Variations and interpretations of naturality in call-by-name lambda-calculi with generalized applications

José Espírito Santo[a], Maria João Frade[b], Luís Pinto[a]

[a] *Centre of Mathematics, University of Minho, Portugal*
[b] *HASLab/INESC TEC & University of Minho, Portugal*

Abstract

In the context of intuitionistic sequent calculus, "naturality" means permutation-freeness (the terminology is essentialy due to Mints). We study naturality in the context of the lambda-calculus with generalized applications and its multiary extension, to cover, under the Curry-Howard correspondence, proof systems ranging from natural deduction (with and without general elimination rules) to a fragment of sequent calculus with an iterable left-introduction rule, and which can still be recognized as a call-by-name lambda-calculus. In this context, naturality consists of a certain restricted use of generalized applications. We consider the further restriction obtained by the combination of naturality with normality w.r.t. the commutative conversion engendered by generalized applications. This combination sheds light on the interpretation of naturality as a vectorization mechanism, allowing a multitude of different ways of structuring lambda-terms, and the structuring of a multitude of interesting fragments of the systems under study. We also consider a relaxation of naturality, called weak naturality: this not only brings similar structural benefits, but also suggests a new "weak" system of natural deduction with generalized applications which is exempt from commutative conversions. In the end, we use all of this evidence as a stepping stone to propose a computational interpretation of generalized application (whether multiary or not, and without any restriction): it includes, alongside the argument(s) for the function, a *general list* – a new, very general, vectorization mechanism, that structures the continuation of the computation.

Keywords: Natural deduction, Sequent calculus, Permutative conversion, Commutative conversion, Normal and natural proofs, Vector notation

1. Introduction

The λ-calculus with generalized applications [1], here denoted $\lambda\mathbf{J}$, corresponds, in its simply-typed version, to natural deduction with general elimination rules [2]. The generality added to the elimination rules engenders a commutative conversion π (very much like what happens with disjunction), and this conversion, in turn, allows the definition of a "fully normal" form for natural deduction proofs, not available with the usual natural deduction system. Such normal proofs are in 1-1 correspondence with the cut-free proofs of the intuitionistic sequent calculus – hence the name $\lambda\mathbf{J}$, a reminder of LJ.

The usual natural deduction system (which corresponds to the simply-typed λ-calculus) has a less straighforward connection with the sequent calculus. In the "folklore" view, two proofs of the intuitionistic sequent calculus determine the same natural deduction proof if and only if they are inter-permutable, that is one can be obtained from the other by permutations of inferences [3, 4]; and the possibility of permuting inferences is an obstacle to the computational interpretation of the sequent calculus along the lines of the Curry-Howard isomorphism [5]. In the cut-free setting, a subset of "normal" sequent calculus proofs can be identified in 1-1 correspondence with normal natural deduction proofs [6, 7, 3]. The terminology "normal", due to Mints, is thus justified. Moreover, such "normal" sequent calculus proofs are permutation-free, as they consist of one representative per each class of inter-permutable proofs [3]. In [8], this 1-1 correspondence is extended to an isomorphism between β-reduction in the λ-calculus and an appropriate cut-elimination rule whose normal forms are Mints' normal proofs.

In a series of papers [9, 10, 11, 12, 13], this state of affairs was extended to the setting of a sequent calculus with cuts, presented as a certain extension of $\lambda\mathbf{J}$ – its *multiary* version, named $\lambda\mathbf{Jm}$. This extension allows a more faithful modeling of reasoning in the sequent calculus, where the possibility of inferring on the left-hand-side of the sequent symbol is on a par with inference on the other side. In fact, $\lambda\mathbf{J}$ captures left-introduction in a form that is not iterable. This is not surprising: after all, $\lambda\mathbf{J}$ corresponds to a form of natural deduction, and the latter just infers on the right-hand-side (this point will be developed below with more detail). But the extension comes with a price: a new conversion μ [14], which is neither π, nor any permutative conversion γ of the sequent calculus that produces Mint's normality.

In a setting with cuts, Mints' normality has to be extended to a concept of *naturality*, so that the normal sequent calculus proofs are those that are natural and cut-free [13]. This paper is about naturality, specificaly how naturality and its variants help in structuring $\lambda\mathbf{Jm}$ and its numerous subsystems, and how the computational interpretation of naturality and its vari-

ants builds a scafold to reach a computational interpretation of the generality added to the application constructor in $\lambda\mathbf{J}$ and $\lambda\mathbf{Jm}$.

Naturality consists of a certain restricted use of generalized applications, as a mechanism for the vectorization of arguments in function application. We consider the further restriction obtained by the combination of naturality with π-normality. This combination determines subsystems which may be interpreted as *formal vector notations*, and which are coupled with isomorphic λ-calculi for which the vector notation is offered. Two examples of this situation will be given. One may be seen as the multiary extension of the above mentioned isomorphism from [8] involving the λ-calculus. The other involves a relaxation of naturality, called weak naturality, and a new system of natural deduction with a restricted form of generalized applications which is exempt from commutative conversions.

