)) \triangleq (\beta \otimes \hat{\exists} x : \alpha.\gamma(x)) \qquad (x \text{ not in } \alpha)$$

In general $\hat{\exists}$ does not satisfy logical η-equivalence, i.e. it cannot be proved that α is equivalent to $\hat{\exists} x.\ \alpha$ when x does not occur free in α (neither sense of linear implication holds). This may come handy though, to represent graphs with isolated nodes.

We do not introduce term congruence explicitly, but we assume α-renaming, β- and η-congruence for λ and $\hat{\lambda}$ (with linearity check for the latter), as well as α-renaming, exchange, and distribution over $\otimes$ for $\hat{\varepsilon}$ (to match the type properties in Obs. 2).

3.3 Transformation rules

A DPO transformation rule (we consider rules with interfaces made only of nodes) can be represented as $\forall \overline{x : A}.\alpha \multimap \beta$ where α,β are graph expressions. Transformation rules can be encoded as typed non-linear variables $p :: \forall \overline{x : A}.\gamma_1 \multimap \gamma_2$ where γ_1,γ_2 are graph formulas. The implicit ! closure guarantees unrestricted applicability, universally quantified variables represent the rule interface, and linear implication represents transformation. The application of p to a closed graph formula $\alpha_G = \hat{\exists} \overline{y : A_y}.\beta_G$ determined by morphism m (as shown in the diagram) relies on the fact that the following rule is derivable

$$\frac{\begin{array}{l}\Gamma;\cdot \Vdash \alpha_G \triangleq \alpha_{G'} \quad \alpha_{G'} = \hat{\exists} \overline{z : A_z}.\alpha_L[\overline{z : A_z} \xleftarrow{d} \overline{x : A_x}] \otimes \alpha_C \\ \Gamma;\cdot \Vdash \alpha_H \triangleq \alpha_{H'} \quad \alpha_{H'} = \hat{\exists} \overline{z : A_z}.\alpha_R[\overline{z : A_z} \xleftarrow{d} \overline{x : A_x}] \otimes \alpha_C\end{array}}{\Gamma; \forall \overline{x : A_x}.\alpha_L \multimap \alpha_R \Vdash \alpha_G \multimap \alpha_H} \overset{p,m}{\Longrightarrow}$$

where the interface morphism d associated with m is represented by the multiple substitution $[\overline{z : A_z} \xleftarrow{d} \overline{x : A_x}]$, with $\overline{z : A_z} \subseteq \overline{y : A_y}$

Prop. 3 The QILL application schema satisfies the DPO conditions.

An informal proof can be built on top of Obs. 2(1,2,3). Injectivity rests on 1 for nodes and on linearity of the consequence relation for edge components, the identification condition rests on 2, the morphism characterisation of instantiations rests on 3, and this, together with the propositional structure of graph expressions, should suffice to ensure that also the dangling edge condition is satisfied.

A sequent $\cdot; G_0, P_1, \ldots, P_k \Vdash G_1$ can express that graph G_1 is reachable from the initial graph G_0 by applying rules $P_1 = \forall \overline{x}_1.\alpha_1 \multimap \beta_1$, …, $P_k = \forall \overline{x}_k.\alpha_k \multimap \beta_k$ once each, abstracting from the application order, each application resulting into a transformation step. A sequent $P_1, \ldots, P_k; G_0 \Vdash G_1$ can express that G_1 is reachable from G_0 by the same rules, regardless of whether or how many times they need to be applied. The parallel applicability of rules $\forall \overline{x}_1.\alpha_1 \multimap \beta_1$, $\forall \overline{x}_2.\alpha_2 \multimap \beta_2$ can be represented as applicability of $\forall \overline{x}_1, \overline{x}_2.\alpha_1 \otimes \alpha_2 \multimap \beta_1 \otimes \beta_2$ (true parallellism) or else of $\forall \overline{x}_1, \overline{x}_2.(\alpha_1 \multimap \beta_1) \otimes (\alpha_2 \multimap \beta_2)$.

3.4 Extending the logic

In order to deal with transformation rules that preserve edge components, we need higher-order universal quantification. This should ensure though, that while abstracted variables range over non-linear terms, in order to allow for multiple variables to be instantiated with the same term, instantiation takes place only when at least a linear resource of compatible type is available. Here we introduce $\restriction$ in order to extend non-linear contexts with premises that can be dropped when linear ones are available, and a notion of quantification that depends upon it, with the following rules

$$\frac{\Gamma, x :: \beta \restriction; \Delta \vdash N :: \alpha}{\Gamma; \Delta \vdash \lambda x.N :: \hat{\forall} x : \beta.\alpha}\ \hat{\forall} R \qquad \frac{\Gamma; \cdot \vdash M :: \beta \restriction \quad \Gamma; \Delta, v :: \alpha[M/x] \vdash N :: \gamma}{\Gamma; \Delta, u :: \hat{\forall} x : \beta.\alpha \vdash \mathsf{let}\ v = uM\ \mathsf{in}\ N :: \gamma}\ \hat{\forall} L$$

$$\frac{\Gamma; u :: \beta, \Delta \vdash N :: \alpha}{\Gamma, x :: \beta \restriction; u :: \beta, \Delta \vdash N :: \alpha}\ \restriction I$$

We need to assume that $\restriction$ premises cannot be introduced arbitrarily — in fact, restricting weakening. A general DPO transformation rule can be represented as $\forall \overline{x : A}.\hat{\forall} \overline{y : \gamma}.\alpha \multimap \beta$ where α, β, γ are graph expressions. The application of a rule requires that, after the instantiation of the interface nodes, interface edges can be instantiated, only if matching components are available as linear resources in the graph — though this does not involve consuming them.

3.5 Conclusion and further work

We have discussed how to represent DPO-GTS in a higher-order quantified extension of ILL. We have extended the encoding in [TH] by considering edge abstraction. We are interested in a logic that allows us to reason about concurrency and reachability at the abstract level, as well as for the synthesis of proof terms that can represent system runs — hence our interest in constructive logic. While we feel that the representation of contextual dependencies in the operational rules needs more investigation, especially with respect to cut elimination, we are also interested in extending the encoding to stochastic GTS.

References

[Abr93] S. Abramsky. Computational interpretation of linear logic. *Theoretical Computer Science* 111, 1993.

[CC04] L. Caires, L. Cardelli. A Spatial Logic for Concurrency II. *Theoretical Computer Science* 322(3):517–565, 2004.

[CMR94] A. Corradini, U. Montanari, F. Rossi. An abstract machine for concurrent modular systems: CHARM. *Theoretical Computer Science* 122:165–200, 1994.

[CP02] I. Cervesato, F. Pfenning. A linear logical framework. *Information and Computation* 179(1):19–75, 2002.

[CS06] I. Cervesato, A. Scedrov. Relating State-Based and Process-Based Concurrency through Linear Logic. *Electron. Notes Theor. Comput. Sci.* 165:145–176, 2006.

[DP08] M. Dodds, D. Plump. From hyperedge relpacement to separation logic and back. In *ICGT 2008 — Doctoral Symposium*. 2008.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of algebraic graph transformation*. Springer, 2006.

[Mil92] D. Miller. The pi-calculus as a theory in linear logic: preliminary resutls. In *Workshop on Extensions of Logic Programming*. LNCS 660, pp. 242–264. Springer, 1992.

[Pfe94] F. Pfenning. Structural Cut Elimination in Linear Logic. Technical report, Carnagie Mellon University, 1994.

[Pfe02] F. Pfenning. Linear Logic — 2002 Draft. Technical report, Carnagie Mellon University, 2002.

[Pit01] A. M. Pitts. Nominal Logic: A First Order Theory of Names and Binding. In *TACS '01: Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software*. Pp. 219–242. Springer-Verlag, 2001.

[Pym02] D. J. Pym. *The semantics and proof-theory of the logics of bunched implications*. Applied Logic Series. Kluwer, 2002.

[TH] P. Torrini, R. Heckel. Towards an embedding of graph transformation systems in intuitionistic linear logic. ICE'09 Workshop.
`www.cs.le.ac.uk/people/pt95/sll69.pdf`

Appendix — other proof rules

$$\frac{}{\Gamma;u :: \alpha \vdash u :: \alpha}\ Id \qquad \frac{}{\Gamma, x :: \alpha;\cdot \vdash x :: \alpha}\ UId \qquad \frac{\Gamma, x :: \alpha; u :: \alpha, \Delta \vdash N :: \gamma}{\Gamma, x :: \alpha; \Delta \vdash \mathsf{let}\ u = \mathsf{copy}(x)\ \mathsf{in}\ N :: \gamma}\ Copy$$

$$\frac{\Gamma;\Delta \vdash N :: \alpha \quad \Gamma; u :: \alpha, \Delta' \vdash M :: \beta}{\Gamma;\Delta, \Delta' \vdash \mathsf{let}\ N = u\ \mathsf{in}\ M :: \beta}\ Cut \qquad \frac{\Gamma;\cdot \vdash N :: \alpha \quad \Gamma, x :: \alpha;\Delta \vdash M :: \beta}{\Gamma;\Delta \vdash \mathsf{let}\ N = x\ \mathsf{in}\ M :: \beta}\ Cut'$$

$$\frac{\Gamma;\Delta_1 \vdash M :: \alpha \quad \Gamma;\Delta_2 \vdash N :: \beta}{\Gamma;\Delta_1, \Delta_2 \vdash M \otimes N :: \alpha \otimes \beta}\ \otimes R \qquad \frac{\Gamma;\Delta, u :: \alpha, v :: \beta \vdash N :: \gamma}{\Gamma;\Delta, w :: \alpha \otimes \beta \vdash \mathsf{let}\ u \otimes v = w\ \mathsf{in}\ N :: \gamma}\ \otimes L$$

$$\frac{\Gamma;\Delta, u :: \alpha \vdash M :: \beta}{\Gamma;\Delta \vdash \hat{\lambda} u : \alpha.\ M :: \alpha \multimap \beta}\ \multimap R \qquad \frac{\Gamma;\Delta_1 \vdash M :: \alpha \quad \Gamma;\Delta_2, u :: \beta \vdash N :: \gamma}{\Gamma;\Delta_1, \Delta_2, v :: \alpha \multimap \beta \vdash \mathsf{let}\ u = v\hat{}N\ \mathsf{in}\ N :: \gamma}\ \multimap L$$

$$\frac{}{\Gamma;\cdot \vdash \mathsf{nil} :: \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma;\Delta \vdash N :: \alpha}{\Gamma;\Delta, u :: \mathbf{1} \vdash \mathsf{let}\ \mathsf{nil} = u\ \mathsf{in}\ N :: \alpha}\ \mathbf{1}L$$

$$\frac{\Gamma;\Delta \vdash M :: \alpha \quad \Gamma;\Delta \vdash N :: \beta}{\Gamma;\Delta \vdash \langle M, N \rangle :: \alpha \& \beta}\ \&R \qquad \frac{}{\Gamma;\Delta \vdash \langle\rangle :: \top}\ \top R$$

$$\frac{\Gamma;\Delta, v :: \alpha \vdash N :: \gamma}{\Gamma;\Delta, u :: \alpha \& \beta \vdash \mathsf{let}\ v = \mathsf{fst}\ u\ \mathsf{in}\ N :: \gamma}\ \&L1 \qquad \frac{\Gamma;\Delta, v :: \beta \vdash N :: \gamma}{\Gamma;\Delta, u :: \alpha \& \beta \vdash \mathsf{let}\ v = \mathsf{fst}\ u\ \mathsf{in}\ N :: \gamma}\ \&L2$$

$$\frac{\Gamma;\cdot \vdash M :: \alpha}{\Gamma;\cdot \vdash !M :: !\alpha}\ !R \qquad \frac{\Gamma, p :: \alpha;\Delta \vdash N :: \beta}{\Gamma;\Delta, u ::!\alpha \vdash \mathsf{let}\ p = !u\ \mathsf{in}\ N :: \beta}\ !L$$

$$\frac{\Gamma, x :: \beta;\Delta \vdash M :: \alpha}{\Gamma;\Delta \vdash \lambda x.\ M :: \forall x : \beta.\ \alpha}\ \forall R \qquad \frac{\Gamma;\cdot \vdash M :: \beta \quad \Gamma;\Delta, v :: \alpha[M/x] \vdash N :: \gamma}{\Gamma;\Delta, u :: \forall x : \beta.\alpha \vdash \mathsf{let}\ v = uM\ \mathsf{in}\ N :: \gamma}\ \forall L$$

Labelled λ-calculi with explicit copy and erase

Maribel Fernández, Nikolaos Siafakas
King's College London, Dept. Computer Science, Strand, London WC2R 2LS, UK
[maribel.fernandez, nikolaos.siafakas]@kcl.ac.uk

We present two rewriting systems that define labelled explicit substitution λ-calculi. Our work is motivated by the close correspondence between Lévy's labelled λ-calculus and paths in proof nets, which played an important role in the understanding of the Geometry of Interaction. The structure of the labels in Lévy's labelled λ-calculus relates to the multiplicative information of paths; the novelty of our work is that we design labelled explicit substitution calculi that also keep track of exponential information present in call-by-value and call-by-name translations of the λ-calculus into linear logic proof-nets.

1 Introduction

Labelled λ-calculi have been used for a variety of applications, for instance, as a technology to keep track of residuals of redexes [6], and in the context of optimal reduction, using Lévy's labels [14]. In Lévy's work, labels give information about the history of redex creation, which allows the identification and classification of copied β-redexes. A different point of view is proposed in [4], where it is shown that a label is actually encoding a path in the syntax tree of a λ-term. This established a tight correspondence between the labelled λ-calculus and the Geometry of Interaction interpretation of cut elimination in linear logic [13].

Inspired by Lévy's labelled λ-calculus, we define labelled λ-calculi where the labels attached to terms capture reduction traces. However, in contrast with Lévy's work, our aim is to use the dynamics of substitution to include information in the labels about the use of resources, which corresponds to the exponentials in proof nets. In Lévy's calculus substitution is a meta-operation: substitutions are propagated exhaustively and in an uncontrolled way. We would like to exploit the fact that substitutions copy labelled terms and hence paths, but it is difficult to tell with a definition such as $(MN)^{\alpha}[P/x] = (M[P/x]N[P/x])^{\alpha}$, whether the labels in P are actually copied or not: P may substitute one or several occurrences of a variable, or it may simply get discarded.

In order to track substitutions we use calculi of explicit substitutions, where substitution is defined at the same level as β-reduction. Over the last years a whole range of explicit substitution calculi have been proposed, starting with the work of de Bruijn [9] and the $\lambda\sigma$-calculus [1]. Since we need to track copy and erasing of substitutions, we will use a calculus where not only substitutions are explicit, but also copy and erase operations are part of the syntax. Specifically, in this paper we use explicit substitution calculi that implement closed reduction strategies [10, 11]. This may be thought of as a more powerful form of combinatory reduction [8] in the sense that β-redexes may be contracted when the argument part or the function part of the redex is closed. This essentially allows more reductions to take place under abstractions. The different possibilities of placing restrictions on the β-rule give rise to different closed reduction strategies, corresponding to different translations of the λ-calculus into proof nets (a survey of available translations can be found in [16]). Closed reduction strategies date back to the late 1980's, in fact, such a strategy was used in the proof of soundness of the Geometry of Interaction [13].

Submitted to:
LINEARITY 2009

Labelled λ-calculi are a useful tool to understand the structure of paths in the Geometry of Interaction: Lévy's labels were used to devise optimisations in GoI abstract machines, defining new strategies of evaluation and techniques for the analysis of λ-calculus programs [5, 3]. The labels in our calculi of explicit substitutions contain, in addition to the multiplicative information contained in Lévy's labels, also information about the exponential part of paths in proof nets. In other words, our labels relate a static concept—a path—with a dynamic one: copying and erasing of substitutions. Thus, the labels can be used not only to identify caller-callee pairs, but also copy and erasing operations. This is demonstrated in this paper using two different labelled calculi. In the first system, the β-rule applies only if the function part of the redex is closed. We relate this labelled system with proof nets using the so-called call-by-value translation. We then define a second labelled λ-calculus where the β-rule applies only if the argument part of the redex is closed; thus, all the substitutions in this system are closed. We show that there is a tight relationship between labels in this system and paths in proof nets, using the so-called call-by-name translation.

The rest of the paper is organised as follows. In Section 2 we review the syntax of the calculus of explicit substitutions (λ_c-terms) and introduce basic terminology regarding linear logic proof nets. In Section 3 we introduce labelled versions of λ_c-terms. Section 4 presents the labelled calculus of closed functions (λ_{lcf}) that we relate to paths in proof nets coming from the call-by-value translation. Similarly, we relate in Section 5 the labelled version of the calculus of closed arguments λ_{lca} to closed cut-elimination of nets obtained from the call-by-name translation. We conclude in Section 6.

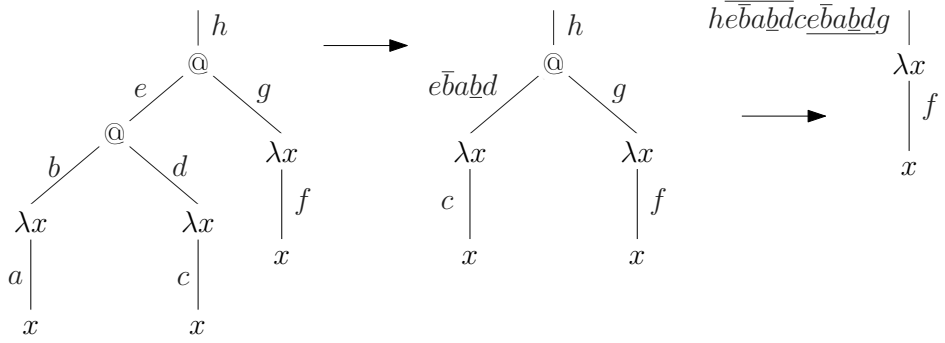
2 Background

We assume some basic knowledge of the λ-calculus [6], linear logic [12], and the geometry of interaction [13]. In this section we recall the main notions and notations that we will use in the rest of the paper.