The isomorphism between the system of weak natural deduction and its vector notation signals that the generality of generalized applications may be used as a vectorization mechanism beyond the natural fragment. This leads us to go all the way and propose a computational interpretation of generalized application in $\lambda\mathbf{J}$ and $\lambda\mathbf{Jm}$: it includes, alongside the argument(s) to be passed to the function, a *general list* – a new, very general, vectorization mechanism, which structures the computation after function application, and which is governed by some of the rules of $\lambda\mathbf{J}$ or $\lambda\mathbf{Jm}$ after an appropriate reinterpretation.

Structure of the paper. Section 2 recalls system $\lambda\mathbf{J}$ and naturality and gives a basic example of isomorphism between the λ-calculus and a formal vector notation. Section 3 introduces weak naturality and a generalization of the previous isomorphism, holding for the weakly natural extension of the λ-calculus. Section 4 recalls system $\lambda\mathbf{Jm}$ and naturality in the multiary setting, and gives the multiary generalization of the first isomorphism. Section 5 gathers all the systems and maps in a coherent picture, and discusses the new system of natural deduction and how the global picture points to a new computational interpretation of generalized applications. Section 6 summarizes the contributions of this paper and suggests directions for future work.

Figure 1: Typing rules for $\lambda\mathbf{J}$

$$\frac{}{x\!:\!A,\Gamma\vdash x\!:\!A}\ \mathit{Axiom} \qquad \frac{x\!:\!A,\Gamma\vdash t\!:\!B}{\Gamma\vdash\lambda x.t\!:\!A\supset B}\ \mathit{Right} \qquad \frac{x:B,\Gamma\vdash v\!:\!C}{\Gamma|B\vdash(x)v\!:\!C}\ \mathit{Select}$$

$$\frac{\Gamma\vdash t\!:\!A\supset B \quad \Gamma;A\supset B\vdash a\!:\!C}{\Gamma\vdash ta\!:\!C}\ \mathit{Cut} \qquad \frac{\Gamma\vdash u\!:\!A \quad \Gamma|B\vdash c\!:\!C}{\Gamma;A\supset B\vdash(u,c)\!:\!C}\ \mathit{Left}$$

2. Generalized applications

2.1. Recapitulation

System $\lambda\mathbf{J}$. Expressions E of $\lambda\mathbf{J}$ are generated by the following grammar:

$$\begin{array}{rrcl} \text{(proof terms)} & t,u,v & ::= & x \mid \lambda x.t \mid ta \\ \text{(J-arguments)} & a & ::= & (u,c) \\ \text{(continuations)} & c & ::= & (x)v \end{array}$$

As usual, *values*, ranged over by V, are variables or λ-abstractions.

We identify simple types with formulas of intuitionistic, propositional, implicational logic, ranged over by A, B, C, D. Contexts Γ are sets of variable declarations $x : A$, with at most one declaration per variable. The typing rules are in Fig. 1. They handle three kinds of sequents, one per syntactic class:

$$(i)\quad \Gamma\vdash t\!:\!A \qquad (ii)\quad \Gamma;A\supset B\vdash a\!:\!C \qquad (iii)\quad \Gamma|C\vdash c\!:\!D\ . \tag{1}$$

$\lambda\mathbf{J}$ is equipped with the reduction relations $\to_\beta$ and $\to_\pi$, given by the compatible closure of the base rules in Fig. 2. Rule β employs ordinary substitution, denoted with $\mathbf{s}$, and calls functions $\lambda x.t$ according to the *call-by-name* paradigm [1]. Rule π employs the *append* operation $a'@a$, defined simultaneously with $c@a$ and $t@a$ as follows:

$$(u,c)@a=(u,c@a) \quad ((x)t)@a=(x)(t@a) \quad V@a=Va \quad (ta')@a=t(a'@a)$$

The π-rule adopted in this paper is an eager version of the original one of ΛJ [1] (even of its variant π' considered in [10, 13]) [2], but retains its

[1] For the call-by-value version of $\lambda\mathbf{J}$ see [15]

[2] The original π rule reads: $t(u,(x)v)a \to t(u,(x)va)$. It simulates the variant of this paper as follows: $t(u,(x)v)a' \to_\pi t(u,(x)(va')) \to_\pi^* t(u,(x)(v@a')) = t((u,(x)v)@a')$, where for the penultimate step one observes by induction on t that $ta \to_\pi^* t@a$.

Figure 2: Reduction rules of $\lambda\mathbf{J}$

$$\begin{array}{lrcl} (\beta) & (\lambda x.t)(u,(y)v) & \to & \mathbf{s}(\mathbf{s}(u,x,t),y,v) \\ (\pi) & (ta)a' & \to & t(a@a') \end{array}$$

good properties. In particular, it induces a strongly normalising, and confluent rewriting system, whose normal forms can be calculated by a mapping π given homomorphically, except for the clause $\pi(ta) = (\pi t)@(\pi a)$. Confluence and strong normalisation of the reduction relation $\to_{\beta\pi}$ also holds (the proofs of the corresponding results in [10] for the variant π' are easily adapted to deliver proofs for the rule π adopted in the present paper).