Labels. There is a well known connection between labels and paths: the label associated to the normal form of a term in Lévy's labelled λ-calculus describes a path in the graph of the term [4]. The set of labels is generated by the grammar: $\alpha, \beta := a \mid \alpha\beta \mid \overline{\alpha} \mid \underline{\alpha}$, where a is an atomic label. Labelled terms are terms of the λ-calculus where each sub-term T has a label attached on it: T^{α}. Labelled β-reduction is given by $((\lambda x.M)^{\alpha}N)^{\beta} \to \beta\overline{\alpha} \bullet M[\underline{\alpha} \bullet N/x]$, where $\bullet$ concatenates labels: $\beta \bullet T^{\alpha} = T^{\beta\alpha}$. Substitution assumes the variable name convention [6]:

$$\begin{array}{lcllcl} x^{\alpha}[N/x] & = & \alpha \bullet N & (\lambda y.M)^{\alpha}[N/x] & = & (\lambda y.M[N/x])^{\alpha} \\ y^{\alpha}[N/x] & = & y^{\alpha} & (MN)^{\alpha}[P/x] & = & (M[P/x]N[P/x])^{\alpha} \end{array}$$

For example, III, where $I = \lambda x.x$, can be labelled, and then reduced as follows:



The final label generated describes a path in the tree representation of the initial term, if we reverse the

underlines. Following this path will lead to the sub-term which corresponds to the normal form, without performing β-reduction. This is just one perspective on Girard's geometry of interaction, initially set up to explain cut-elimination in linear logic. Here we are interested in the λ-calculus, but we can use the geometry of interaction through a translation into proof nets. These paths are precisely the ones that the GoI Machine follows [17]. In this example, the structure of the labels (overlining and underlining) tells us about the *multiplicative* information, and does not directly offer any information about the exponentials. To add explicitly the exponential information we would need to choose one of the known translations of the λ-calculus into proof nets, and it would be different in each case. Further, to maintain this information, we would need to monitor the progress of substitutions, so we need to define a notion of labelled λ-calculus for explicit substitutions. The main contribution of this paper is to show how this can be done.

Explicit substitutions and resource management. Explicit substitution calculi give first class citizenship to the otherwise meta-level substitution operation. Since we need to track copy and erasing of substitutions, in this paper we will use a calculus where not only substitutions are explicit, but also copy and erase operations are part of the syntax. The explicit substitution calculi defined in [10, 11] are well-adapted for this work: besides having explicit constructs for substitutions, terms include constructs for copying (δ) and erasing (ε) of substitutions. The motivation of such constructs can be traced back to linear logic, where the structural rules of weakening and contraction become first class logical rules, and to Abramsky's work [2] on proof expressions.

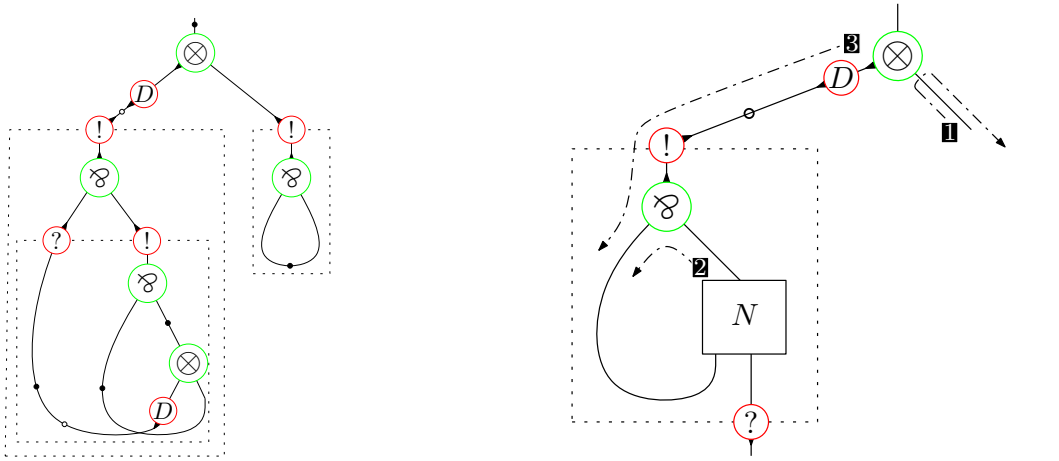
The table below defines the syntax of λ_c-terms, together with the variable constraints that ensure that variables occur linearly in a term. We use $\mathsf{fv}(\cdot)$ to denote the set of free variables of a term. We refer the reader to [11] for a compilation from λ-terms to λ_c-terms.

Term	Variable Constraint	Free variables
x	$\sim$	$\{x\}$
$\lambda x.M$	$x \in \mathsf{fv}(M)$	$\mathsf{fv}(M) - \{x\}$
MN	$\mathsf{fv}(M) \cap \mathsf{fv}(N) = \emptyset$	$\mathsf{fv}(M) \cup \mathsf{fv}(N)$
$\varepsilon_x.M$	$x \notin \mathsf{fv}(M)$	$\mathsf{fv}(M) \cup \{x\}$
$\delta_x^{y,z}.M$	$x \notin \mathsf{fv}(M), y \neq z, \{y,z\} \subseteq \mathsf{fv}(M)$	$(\mathsf{fv}(M) - \{y,z\}) \cup \{x\}$
$M[N/x]$	$x \in \mathsf{fv}(M), (\mathsf{fv}(M) - \{x\}) \cap \mathsf{fv}(N) = \emptyset$	$(\mathsf{fv}(M) - \{x\}) \cup \mathsf{fv}(N)$

For example, the compilation of $\lambda x.\lambda y.x$ is $\lambda x.\lambda y.\varepsilon_y.x$, and the compilation of $(\lambda x.xx)(\lambda x.xz)$ is $(\lambda x.\delta_x^{x',x''}.x'x'')(\lambda x.xz)$. We remark that both yield terms that satisfy the variable constraints.

Using λ_c-terms, two explicit substitution calculi were defined in [11]: λ_{cf}, the calculus of closed functions, and λ_{ca}, the calculus of closed arguments. In the former, the β rule requires the function part of the redex to be closed, whereas in the latter, the argument part must be closed to trigger a β-reduction. In the rest of the paper we define labelled versions of these calculi and relate labels to proof net paths.

Proof nets and the Geometry of Interaction. The canonical syntax of a linear logic proof is a graphical one: a proof net. Nets are built using the following set of nodes: *Axiom (•)* and *cut (∘)* nodes (such nodes are dummy-nodes in our work and simply represent "linking" information); Multiplicative nodes: $\otimes$ and $\parr$; Exponential nodes: *contraction (fan)*, *of-course (!)*, *why-not (?)*, *dereliction (D)* and *weakening (W)*. A sub-net may be enclosed in a *box* built with one *of-course*-node and $n \geq 0$ *why-not*-nodes, which we call auxiliary doors of a box. The original presentation of a net is oriented so that edges designated as conclusions of a node point downwardly: we abuse the natural orientation and indicate with an arrow-head the conclusion of the node (see Figure 1); all other edges are premises. Such structures may be obtained by one of the standard translations, call-by-name or call-by-value [16, 12], of λ-terms. For instance, the net in Figure 1a is obtained by the call-by-value translation of the λ-term $(\lambda xy.xy)(\lambda x.x)$.



(a) A proof net obtained by the call-by-value translation of the term $(\lambda xy.xy)(\lambda x.x)$

(b) A bouncing (1), a twisting (2), and a straight path (3)

Figure 1: Examples of proof-nets and paths

We assume a weak form of cut elimination—closed cut elimination— denoted by $\Rightarrow$. This is ordinary multiplicative and exponential proof-net reduction with the restriction that every exponential step can handle only boxes with no auxiliary doors. We refer the reader to [13, 16] for a specification of this cut-elimination strategy, which is also used in the context of Interactions Nets [15].

In this work we obtain labelled (weighted) versions of nets via inductive translations of λ_c-terms, which we define later on.

Weighted nets. To each edge of a net, we associate a *weight* (w) built from terms of the dynamic algebra $L^\star$: constants p,q (multiplicative), r,s,t,d (exponential), 0 and 1; an associative composition operator "." with unit 1 and absorbing element 0; an involution $(\cdot)^*$ and a unary morphism $!(\cdot)$. We use meta-variables $\alpha,\beta,\ldots$ for terms and we shall write $!^n(\cdot)$ for $n \geq 0$ applications of the morphism. Intuitively, weights are used to identify paths, and the algebra is used to pick out the paths that survive reduction. We omit the definition of the correct labelling (with weights) of a net: this will again be obtained by translations of λ_c-terms into labelled proof nets.

We use metavariables $\phi,\chi\ldots$ to range over paths. Paths are assumed to be **a)** *non-twisting*, that is, paths are not over different premises of the same node and **b)** *non-bouncing*, that is, paths do not bounce off nodes. We call such paths *straight*; these traverse a weighted edge e forwardly when moving towards the premise of the incident node (resp. backwardly e^r when moving towards the conclusion) such that direction changes happen only at cut and axiom links. For instance, Figure 1b shows a bouncing, a twisting, and a straight path respectively. The weight of a path is 1 if it traverses no edge — this weight is the identity for composition which we usually omit; if $\phi = e \cdot \psi$ is a path then its weight $w(\phi)$ is defined to be[1] $w(e) \cdot w(\psi)$ and we have $w(e^r) = w(e)^*$. We are mainly interested in the statics of the algebra and omit the equations that terms satisfy. We refer the reader to [7, 3, 5] for a more detailed treatment.

3 Labelled terms

We attach labels to λ_c-terms in order to capture information not only about β-reductions but also about propagation of substitutions. We adopt the same language for labels as in [20], where a confluent system

[1] Traditionally, weights are composed antimorphically but in this work we read paths and compose weights from left to right.

$$
\begin{array}{lcllcl}
(a)^r & = & a & \beta \bullet x^{\alpha} & = & x^{\beta\alpha} \\
(\alpha\beta)^r & = & (\beta)^r \cdot (\alpha)^r & \beta \bullet (\lambda x.M)^{\alpha} & = & (\lambda x.M)^{\beta\alpha} \\
(\overline{\alpha})^r & = & \overline{(\alpha)^r} & \beta \bullet (MN)^{\alpha} & = & (MN)^{\beta\alpha} \\
(\underline{\alpha})^r & = & \underline{(\alpha)^r} & \alpha \bullet (\delta_x^{y,z}.M) & = & (\delta_x^{y,z}.\alpha \bullet M) \\
(\overrightarrow{E})^r & = & \overleftarrow{E} & \alpha \bullet (\varepsilon_x.M) & = & (\varepsilon_x.\alpha \bullet M) \\
(\overleftarrow{E})^r & = & \overrightarrow{E} & \alpha \bullet (M[N/x]) & = & (\alpha \bullet M)[N/x]
\end{array}
$$

Table 1: Operation of $(\cdot)^r$ and $\cdot \bullet \cdot$

(λ_{lcf}) is defined and informally related to traces in a call-by-value proof-net translation. To establish the required correspondence, in this paper we translate (in Sections 4 and 5) our labels into terms of $L^{\star}$ (defined in the previous section), which is the de-facto language for labels in proof nets.

Definition 1. Labels are defined by the following grammar; a is an atomic label taken from a denumerable set $\{a,b,\ldots\}$, and all labels in C are atomic.

$$\alpha,\beta := a \mid \alpha\cdot\beta \mid \overline{\alpha} \mid \underline{\alpha} \mid C; \qquad C := \overrightarrow{E} \mid \overleftarrow{E}; \qquad E := D \mid\, !\, \mid ? \mid R \mid S \mid W;$$

These labels are similar to Lévy's labels except that we have (atomic) markers to describe exponentials, motivated by the constants in the algebra $L^{\star}$. Atomic labels from $\{a,b,\ldots\}$ correspond to 1 while W corresponds to 0; the marker ? corresponds to t and $\{d,r,s\}$ are easily recognised in our labels. Multiplicative constants $\{p,q\}$ are treated implicitly via over-lining and underlining. One may recover the multiplicative information, which is simply a bracketing, using the function f which is the identity transformation on all labels except for $f\ \overline{\alpha} = \overrightarrow{Q}\cdot(f\ \alpha)\ \cdot\overleftarrow{Q}$ and $f\ \underline{\alpha} = \overrightarrow{P}\cdot(f\ \alpha)\ \cdot\overleftarrow{P}$. The marker ! deserves more attention since the straightforward analogue in the algebra is the morphism $!(\cdot)$. The purpose of the marker is to delimit regions in the label, that is, paths that traverse edges entirely contained in a box.

Initialisation: To each term in λ_c, except for δ, ε and substitution-terms, we associate a unique and pairwise distinct label from $\{a,b,\cdots\}$. Notice that we do not place any exponential markers on initialised terms: it is the action of the substitution that yields their correct placement. From now on, we assume that all λ_c-terms come from the compilation of a λ-term and receive an initial labelling. Free and bound variables of labelled λ_c-terms are defined in the usual way.

4 The labelled calculus of closed functions

In this section we define the labelled calculus λ_{lcf}, that yields traces for the call-by-value translation of λ_c-terms into linear logic proof nets.

Definition 2 (Labelled Reduction in λ_{lcf})**.** The *Beta* rule of the labelled calculus λ_{lcf} is defined by

$$((\lambda x.M)^{\alpha}N)^{\beta} \to_{lcf} \beta\,\overrightarrow{\mathsf{D}\,\alpha\,\overleftarrow{!}}\bullet M[\underline{(\overrightarrow{\mathsf{D}\,\alpha\,\overleftarrow{!}})^r}\bullet N/x] \quad \textit{if}\ \mathsf{fv}((\lambda x.M)^{\alpha}) = \emptyset$$

The operator $\bullet$ and the function $(\cdot)^r$ on labels are defined in Table 1. We place substitution-rules (σ) at the same level as the *Beta* rule. These are given in Table 2. We write $\to^*_{lcf}$ for the transitive reflexive closure of $\to_{lcf}$ and we may omit the name of the relation when it is clear from the context. Reduction is allowed to take place under any context satisfying the conditions.

Rule	Reduction			Condition
Lam	$(\lambda y.M)^{\alpha}[N/x]$	$\rightarrow_{lcf}$	$(\lambda y.M[\overrightarrow{?} \bullet N/x])^{\alpha}$	$\mathsf{fv}(N) = \emptyset$
App1	$(MN)^{\alpha}[P/x]$	$\rightarrow_{lcf}$	$(M[P/x]N)^{\alpha}$	$x \in \mathsf{fv}(M)$
App2	$(MN)^{\alpha}[P/x]$	$\rightarrow_{lcf}$	$(MN[P/x])^{\alpha}$	$x \in \mathsf{fv}(N)$
Cpy1	$(\delta_x^{y,z}.M)[N/x]$	$\rightarrow_{lcf}$	$M[\overrightarrow{\mathsf{R}} \bullet N/y][\overrightarrow{\mathsf{S}} \bullet N/z]$	$\mathsf{fv}(N) = \emptyset$
Cpy2	$(\delta_x^{y,z}.M)[N/x']$	$\rightarrow_{lcf}$	$(\delta_x^{y,z}.M[N/x'])$	$\sim$
Ers1	$(\varepsilon_x.M)[N/x]$	$\rightarrow_{lcf}$	$M, \quad \{\overrightarrow{\mathsf{W}} \bullet N\} \cup B$	$\mathsf{fv}(N) = \emptyset$
Ers2	$(\varepsilon_x.M)[N/x']$	$\rightarrow_{lcf}$	$(\varepsilon_x.M[N/x'])$	$\sim$
Var	$x^{\alpha}[N/x]$	$\rightarrow_{lcf}$	$\alpha \bullet N$	$\sim$
Cmp	$M[P/y][N/x]$	$\rightarrow_{lcf}$	$M[P[N/x]/y]$	$x \in \mathsf{fv}(P)$

Table 2: Labelled substitution (σ) rules in λ_{lcf}

The calculus defined above is a labelled version of λ_{cf}, the calculus of closed functions [11]. The conditions on the rules may not allow a substitution to fully propagate (i.e., a normal form may contain substitutions) but the calculus is adequate for evaluation to weak head normal form [11]. Although reduction is weak, the system does not restrict reduction under abstraction altogether as do theories of the weak λ-calculus.

Intuitively, the purpose of the *Beta* rule is to capture two paths, one leading into the body and one to the argument of the proof net representation of the function application. The controlled copying and erasing (*Cpy1* and *Ers1*) of substitutions allows the identification of paths that start from contraction nodes and weakening nodes respectively. Rule *Ers1* has a side effect; erased paths are kept in a set B, which is initially empty. Formally, this rewriting system is working on pairs (M,B) of a λ-term and a set of labels. The set B deserves more regards: there exist paths in the GoI that do not survive the action of reduction (they are killed off by the dynamics of the algebra), however, one could impose a strategy where such sub-paths may be traversed indeed. For instance, evaluating arguments before function application gives rise to traversals of paths that lead to terms that belong in B. In this sense, it is not only the arguments that get discarded but also their labels, i.e. the paths starting from variables that lead to unneeded arguments. We omit explicit labels on copying (δ) and erasing (ε) constructs: these are used just to guide the substitutions. The composition rule *Cmp* is vital in the calculus because we may create open substitutions in the *Beta* rule.