λ-calculus. Let the λ-terms be given by the grammar

$$t, u, v \ ::= \ x \,|\lambda x.t \,|\, tu$$

There is a trivial encoding $(\_)^\bullet$ of λ-terms as $\lambda\mathbf{J}$-terms, given by $(tu)^\bullet = t^\bullet(u^\bullet,(x)x)$. This map establishes a bijection between the set of λ-terms and the set of *trivial* $\lambda\mathbf{J}$-terms, the latter being characterized by being restricted to J-arguments of the form $(u,(x)x)$, with *trivial continuation* $(x)x$. Such trivial terms are obviously closed under substitution, and hence closed under β-reduction; and obviously such β-reduction of trivial terms is just a rephrasing of β-reduction of λ-terms, with ordinary arguments u written $(u,(x)x)$. We will not distinguish between λ-terms and their counterpart as trivial $\lambda\mathbf{J}$-terms. In this way, λ-terms form a proper subset of $\lambda\mathbf{J}$. More generally, when writing a term $t(u,c)$, we may omit c and write $t(u)$, or just tu, when c is trivial. The subset of trivial terms does not constitute a subsystem of $\lambda\mathbf{J}$, because it is not closed under π.

2.2. Naturality

Even if $\lambda\mathbf{J}$ is meant as a system of proof terms for natural deduction with general elimination rules [1, 2], the $\beta\pi$-normal forms correspond to the cut-free LJ proofs – and this actually explains the name "$\lambda\mathbf{J}$" [1]. Such normal forms are thus a vehicle to study the phenomenon of permutability in sequent calculus proofs, and one can characterize with them a class of cut-free sequent calculus proofs, named *normal*, in 1-1 correspondence with normal natural deductions [3, 6]. In [13] the terminology "natural" was introduced, so that $t \in \Lambda J$ is "normal" iff t is "natural" and a $\beta\pi$-normal form. Naturality is a central concept in the present paper.

Definition 1 (Natural lists). Natural *arguments and lists and* x-natural *terms are defined simultaneously as follows:*

- *An argument* a *is natural if* $a = (u, c)$ *and* c *is a natural list.*
- *A continuation* c *is a natural list if* $c = (x)v$ *and* v *is* x*-natural.*
- *A term* v *is* x*-natural if* $v = x$ *or:* $v = xa$ *and* $x \notin a$ *and* a *is natural.*

We let L range over natural lists. Every natural list L is either $\mathbf{nil} := (x)x$ or $(u{+}L) := (x)x(u, L)$, with $x \notin u, L$.

System $\lambda\mathbf{n}$ and its subsystem $\vec{\lambda}\mathbf{n}$. An expression of $\lambda\mathbf{J}$ is *natural* if all continuations occurring in it are natural lists. Alternatively, natural expressions are generated by the following grammar:

$$\begin{array}{rrcl} \text{(natural proof terms)} & t, u, v & ::= & x \mid \lambda x.t \mid ta \\ \text{(natural J-arguments)} & a & ::= & (u, L) \\ \text{(natural continuations)} & L & ::= & (x)v, \text{ with } v \text{ } x\text{-natural} \end{array}$$

Notice that a natural continuation is a natural list, but not conversely: in a natural continuation $(x)v$, v is not only x-natural, but also natural.

This syntax is closed for substitution: $\mathbf{s}(u, x, t)$ is natural if u, t are; and for a natural argument a, one sees by simultaneous induction on natural L and a' and x-natural v that the continuation $L@a$ and the argument $a'@a$ are natural and that the term $v@a$ is x-natural when $x \notin a$. Therefore, this syntax is closed for the reduction rules β and π and constitutes a subsystem of $\lambda\mathbf{J}$ called $\lambda\mathbf{n}$.

The π-normal forms of $\lambda\mathbf{n}$ are characterized by requiring t in ta to be a value V. Such normal forms are still closed under β, provided β is redefined as employing, not ordinary substitution $\mathbf{s}(u, x, E)$, but instead a special substitution $\mathbb{S}(u, x, E)$. The latter is defined exactly as ordinary substitution, except that $\mathbb{S}(u, x, Va) = \mathbb{S}(u, x, V)@\mathbb{S}(u, x, a)$.[3] Preservation of π-normal forms by the special substitution use the fact that also the append operations preserve them, and of course special substitution preserves natural expressions. The π-normal forms of $\lambda\mathbf{n}$, equipped with β, constitute the subsystem $\vec{\lambda}\mathbf{n}$ of $\lambda\mathbf{n}$.

[3]Notice $\mathbb{S}(u, x, V)\mathbb{S}(u, x, a)$ is not guaranteed to be π-normal. This exceptional equation can be written differently, as in [13]: $\mathbb{S}(u, x, xa) = u@\mathbb{S}(u, x, a)$; and $\mathbb{S}(u, x, Va) = \mathbb{S}(u, x, V)\mathbb{S}(u, x, a)$, if $V \neq x$.

2.3. The basic isomorphism

The subsystem $\vec{\lambda}\mathbf{n}$ is isomorphic to the ordinary λ-calculus, in the sense that there is a bijection between the sets of λ-terms and the π-normal forms of natural proof terms which is an isomorphism between the respective β-reduction relations. This fact is a cornerstone for the present paper. It is also an unsurprising fact, when these systems are put in a wider context involving the sequent calculus - we will return to this point. For now, the isomorphism can be motivated as an alternative encoding of λ-terms as $\lambda\mathbf{J}$-terms whose codes, contrary to the trivial encoding seen before, form a subsystem of $\lambda\mathbf{J}$.