The calculus is α-conversion free: the propagation of open substitutions through abstractions is the source of variable capture, which is here avoided due to the conditions imposed on the *Lam* rule. Moreover, the labelled calculus has the following useful properties:

Property. *1.* Strong normalisation of substitution rules. *The reduction relation generated by the σ-rules is terminating.*

2. Propagation of substitutions. *Let $T = M[N/x]$. If $\mathsf{fv}(N) = \emptyset$ then T is not a normal form. As a corollary closed substitutions can be fully propagated.*

3. Confluence. *λ_{lcf} reductions are confluent: if $M \rightarrow^*_{lcf} N_1$ and $M \rightarrow^*_{lcf} N_2$ then there exists a term P such that $N_1 \rightarrow^*_{lcf} P$ and $N_2 \rightarrow^*_{lcf} P$.*

The proof of termination of σ is based on the observation that rules push the substitution down the term (which can be formalised using the standard interpretation method). Since the propagation rules are defined for closed substitutions, it is easy to see that closed substitutions do not block. The proof of confluence is more delicate. We derive confluence in three steps: First, we show confluence of the σ rules

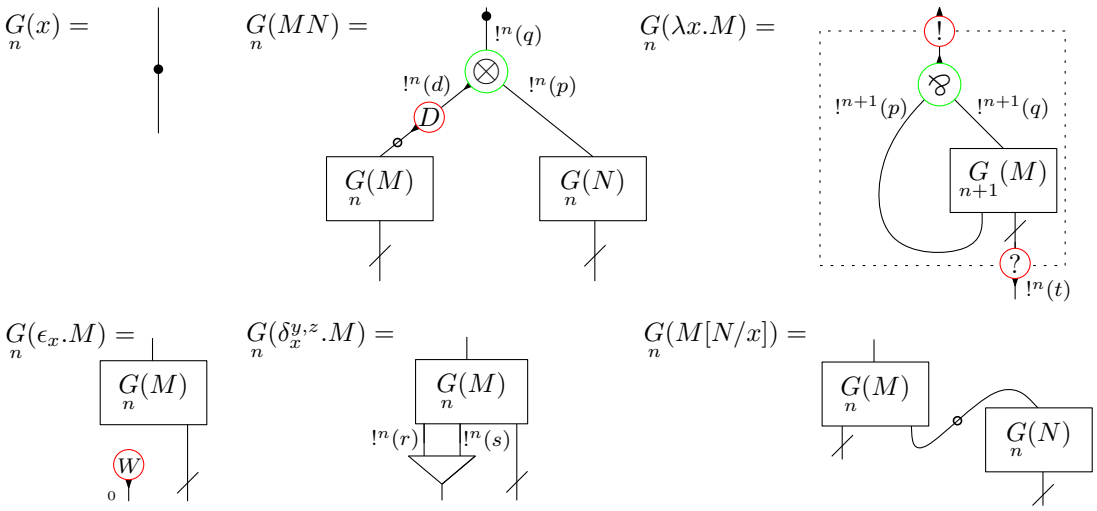


Figure 2: Translation of λ_c-terms into weighted call-by-value proof nets

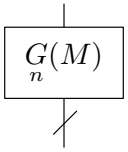
(local confluence suffices, by Newman's lemma [18], since the rules are terminating). Then we show that β alone is confluent, and finally we use Rosen's lemma [19], showing the commutation of the β and σ reduction relations. For detailed proofs we refer the reader to [21].

Labels in λ_{lcf} and paths in the call-by-value translation

There is a correspondence between our labels and the paths in weighted proof nets. The aim of the remainder of the section is to justify the way in which this calculus records paths in proof nets. This will provide us with a closer look at the operational behaviour of the calculus, especially at the level of propagation of substitutions, and will highlight the relationship, but also the differences, between term reduction and proof net reduction.

We first define a call-by-value translation from labelled λ_c-terms to proof nets labelled with weights taken from algebra L^*, and then we show that the set of labels generated at each rewrite step in the calculus coincides with the set of weights in the corresponding nets. For simplicity, we first consider the translation of unlabelled terms; then we extend the translation to labelled terms.

Definition 3. In Figure 2 we give the translation function $G_n(\cdot)$ from unlabelled λ_c-terms to call-by-value proof nets. We use a parameter n ($n \geq 0$) to record a current-box level, which indicates the box-nesting in which the translation works. The translation of a term M is obtained with $G_0(M)$, indicating the absence of box-nesting in initial terms. In our translation we omit the weights 1 of cuts and axioms. In general, the translation of a term M at level n is a proof net:



where the crossed wire at the bottom represents a set of edges corresponding to the free variables in M.

At the syntactical level, the correspondence of a δ-term to a duplicator (fan) node is evident as well as the correspondence of ε-term to weakening (W) nodes. In the former case, we assume that the variable y (resp. z) corresponds to the link labelled with r (resp. s). The free variable x corresponds to the wire at the conclusion of the fan. The case for the erasing term is similar, the free variable x corresponds to

Atomic labels			Composite labels		
$lw[\![a]\!]^n$	$=$	$(1,n)$			
$lw[\![\overrightarrow{\mathsf{R}}]\!]^n$	$=$	$(!^n(r),n)$	$lw[\![\overline{\alpha}]\!]^n$	$=$	$((!^n(q))\cdot w\cdot !^{n'}(q^*),n')$
$lw[\![\overleftarrow{\mathsf{R}}]\!]^n$	$=$	$(!^n(r^*),n)$		where	$(w,n') = lw[\![\alpha]\!]^n$
$lw[\![\overrightarrow{\mathsf{S}}]\!]^n$	$=$	$(!^n(s),n)$			
$lw[\![\overleftarrow{\mathsf{S}}]\!]^n$	$=$	$(!^n(s^*),n)$	$lw[\![\underline{\alpha}]\!]^n$	$=$	$((!^n(p))\cdot w\cdot !^{n'}(p^*),n')$
$lw[\![\overrightarrow{\mathsf{D}}]\!]^n$	$=$	$(!^n(d),n)$		where	$(w,n') = lw[\![\alpha]\!]^n$
$lw[\![\overleftarrow{\mathsf{D}}]\!]^n$	$=$	$(!^n(d^*),n)$			
$lw[\![\overrightarrow{?}]\!]^n$	$=$	$(!^{n-1}(t^*),n-1)$	$lw[\![\alpha\beta]\!]^n$	$=$	$(w\cdot w',n'')$
$lw[\![\overleftarrow{?}]\!]^n$	$=$	$(!^n(t),n+1)$		where	$(w,n') = lw[\![\alpha]\!]^n$
$lw[\![\overrightarrow{!}]\!]^n$	$=$	$(1,n-1)$			$(w',n'') = lw[\![\beta]\!]^{n'}$
$lw[\![\overleftarrow{!}]\!]^n$	$=$	$(1,n+1)$			

Table 3: Translation of labels into weights

the conclusion of the weakening node. The encoding of a substitution term is the most interesting: a substitution redex corresponds to a cut in proof nets.

We next give the translation of labelled terms, which is similar except that now we must translate labels of the calculus into terms of the algebra. This is a two-step process: first we must consider how labels correspond to weights in proof nets, and then we must place the weight on an edge of the graph.

Definition 4. The weight of a label is obtained by the function $lw :: label \to level \to (weight, level)$ where $level \in \mathbb{N}$ is a level (or box depth) number. Given a label and the level number of the first label, the function defined in Table 3 yields a weight together with the level number of the last label. We assume that the input level number always is an appropriate one.

Before we give the translation, we introduce a convention that will help us to reason about input and output levels: instead of projecting the weight from the tuple in the previous definition, we place the tuple itself on a wire like this:

$$\text{——}\ (a,o)\ \text{——}\ !^o(\beta)\ \text{——}$$

where α is the weight of the translated label, o is a level number. Now, after projecting the weight from the tuple one may compose with existing weights on the wire ($!^o(\beta)$). This simply introduces a delay in our construction that helps us to maintain the levels.

The translation of a labelled term is obtained using the function $G_i(M)$, which is the same as in Definition 3, with the difference that now when we call $G_i(M)$, the parameter i depends on the translation of the label. Specifically, for each term that has a label on its root, we first translate the label using Definition 4, and place the obtained output onto the root of G. In Figure 3 we show the translation of labelled application and substitution terms and the remaining cases can be easily reconstructed from the translation of the unlabelled terms. Thus, the only difference is that when a term has a label on the root, the translation must use the output level to propagate to subterms.

To extract the external label of a term we use the function:

$$\begin{array}{lcllcl} \text{label } x^\alpha & = & \alpha & \text{label } (MN)^\alpha & = & \alpha \\ \text{label } (\lambda x.M)^a & = & \alpha & \text{label } (\delta_x^{y,z}.M) & = & \text{label } M \\ \text{label } (\varepsilon_x.M) & = & \text{label } M & \text{label } (M[N/x]) & = & \text{label } M \end{array}$$

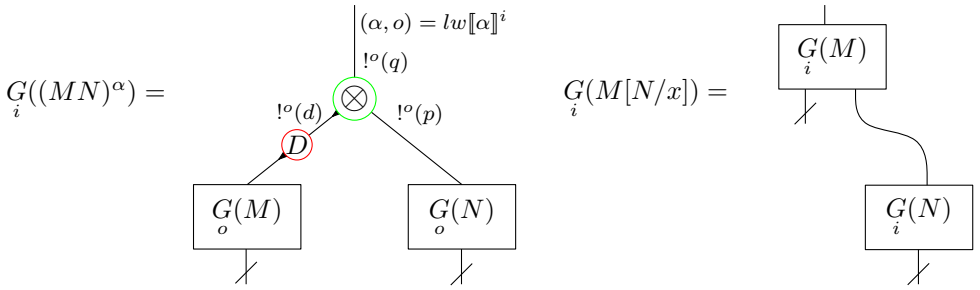


Figure 3: Translation of labelled application and substitution terms

Similarly, we can get the label of a free variable in a term. This is just a search and we omit the definition. Thanks to the linearity of terms, there is exactly one wire for each free variable in the term.

Main result. Before proving the main result of this section (Theorem 1), which states the correspondence between labels and paths, we need a few general properties.

Proposition 1 (First and last atomic labels)**.** *Let k be the external label of an initialised term T, and assume $T \to^* T'$.*

1. *If $(label\ T') = l_1 \dots l_n$, $n \geq 1$, then l_1 is k.*
2. *Let N be an application, abstraction or variable subterm of T' with $(label\ N) = l_1 \dots l_n$, $n \geq 1$. The atomic label l_n identifies an application (resp. abstraction, resp. variable) term in T iff N is an application (resp. abstraction, resp. variable) term.*

This property captures the idea that we cannot lose the original root of a reduction and terms never forget about their initial label. Notice that we consider terms that do receive a label by the initialisation. Thus the last atomic label of a string on a term-construct is the label the construct has obtained by initialisation. This is because labels get prefixed by the actions of the calculus. Additionally, notice that in $((\lambda x.M)^\alpha N)^\beta \to \beta\overline{\alpha} \bullet M[\underline{\alpha}^r N/z]$, where we forget about the markers, we know that the last label of β must be the one of the application node in which M was the functional part. But we also know the first label of α: if it is atomic, then it is the label of this λ in the initial term. Otherwise it is the label of the functional edge of the application node identified by the last label of β. One argues similarly for the argument.

Lemma 1. *If $T = M[N/x]$ then there is a decomposition $(label\ N) = \omega\underline{\alpha}\sigma$ such that ω is a prefix built with exponential markers having the shape $\overrightarrow{E}_1 \dots \overrightarrow{E}_n$ with $n \geq 0$.*

Proof. A simple inspection of the rewrite rules shows that we always prefix the external label of N with some exponential marker as long as the substitution propagates with label sensitive rules. The moment where ω itself gets prefixed is during a variable substitution which stops the process. □

The previous statement allows us to point out the distinguishing pattern of labels on substitution terms where we shall see that label sensitive propagation of substitutions corresponds to building an exponential path in a proof-net.

Corollary 1. *The atomic label that stands on an initialised variable can be followed only by $\omega\underline{\alpha}\sigma$.*

Proof. This is a consequence of the last-label property stated above and the action of the rewrite rule *Var*. □

Next we show that the labels of the calculus adequately trace paths in proof-nets. In particular, we show that the set of labels generated in the calculus coincides with weights of straight paths in proof-nets in the following sense:

Theorem 1. *Let $W_G = \{w(\phi) \mid \phi \in \text{straight paths of } G\}$ denote the set of weights of straight paths observable in a graph G and let T be a labelled term. If $T \to T'$ then $W_{G(T)} = W_{G(T')}$.*

Proof. The proof is by induction on T and is given in the appendix. □

5 The labelled calculus of closed arguments

In this section we define a labelled calculus, called λ_{lca}, that yields traces for the call-by-name translation of λ_c-terms into linear logic proof nets.

Definition 5 (Labelled reduction in λ_{lca})**.** The new *Beta* rule is defined by:

$$((\lambda x.M)^{\alpha}N)^{\beta} \to_{lca} \beta \bullet \overline{\alpha} M[\underline{(\alpha)^{r}\overleftarrow{!}} \bullet N/x] \quad \text{if } \mathsf{fv}(N) = \emptyset$$

where the operator $\bullet$ is given in Definition 2. The substitution rules for this system are presented below:

Rule	Reduction			Condition
Lam	$(\lambda y.M)^{\alpha}[N/x]$	$\to_{\lambda_{lca}}$	$(\lambda y.M[N/x])^{\alpha}$	$\sim$
App1	$(MN)^{\alpha}[P/x]$	$\to_{lca}$	$(M[P/x]N)^{\alpha}$	$x \in \mathsf{fv}(M)$
App2	$(MN)^{\alpha}[P/x]$	$\to_{lca}$	$(MN[\overrightarrow{?} \bullet P/x])^{\alpha}$	$x \in \mathsf{fv}(N)$
Cpy1	$(\delta_x^{y,z}.M)[N/x]$	$\to_{lca}$	$M[\overrightarrow{\mathsf{R}} \bullet N/x][\overrightarrow{\mathsf{S}} \bullet N/x]$	$\sim$
Cpy2	$(\delta_x^{y,z}.M)[N/x']$	$\to_{lca}$	$(\delta_x^{y,z}.M[N/x'])$	$\sim$
Ers1	$(\varepsilon_x.M)[N/x]$	$\to_{lca}$	$M, \quad \{\overrightarrow{\mathsf{W}} \bullet N\} \cup B$	$\sim$
Ers2	$(\varepsilon_x.M)[N/x']$	$\to_{lca}$	$(\varepsilon_x.M[N/x'])$	$\sim$
Var	$x^{\alpha}[N/x]$	$\to_{lca}$	$\alpha \overrightarrow{\mathsf{D}} \bullet N$	$\sim$

This is the labelled version of the calculus of closed arguments in [11] and we refer the reader to [21] for a proof of confluence for the labelled version.

Labels in λ_{lca} and paths in the call-by-name translation

The particularities of the calculus are best understood via the correspondence to the call-by-name translation. Thus, let us move directly to the translation of terms into call-by-name proof nets. We provide a simplified presentation where instead of placing a translated label on a wire, we simply place the label itself. As before, multiplicative information is kept implicit via overlining and underlining. Hence, the general form of the translation takes a labelled term and places the label of the term (when it has one) at the root of the graph: We give the translation in Figure 4. Remark that the translation of an argument (and the substitution) always involve a box structure. We do not repeat translations for δ and ε-terms since these translate in the same way as before.

The following theorem is the main result of this section. It establishes a correspondence between labels and paths in the call-by-name proof net translation.

Theorem 2. *1. If $T \to_{lca} T'$ then $W_{G(T)} = W_{G(T')}$, where W_G is defined as in Theorem 1.*

2. Suppose T is a term obtained by erasing the labels and $\to_{ca}$ is the system generated by the rules for λ_{lca} by removing the labels. If $T \to_{ca} T'$ then $G(T) \Rightarrow^{*} G(T')$ using closed cut elimination, where G is the call-by-name translation.

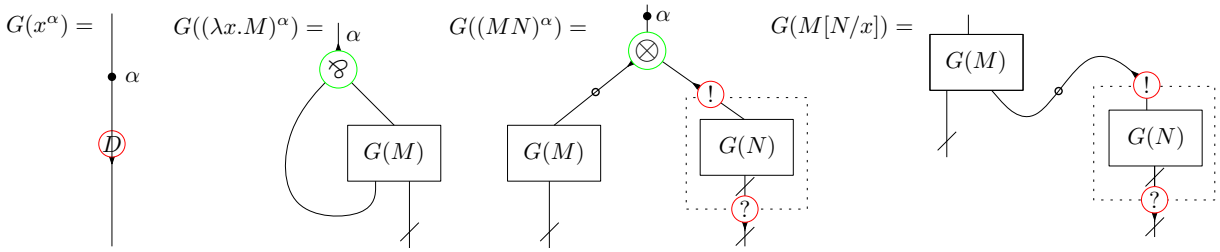


Figure 4: Translation of λ_c into call-by-name nets

This theorem is stronger than Theorem 1: there is a correspondence between labels in λ_{lca} and paths in the call-by-name proof net translation, and between the dynamics of the calculus and the proof-net dynamics of closed cut elimination $\Rightarrow$ (due to space constraints we omit the definition of $\Rightarrow$ and refer to [21]). Rules *App1, Lam, Cpy2 and Ers2* correspond to identities while the remaining rules correspond to single step graph rewriting. To obtain a similar result for λ_{lcf} with the call-by-value translation, we need to impose more conditions on the substitution rules (requiring closed values instead of simply closed terms).

6 Conclusions and future work

We have investigated labelled λ-calculi with explicit substitutions. The labels give insight into how the dynamics of a λ-calculus corresponds to building paths in proof nets, and they also allow us to understand better the underlying calculi. For instance, label insensitive substitution rules witness that some actions in the calculi capture "less essential" computations, that is, an additional price for bureaucracy of syntax is paid in relation to the corresponding proof-net dynamics. From this point of view, it is interesting to ask about whether strategies for these calculi exist such that each propagation of substitution corresponds to stretching a path in a corresponding proof-net. On the other hand, investigation of new labelled versions of known strategies defined for the underlying calculi could help in understanding and establishing requirements for new proof-net reduction strategies.

The use of closed reduction in the current work simplifies the computation of the labels, since only closed substitutions are copied/erased. The methodology can be extended to systems that copy terms with free variables, but one would need to use global functions to update the labels; instead, using closed reduction, label computations are local, in the spirit of the geometry of interaction. Notice that duplication (resp. erasing) of free variables causes further copying (resp. erasing), which in our case would require on the fly instantiation of additional δ-terms (resp. ε-terms). Without implying that such calculi need to be confluent, we remark that the system without the closed conditions introduces non joinable critical pairs in the reduction rules resulting in non-confluent systems. Notice that path computation in the GoI has only been shown sound for nets that do not contain auxiliary doors.

The main results of this paper establish that the labels are adequate enough for the representation of paths in proof-nets. This makes these calculi appealing for intermediate representation of implementations of programing languages where target compilation structures are linear logic proof-nets.

For the study of shared reductions in proof nets, and in the calculus itself, a few additions would be useful: it would be certainly interesting to track not only cuts that correspond to *Beta*-redexes but also exponential cuts corresponding to substitutions. For this, we should allow copy, erase and substitution terms to bear labels. These additions could also be useful towards obtaining a standardisation result for closed reduction calculi.