The sensitive point in the encoding of λ-terms is not the encoding of values. As to application, the trivial encoding of the λ-term $Vu_1 \cdots u_m$ is $V(u_1) \cdots (u_m)$. Here $(u_i) = (u_i, (x)x)$. Let $a_i := (u_i)$. The alternative encoding is Va where a is the natural list $(u_1, \cdots, u_m) := a_1@ \cdots @a_m$.[4]

Formally, the encoding t° of a λ-term t is defined simultaneously with $(t, R)^\circ$, for t a λ-term and R a natural J-argument, as follows:

- $x^\circ = x$ and $(\lambda x.t)^\circ = \lambda x.t^\circ$ and $(tu)^\circ = (t, (u^\circ, (z)z))^\circ$
- $(V, R)^\circ = V^\circ R$ and $(tu, R)^\circ = (t, (u^\circ, (z)zR))^\circ$, with $z \notin R$.

The inverse map is as follows. For a $\vec{\lambda}\mathbf{n}$-term t, Φt is defined simultaneously with $\Phi(t, R)$, for t a λ-term and R a natural J-argument, as follows:

- $\Phi x = x$ and $\Phi(\lambda x.t) = \lambda x.\Phi t$ and $\Phi(VR) = \Phi(\Phi V, R)$
- $\Phi(t, (u, (x)x)) = t\Phi u$ and $\Phi(t, (u, (x)xR)) = \Phi(t\Phi u, R)$.

These maps not only are each other's inverses, but also establish an isomorphism between the relations $\to_\beta$ in λ and $\vec{\lambda}\mathbf{n}$. We collect these facts in the following:

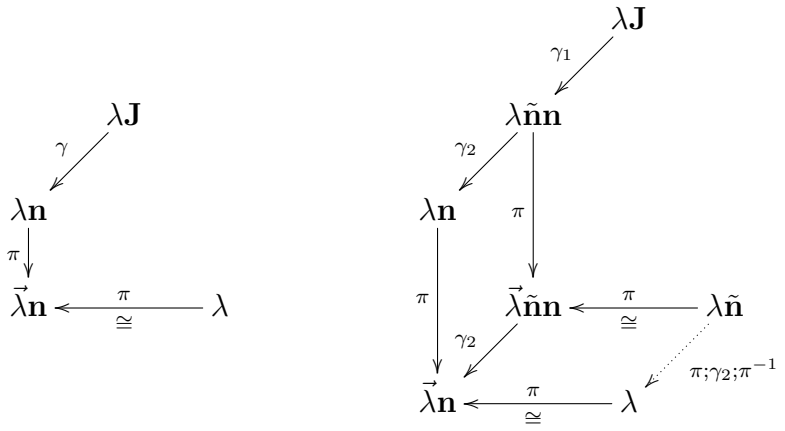
Theorem 1. $\lambda \cong \vec{\lambda}\mathbf{n}$.

As said, this will follow from know facts about the sequent calculus. In addition, this fact will be immediately generalized in the following section.

We can summarize the situation so far in the left diagram of Fig. 3. As the diagram suggests, t° is the π-normal form $\pi(t)$ of a λ-term t. This is proved by induction on t, using the fact $(t, R)^\circ = t^\circ@R$, the latter also proved by induction on t. For the map $\gamma : \lambda\mathbf{J} \to \lambda\mathbf{n}$, see the next section.

[4] Notice the append of natural J-arguments is associative.

Figure 3: From naturality to weak naturality



Example - part 1. Let us illustrate (non-)naturality starting from the paradigmatic example of a λ-term $t_1 := t_0u_1u_2$, with $t_0 := \lambda x.t$. At the end of Section 2.1 we agreed that $\lambda \subset \lambda\mathbf{J}$ - indeed we have $\lambda \subset \lambda\mathbf{n}$. For instance, t_1 is the natural term $t_0(u_1, (x_1)x_1)(u_2, (x_2)x_2)$. A single π-reduction step $t_1 \to_\pi t_0(u_1, u_2) = t_0(u_1, (x_1)x_1(u_2, (x_2)x_2)) =: t_2$ leads us beyond the set of λ-terms. The term t_2, still in $\lambda\mathbf{n}$, is an application with continuation $c_1 := (x_1)x_1(u_2, (x_2)x_2)$, which is a natural list (notice $x_1 \notin u_2$). If t_1 has type B, then t_0 has type of the form $A_1 \supset A_2 \supset B$, the first argument u_1 has type A_1 and

$$\lfloor A_2 \supset B \vdash c_1 : B \qquad \vdash u_2 : A_2 \ .$$

Naturality is lost if, for instance, c_1 is $(x_1)v_1(u_2, (x_2)x_2)$, with $v_1 \neq x_1$. In the most general case of this situation, x_1 is allowed to occur in v_1 and u_2. Again, let $t_2 = t_0(u_1, c_1)$, still of type B. Then, for some type C, t_0 has type $A_1 \supset C$ and

$$\lfloor C \vdash c_1 : B \qquad x_1 : C \vdash v_1 : A_2 \supset B \qquad x_1 : C \vdash u_2 : A_2 \ .$$

The term v_1 is a "bridge" between the types C and $A_2 \supset B$. In the natural case, when $v_1 = x_1$ and x_1 does not occur afterwards, v_1 acts as a "link" to the second argument u_2. But, in the general case, v_1 is itself involved in an application, namely v_1u_2, and the continuation $c_1 = (x_1)v_1u_2$ is a parametric version of such application.