References

[1] M. Abadi, L. Cardelli, and P. L. Curien. Explicit substitutions. *Journal of Functional Programming*, 1:31–46, 1991.

[2] S. Abramsky. Computational Interpretations of Linear Logic. *Theoretical Computer Science*, 111:3–57, 1993.

[3] A. Asperti, V. Danos, C. Laneve, and L. Regnier. Paths in the lambda-calculus. In *Logic in Computer Science*, pages 426 – 436, 1994.

[4] A. Asperti and C. Laneve. Paths, computations and labels in the lambda-calculus. In *RTA-93: Selected papers of the fifth international conference on Rewriting techniques and applications*, pages 277–297, Amsterdam, The Netherlands, The Netherlands, 1995. Elsevier Science Publishers B. V.

[5] A. Asperti and S. Guerrini. *The optimal implementation of functional programming languages*. Cambridge University Press, 1998.

[6] H. P. Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[7] V. Danos. *La Logique Linéaire appliquée a l'étude de divers processus de normalisation (et principalement du λ-calcul).* PhD thesis, Universite Paris VII, 1990.

[8] N. Çağman and J. R. Hindley. Combinatory weak reduction in lambda calculus. *Theor. Comput. Sci.*, 198(1-2):239–247, 1998.

[9] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Department of Mathematics, Eindhoven University of Technology. T.H.-Report 78-WSK-03, 1978.

[10] M. Fernández and I. Mackie. Closed reductions in the lambda-calculus. In *CSL '99: Proceedings of the 13th International Workshop and 8th Annual Conference of the EACSL on Computer Science Logic*, pages 220–234, London, UK, 1999. Springer-Verlag.

[11] M. Fernández, I. Mackie, and F.-R. Sinot. Closed reduction: explicit substitutions without α-conversion. *Mathematical. Structures in Comp. Sci.*, 15(2):343–381, 2005.

[12] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[13] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In C. Bonotto, R. Ferro, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, pages 221–260. North-Holland, 1989.

[14] J.-J. Levy. *Reductions correctes et optimales dans le lambda-calcul*. PhD thesis, These de doctorat d'etat, Universite Paris VII, 1978.

[15] I. Mackie. Interaction nets for linear logic. *Theor. Comput. Sci.*, 247(1-2):83–140, 2000.

[16] I. Mackie. *The Geometry of Implementation*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, September 1994.

[17] I. Mackie. The geometry of interaction machine. In *POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 198–208, New York, NY, USA, 1995. ACM.

[18] M. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[19] B. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.

[20] N. Siafakas. A fully labelled lambda calculus: Towards closed reduction in the geometry of interaction machine. In *Proceedings of Developments in Computation Models 2006. Electron. Notes Theor. Comput. Sci.*, 171(3):111–126, 2007.

[21] N. Siafakas. A framework for path-based computation in functional programming languages. PhD thesis, King's College London, University of London, 2008. Available from `http://www.dcs.kcl.ac.uk/pg/siafakas/thesis.pdf`

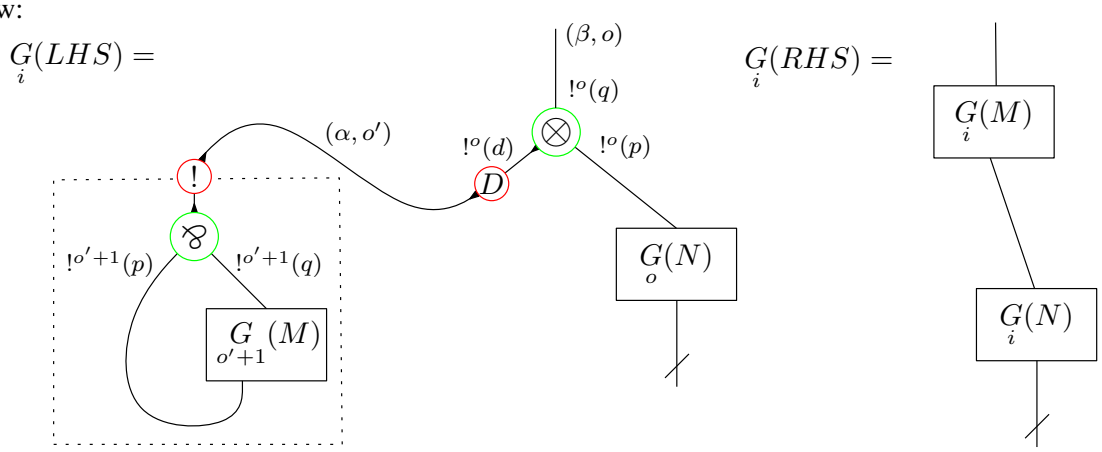
APPENDIX

Proof of Theorem 1

Theorem A-1. Let $W_G = \{w(\phi) \mid \phi \in \text{straight paths of } G\}$ denote the set of weights of straight paths observable in a graph G and let T be a labelled term. If $T \to T'$ then $W_{G(T)} = W_{G(T')}$.

Proof. By induction on T. The only interesting case is when the reduction takes place at the root position. We show the property by cases on the rule applied.

Case Beta: We give the graphical representation of the left- and right-hand sides of the *Beta*-rule below:



Notice that there are no auxiliary doors since the function must be closed. We must show that weights of straight paths in the left hand side are found in the right hand side, that is, the weights of

1. $\phi = \rho - (\beta, o) - !^o(q) - !^o(d) - (\alpha, o') - !^{o'+1}(q^*) - \mu$ where ρ is a path ending at the root of the left hand side and μ is a path starting in M;
2. $\psi = \mu' - !^{o'+1}(p) - (a^*, o') - !^o(d^*) - !^o(p^*) - \nu$ with μ' ending in the (translation of) free variable of M and ν starting in N; and
3. ψ^r, ϕ^r

are found in the right hand side of the figure. Notice that in the right hand side, the translation of a substitution term does not place any label at the immediate root of the graph and there seems to be a kind of mismatch with the level numbers in which the subgraphs are called, however this is correct. Since we do not have labels on substitution terms, the translation will respect the $i's$ later on, that is, we can apply the induction hypothesis. We show the property by checking that the external label of M fixes the level number yielding the weight we are after.

- We have $l = label\ M = \beta\, \overrightarrow{\mathsf{D}\, \alpha\, !}\, \cdot \beta'$, where β' is some suffix. The weight of l is given by

$$lw[\![\beta\, \overrightarrow{\mathsf{D}\, \alpha\, !}\, \beta']\!]^i = w((\beta, o) - !^o(q) - !^o(d) - (\alpha, o') - !^{o'+1}(q^*)) \cdot lw[\![\beta']\!]^{o'+1}$$

  which completes the first case.

- We work in a similar fashion with the second case where we first translate $label\ N = \underline{(\overrightarrow{\mathsf{D}\, \alpha\, !})^r \gamma}$. Under the assumption that the wire connecting the free variable x is at level $o+1$ we have

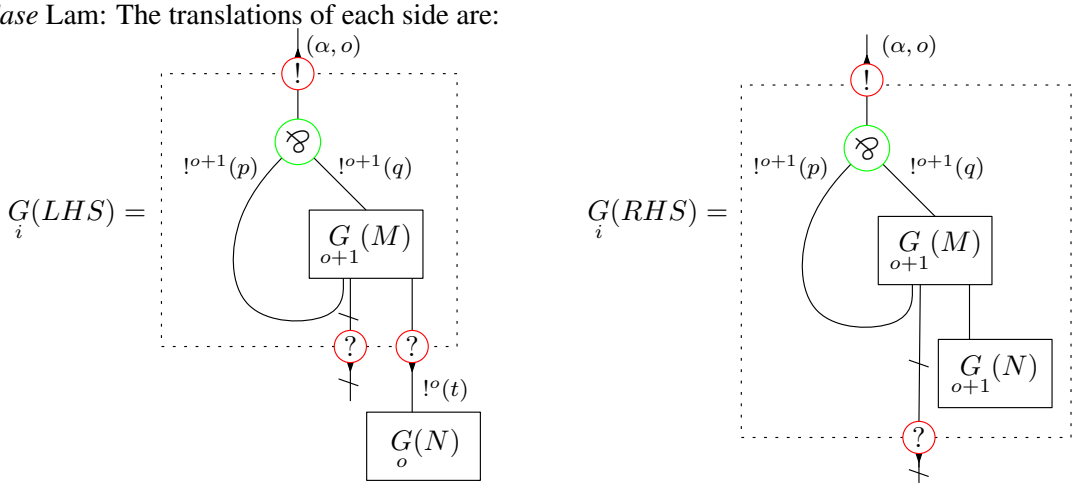
$$lw[\![\underline{(\overrightarrow{\mathsf{D}\, \alpha\, !})^r \gamma}]\!]^i = w(!^{o'+1}(p) - (a^*, o') - !^o(d^*) - !^o(p^*)) \cdot lw[\![\gamma]\!]^o$$

The translation calls with a suitable i big enough to cover the open scopes at the first label and returns the open scopes at the last label. Now in ψ, reading a in reverse means that the indicated scope number is the one for the first label and hence decreases.

One argues in the same way for the reverse cases of ϕ and ψ.