3. Weak naturality

We now want to develop the situation depicted in the right diagram of Fig. 3. First we introduce a relaxation of the concept of naturality, which

determines a subsystem $\lambda\tilde{\mathbf{n}}\mathbf{n}$ of $\lambda\mathbf{J}$. Then we can form a system $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$ consisting of the π-normal forms of $\lambda\tilde{\mathbf{n}}\mathbf{n}$ equipped with a rule β defined as in $\lambda\mathbf{J}$, except that it employs a special substitution operation - this generalizes the system $\vec{\lambda}\mathbf{n}$ seen before. There is an extension $\lambda\tilde{\mathbf{n}}$ of the λ-calculus with which we can obtain the isomorphism $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n} \cong \lambda\tilde{\mathbf{n}}$, a generalization of $\vec{\lambda}\mathbf{n} \cong \lambda$.

3.1. A relaxation of naturality

Let us analyze $\vec{\lambda}\mathbf{n} \cong \lambda$. An application ta in λ uses a trivial argument $a = (u, c)$ with a trivial continuation $c = (x)x$. In $\vec{\lambda}\mathbf{n}$, application Va' is constrained to have a value in the function position, but, as a compensation, the argument $a' = (u, L)$ is a natural one, with a natural continuation L, living in a class of continuations that can be obtained in this way: take the trivial continuations, and close them under the operation $c \mapsto c@a$, for a a trivial argument.

Now, we generalize this picture, starting, not from trivial continuations, but from continuations $(x)V$. Let us call an argument of the form $(u, (x)V)$ *weakly trivial*. This determines an extension $\lambda\tilde{\mathbf{n}}$ of the λ-calculus where applications have the form ta, with weakly trivial a. The isomorphic representation, in $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$, has applications Va', where $a' = (u, \mathbb{L})$, with $\mathbb{L}$ a continuation living in the class obtained by closing the weakly trivial continuations under the operation $c \mapsto c@a$, for a a weakly trivial argument. In this way we obtain a more general class of continuations than that of natural lists.

Definition 2 (Weakly natural lists). Weakly natural *arguments and lists and* weakly x-natural *terms and values are defined simultaneously as follows:*

- *An argument $a = (u, c)$ is weakly natural if c is a weakly natural list.*
- *A continuation $c = (x)v$ is a weakly natural list if v is weakly x-natural.*
- *A term v is weakly x-natural if $v = V$ and V is weakly x-natural or: $v = Va$ and $x \notin a$ and V is weakly x-natural and a is weakly natural.*
- *A value V is weakly x-natural if $V = x$ or V is a λ-abstraction.*

We let $\mathbb{L}$ range over weakly natural lists.

System $\lambda\tilde{\mathbf{n}}\mathbf{n}$ and its subsystem $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$. An expression of $\lambda\mathbf{J}$ is *weakly natural* if all continuations occurring in it are weakly natural lists. Alternatively, weakly natural expressions are generated by the following grammar:

(weakly natural proof terms)	t, u, v	$::=$	$x \mid \lambda x.t \mid ta$
(weakly natural J-arguments)	a	$::=$	$(u, \mathbb{L})$
(weakly natural continuations)	$\mathbb{L}$	$::=$	$(x)v$, with v weakly x-natural

Notice that a weakly natural continuation is a weakly natural list, but not conversely: in a weakly natural continuation $(x)v$, v is not only weakly x-natural, but also weakly natural.

This syntax is closed for substitution: Given weakly normal u, one proves by simultaneous induction on weakly natural t, a, and $\mathbb{L}$ that: $\mathbf{s}(u,x,t)$, $\mathbf{s}(u,x,a)$ and $\mathbf{s}(u,x,\mathbb{L})$ are weakly natural; in addition, $\mathbf{s}(u,x,t)$ is weakly y-natural (for $y \neq x$), if t is so. Notice the subtle point: $\mathbf{s}(u,x,V)$ is a weakly y-natural *value*, regardless of u, if V is one such value (for $y \neq x$). One also sees by simultaneous induction on weakly natural $\mathbb{L}$ and a' and weakly x-natural v that the continuation $\mathbb{L}@a$ and the argument $a'@a$ are weakly natural and that the term $v@a$ is weakly x-natural, when a is weakly natural and $x \notin a$. Therefore, this syntax is closed for the reduction rules β and π and constitutes a subsystem of $\lambda\mathbf{J}$ called $\lambda\tilde{\mathbf{n}}\mathbf{n}$.

The π-normal forms of $\lambda\tilde{\mathbf{n}}\mathbf{n}$ are characterized by requiring t in ta to be a value V. As for $\lambda\mathbf{n}$, such normal forms are still closed under β, provided β is redefined as employing, not ordinary substitution $\mathbf{s}(u,x,E)$, but instead a special substitution $\mathbb{S}(u,x,E)$, defined exactly as ordinary substitution, except that $\mathbb{S}(u,x,Va) = \mathbb{S}(u,x,V)@\mathbb{S}(u,x,a)$. The π-normal forms of $\lambda\tilde{\mathbf{n}}\mathbf{n}$, equipped with β constitute the subsystem $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$ of $\lambda\tilde{\mathbf{n}}\mathbf{n}$.

3.2. A new isomorphism

We define the system $\lambda\tilde{\mathbf{n}}$ suggested above. Its terms are generated by

$$t,u ::= V \,|\, ta \qquad V ::= x \,|\, \lambda x.t \qquad a ::= (u,(x)V')$$

where, in the last production, V' is a weakly x-natural value. This restriction ensures this syntax to be closed for ordinary substitution. For this reason, we can define rule β in $\lambda\tilde{\mathbf{n}}$ exactly as in $\lambda\mathbf{J}$, and therefore this syntax forms a subset of the $\lambda\mathbf{J}$-terms closed for β-reduction. However, as with the λ-calculus, this syntax is not closed for π-reduction.

The encoding $t^\circ \in \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$ of a $\lambda\tilde{\mathbf{n}}$-term t is based on this idea: given weakly natural arguments $r_i = (u_i,(x)V_i)$, $Vr_1\cdots r_m$ is mapped to $V(r_1,\cdots,r_m)$, where $(r_1,\cdots,r_m) = r_1@\cdots@r_m$. The definition of t° is done simultaneously with $(t,R)^\circ$, for t a $\lambda\tilde{\mathbf{n}}$-term and R a weakly natural J-argument, as follows:

- $x^\circ = x$ and $(\lambda x.t)^\circ = \lambda x.t^\circ$ and $(t(u,(x)V))^\circ = (t,(u^\circ,(x)V^\circ))^\circ$
- $(V,R)^\circ = V^\circ R$ and $(t(u,(x)V),R)^\circ = (t,(u^\circ,(x)V^\circ R))^\circ$.

For the inverse map, given a $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$-term t, $\Phi t \in \lambda\tilde{\mathbf{n}}$ is defined simultaneously with $\Phi(t,R)$, for t a $\lambda\tilde{\mathbf{n}}$-term and R a weakly natural J-argument, as follows:

- $\Phi x = x$ and $\Phi(\lambda x.t) = \lambda x.\Phi t$ and $\Phi(VR) = \Phi(\Phi V, R)$
- $\Phi(t,(u,(x)V)) = t(\Phi u,(x)\Phi V)$ and $\Phi(t,(u,(x)VR)) = \Phi(t(\Phi u,(x)\Phi V),R)$.

Theorem 2. $\lambda\tilde{\mathbf{n}} \cong \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$. *(See Appendix A.1 for details of the proof.)*

Exactly as suggested after Theorem 1, one proves that t° is the π-normal form of t.

3.3. Reduction to (weak) natural form

As in [13], reduction to natural form can be obtained naively through a map $\gamma : \lambda\mathbf{J} \to \lambda\mathbf{n}$. The relaxed notion of weak normality suggests a split of γ as $\gamma_1 : \lambda\mathbf{J} \to \lambda\tilde{\mathbf{n}}\mathbf{n}$ followed by $\gamma_2 : \lambda\tilde{\mathbf{n}}\mathbf{n} \to \lambda\mathbf{n}$. The latter should be the restriction of γ to $\lambda\tilde{\mathbf{n}}\mathbf{n}$. We suggest here a definition of γ_1, based on some considerations about weak naturality that adapt similar ones developed for naturality in [13]. With this, we will refine the left diagram of Fig. 3, to obtain the right diagram, already integrating the isomorphism $\lambda\tilde{\mathbf{n}} \cong \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$.

Let us define (as in [10, 13]): $mla(x,v)$ if $v = V(u,c)$, V is x-natural, and $x \notin u,c$. V is x-natural if $V = x$. We also say $mla((x)v)$ when $mla(x,v)$. [5] We also define the append of two continuations $c'@c$ simultaneously with $v@c$ as follows: If $v = V(u,c')$ and $x \notin u,c'$, and V is x-natural then $((x)v)@c = (x)V(u,c'@c)$; else $((x)v)@c = (x)(v@c)$; and $v@(x)t = \mathbb{S}(v,x,t)$.[6]

Every continuation c can be decomposed as $\delta(c) = \langle L,c'\rangle$ with L a natural list, $\neg mla(c')$ and $c = L@c'$. The function δ is given by: $\delta((x)v) =$ if $\neg mla(x,v)$ then $\langle \mathbf{nil},(x)v\rangle$; else let $v = x(u,c)$, let $\delta(c) = \langle L,c'\rangle$ in $\langle (x)x(u,L),c'\rangle$. Two characterizations of natural lists follow immediately: if $\delta(c) = \langle L,c'\rangle$ then c is a natural list iff $c' = \mathbf{nil}$; and c is a natural list iff there is L such that $c = L@\mathbf{nil}$.