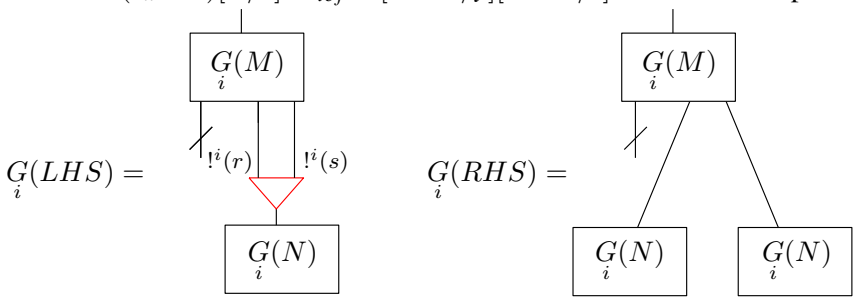
Case Var*:* This is where two paths meet and get glued via an axiom-link. The translation of the left- and right-hand sides do not tell us anything useful since we have just wires (identities) and thus the weights of paths remain the same.

Case Lam: The translations of each side are:



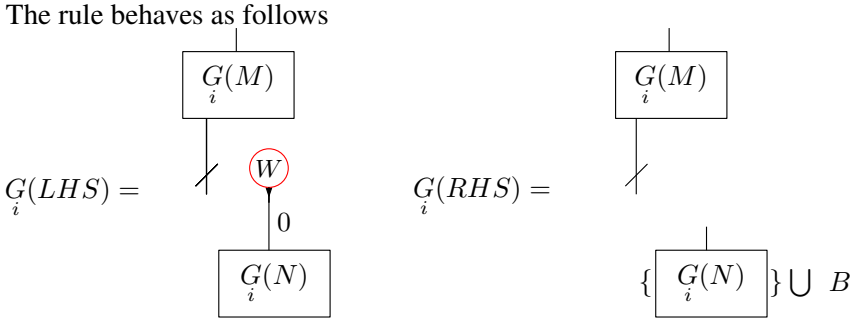
Recall that the external label of N may give us a prefix of exponential markers followed by an underline or just an underline. The interesting point is that the right hand side suggests that the translation is called with $o+1$ for N and looks like a source of a mismatch. But the external label on N in the rhs must start with an exponential marker ? and this decreases the o upon which N is translated. Thus the weights of paths remain the same and this completes the case.

Case Cpy1*:* We have $(\delta_x^{y,z}.M)[N/x] \to_{lcf} M[\overrightarrow{\mathsf{R}} \bullet N/y][\overrightarrow{\mathsf{S}} \bullet N/z]$ with the corresponding translations



We argue as before and there is nothing to say about the levels. The case is similar for erasing.

Case Ers1*:* The rule behaves as follows



There are killed paths in the left hand side and we have the same on the right. Note that erased paths do not survive reduction but one may walk these with a strategy. This is the reason for keeping the set B, which is initially empty.

The translation of rules App1, App2, Cpy2, Ers2 and Cmp correspond to equalities. □

Proof of Theorem 2

Lemma A-1. If $T = M[N/x]$ then there is a decomposition $(label\ N) = \omega\underline{\alpha}\overrightarrow{!}\sigma$ such that ω is a prefix built with exponential markers having the shape $\overrightarrow{E}_1 \ldots \overrightarrow{E}_n$ where $n \geq 0$ and E is not a D marker.

Proof. Since the term N belongs to a substitution term, its external label must have been generated by a *Beta* rule generating the sub-label $\underline{\alpha}\overrightarrow{!}\sigma$. By using the rules in σ, we can generate only an ω prefix. It cannot contain a dereliction marker since this can come from the *Var* rule which stops the process; that is, the root of T is not a substitution term anymore. □

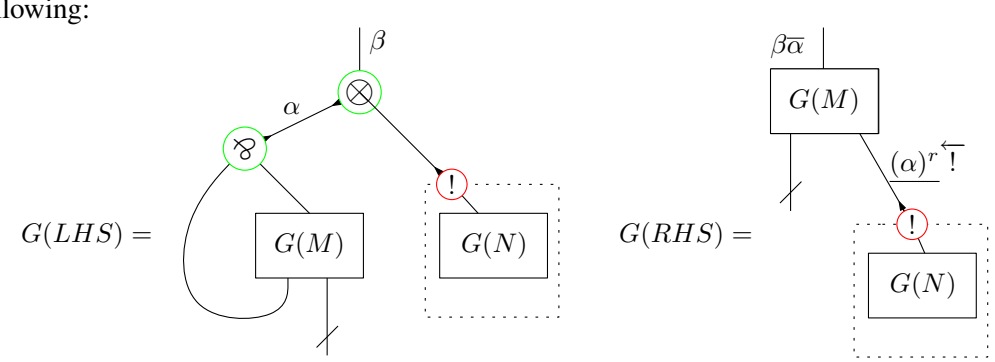
The consequence of the lemma is that derelictions are followed directly by exponentials. Notice that this is different form the previous system.

Theorem A-2.

1. If $T \to_{lca} T'$ then $W_{G(T)} = W_{G(T')}$.
2. Suppose T is a term obtained by erasing the labels and $\to_{ca}$ is the system generated by the rules for λ_{lca} by removing the labels. If $T \to_{ca} T'$ then $G(T) \Rightarrow G(T')$ using closed cut elimination, where G is the call-by-name translation.

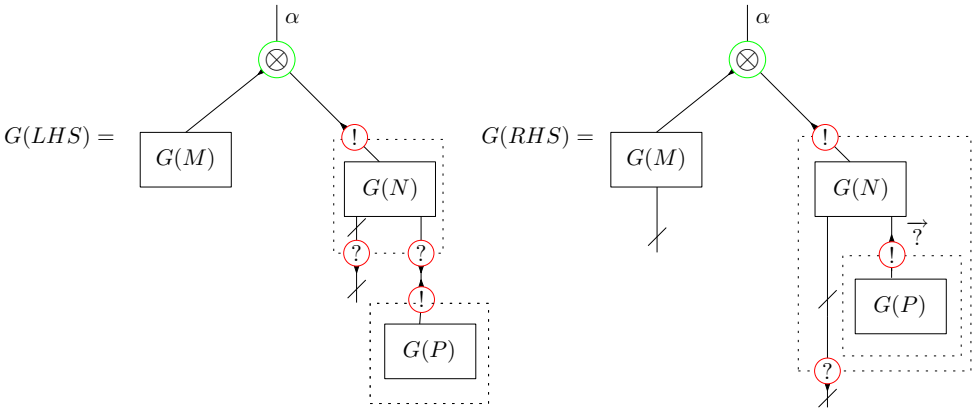
Proof. We proceed by cases and argue about both properties.

Case Beta*:* $((\lambda x.M)^{\alpha}N)^{\beta} \to_{lca} \beta \bullet \overline{\alpha}M[\underline{(\alpha)^r}\overleftarrow{!} \bullet N/x]$ *if* $\mathsf{fv}(N) = \emptyset$ where $\mathsf{fv}(N) = \emptyset$. The situation is the following:



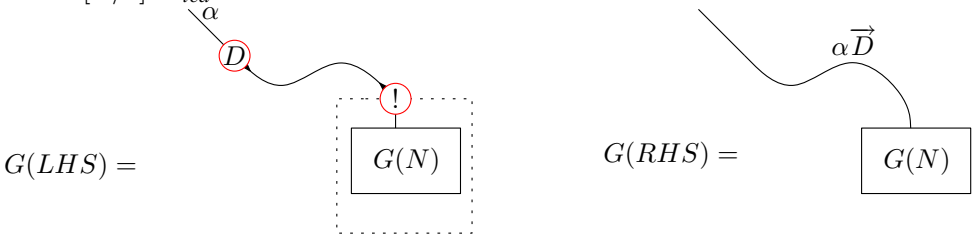
Two paths are of interest: one entering from the root, moving along the cut and ending at the root of $G(M)$ and one coming from the free variable, travelling along the cut and ending up at the root of $G(N)$. Both weights are preserved in the right hand side. The second point of our claim is satisfied since the closed cut elimination sequence corresponds to one multiplicative cut.

Case App2*:* $(MN)^{\alpha}[P/x] \to_{lca} (MN[\overrightarrow{?} \bullet P/x])^{\alpha}$ where $x \in \mathsf{fv}(N)$ and the translation of both sides become



For the first point in our claim, weights are preserved by recording the auxiliary marker and for the second point, the graph rewrite corresponds to a closed commutative cut. Notice that there are no closedness conditions on the σ-rules, but since *Beta* is the only rule that can generate a substitution, it must be a closed one.

Case Var*:* $x^{\alpha}[N/x] \rightarrow_{lca} \alpha\,\overrightarrow{\mathsf{D}} \bullet N$ and the situation becomes



There seems to be a mismatch with the graph rewriting rule and paths that we record since we removed the box in the right hand side. However, by Lemma A-1, the external label of the argument must have the shape $(label\ N) = \omega\underline{\alpha}\,\overrightarrow{!}\,\sigma$ such that ω is built with exponential markers and the trailing exponential box marker restores our level information. Regarding our second point, this simply corresponds to a closed dereliction cut.

Case Cpy1, Ers1*:* With respect to the paths, both cases are similar as before. Regarding the second point of the claim, the translations correspond to a closed contraction and weakening cut respectively.

For the remaining cases, the left and right hand side translations are identical. □