Profiting from δ, it is natural to propose a map $\gamma : \lambda\mathbf{J} \to \lambda\mathbf{n}$ whose translation of applications is $\gamma(t(u,c)) = \mathbf{s}(\gamma t(\gamma u,\gamma L),x,\gamma v)$, where $\delta(c) = \langle L,(x)v\rangle$.[7]

[5] Read $mla(x,v)$ as "v is an application, and is also a linear left-introduction whose main formula is an implication which is the type of x".

[6] Note that these simultaneously defined append operations $c'@c$ and $v@(x)t$ differ from the respective ones of [13] in that the latter sets $v@(x)t = \mathbf{s}(v,x,t)$, employing ordinary substitution, instead of the special one. The change is necessary in order to match the previous change in the definition of $c@a$ an keep a Lemma from [13], which says $c@a = c@(z)za$, for $z \notin a$.

[7] We could consider a variant which employs the special substitution $\mathbb{S}$ rather than ordinary substitution $\mathbf{s}$, as is done in the definition of map γ in [13]. Such variant would be a different function from the function γ being proposed here.

We can repeat these developments for weak naturality. We define: $wmla(x, v)$ if $v = V(u, c)$, V is weakly x-natural, and $x \notin u, c$. We also say $wmla((x)v)$ when $wmla(x, v)$. We define the *weak append* of two continuations $c'@_w c$ simultaneously with $v@_w c$ as follows: If $v = V(u, c')$ and $x \notin u, c'$, and V is weakly x-natural then $((x)v)@_w c = (x)V(u, c'@_w c)$; else $((x)v)@_w c = (x)(v@_w c)$; and $v@_w(x)t = \mathbb{S}(v, x, t)$, as before.

Every continuation c can be decomposed as $\delta_w(c) = \langle \mathbb{L}, c' \rangle$ with $\mathbb{L}$ a weakly natural list, $\neg wmla(c')$ and $c = \mathbb{L}@_w c'$. The function δ_w is given by: $\delta_w((x)v) =$ if $\neg wmla(x, v)$ then (if $v = V$ and V is weakly x-natural, then $\langle (x)v, \mathbf{nil} \rangle$ else $\langle \mathbf{nil}, (x)v \rangle$); else let $v = V(u, c)$, let $\delta_w(c) = \langle \mathbb{L}, c' \rangle$ in $\langle (x)V(u, \mathbb{L}), c' \rangle$. Two characterizations of weakly natural lists follow immediately: if $\delta_w(c) = \langle \mathbb{L}, c' \rangle$ then c is a weakly natural list iff $c' = \mathbf{nil}$; and c is a weakly natural list iff there is $\mathbb{L}$ such that $c = \mathbb{L}@\mathbf{nil}$.

Using δ_w, we can propose at last a map $\gamma_1 : \lambda\mathbf{J} \to \lambda\tilde{\mathbf{nn}}$ returning a weakly natural term $\gamma_1(t)$, for each $t \in \lambda\mathbf{J}$. Its translation of applications is $\gamma_1(t(u, c)) = \mathbf{s}(\gamma_1 t(\gamma_1 u, \gamma_1 \mathbb{L}), x, \gamma_1 v)$, where $\delta_w(c) = \langle \mathbb{L}, (x)v \rangle$.

4. The multiary extension

4.1. Recapitulation

Motivation. As said, the $\beta\pi$-normal forms of $\lambda\mathbf{J}$ represent the cut-free sequent calculus proofs. The "multiary" variant of them was essentially introduced in [14], to make a termination argument about permutative conversions [3]. However, the "multiary" variant of $\lambda\mathbf{J}$-terms, when equipped with appropriate reduction rules extending those of $\lambda\mathbf{J}$, have another advantage: they constitute a system, named $\lambda\mathbf{Jm}$ [12], which captures the sequent calculus more faithfully than $\lambda\mathbf{J}$.

To make this evident, we made a presentation of $\lambda\mathbf{J}$ which is a bit unusual, in its separation of three syntactic classes and forms of sequents (recall (1)) – but the purpose was to show that the inference process in the left-hand-side of sequents, which is the hallmark of the sequent calculus, is degenerate in $\lambda\mathbf{J}$. Referring to the inference rules in Fig. 1, we can select a formula B in the left-hand-side, producing $(x)v$; this formula has to be used as active formula in the right premiss of a $Left$ inference, producing $(u, (x)v)$; but then the left-introduced formula $A \supset B$ is immediately cut with a proof t of the same formula, and the process returns to the situation where no formula is selected in the left-hand side. If we want to chain two left introductions, we have to make an artificial roundabout through a cut against an axiom, like this: $(u', (y)y(u, (x)v))$. Multiarity is a facility allowing the accumulation of those u's in a non-empty list U, allowing the inference process to stay in the

left-hand-side.

System $\lambda\mathbf{Jm}$. Expressions E of $\lambda\mathbf{Jm}$ are generated by the grammar

$$\begin{array}{rrcl} \text{(proof terms)} & t,u,v & ::= & x \mid \lambda x.t \mid ta \\ \text{(Jm-arguments)} & a & ::= & (u,l,c) \\ \text{(lists)} & l & ::= & u::l \mid [] \\ \text{(continuations)} & c & ::= & (x)v \end{array}$$

Relatively to $\lambda\mathbf{J}$, the difference is in the Jm-arguments, with its non-empty list u, l of arguments. Quite often we will write such lists as (u, l), and U will range over them; Jm-arguments can thus be written as $((u, l), c)$ or (U, c). For lists l, we keep using the usual notation $[u_1, \cdots, u_m]$ to denote $(u_1 :: \cdots (u_m :: []))$.[8] For non-empty lists U we will allow ourselves, in some informal discussions, the notation $U_1 \underline{@} U_2$, defined by

$$(u_1, l_1)\underline{@}(u_2, l_2) = (u_1, l_1@(u_2 :: l_2)) \ .$$

If we erase construction $u :: l$, we can dispense with the class of lists, with Jm-arguments reduced to the *unary* form $(u, [], c)$. The corresponding unary terms (the $\lambda\mathbf{Jm}$-terms where every application is unary) is nothing else but a rephrasing of $\lambda\mathbf{J}$-terms inside $\lambda\mathbf{Jm}$, the trivial encoding $(\_)^\blacklozenge : \lambda\mathbf{J} \to \lambda\mathbf{Jm}$ determined by $(u, c)^\blacklozenge = (u^\blacklozenge, [], c^\blacklozenge)$. We will treat this encoding as an identity, by not distingushing between (u, c) and $(u, [], c)$. In this way, $\lambda\mathbf{J}$-terms form a proper subset of $\lambda\mathbf{Jm}$. This practice is well justified at the levels of typing and reduction, as we now see.

The typing system of $\lambda\mathbf{Jm}$ requires a further form of sequents, $\Gamma; B \vdash l : C$, corresponding to the new syntactic class of lists l. The new typing rules, relative to those already in Fig. 1, are shown in Fig. 4. If a sequent of the mentioned form is derived in the system, then $B = \vec{B_i} \supset C$ (this notation abbreviates $B_1 \supset \cdots \supset B_m \supset C$, including the case $m = 0$). So, the $Leftm$ inference rule in that figure introduces $A \supset C$ only if $l = []$ and $B = C$ – the base case giving the $Left$ rule of $\lambda\mathbf{J}$.

The reduction rules of $\lambda\mathbf{Jm}$ are recalled in Fig. 5. The novelties w. r. t. $\lambda\mathbf{J}$ are the split of β into two cases, and the new rule μ, both novelties signaling the presence of lists of arguments l in applications ta. A β-redex has the form $(\lambda x.t)(U, (y)v)$. If U is a unary list $(u, [])$, then one can apply rule β_1, which reduces as rule β of $\lambda\mathbf{J}$. If $U = (u, u' :: l)$, then one applies rule β_2, which calls the function with the first argument u, and rearranges the Jm-argument.

[8] On one occasion below (Fig. 10) we will write the non-empty list $U = (u_0, l)$ as $[u_0, u_1, \cdots, u_n]$.

Figure 4: Typing rules for $\lambda\mathbf{Jm}$

$$\frac{\Gamma\vdash u:A \quad \Gamma;B\vdash l:C \quad \Gamma|C\vdash c:D}{\Gamma;A\supset B\vdash(u,l,c):D}\ \mathit{Leftm}$$

$$\frac{}{\Gamma;C\vdash[]:C}\ \mathit{Ax} \qquad \frac{\Gamma\vdash u:A \quad \Gamma;B\vdash l:C}{\Gamma;A\supset B\vdash u::l:C}\ \mathit{Lft}$$

Figure 5: Reduction rules of $\lambda\mathbf{Jm}$

$$\begin{array}{rrcl}
(\beta_1) & (\lambda x.t)(u,[],(y)v) & \to & \mathbf{s}(\mathbf{s}(u,x,t),y,v)\\
(\beta_2) & (\lambda x.t)(u,u'::l,c) & \to & \mathbf{s}(u,x,t)(u',l,c)\\
(\pi) & (ta)a' & \to & t(a@a')\\
(\mu) & (u,l,(x)x(u',l',c')) & \to & (u,l@(u'::l'),c'),\ \text{ if } x\notin u',l',c'
\end{array}$$

Rule μ (whose idea goes back to [14]) eliminates the roundabout cut $x(u',l',c)$ - recall the discussion above in the second paragraph of the motivation; the rule can also be written in a more compact form, as $(U,(x)x(U',c')) \to (U@U',c')$, with $x\notin U',c'$.

The reduction rules employ substitution, written $\mathbf{s}(u,x,t)$, and the append operation, introduced before for $\lambda\mathbf{J}$, but in its multiary version, obtained by this single change: $(U,c)@a = (U,c@a)$. As said for $\lambda\mathbf{J}$, this append operation turns the corresponding version of rule π more "eager" than any employed before: if the redex is $(ta')a$, a is "pushed" inside a' as much as possible.

Subsystem $\lambda\overline{\mathbf{n}}\mathbf{m}$. The unary terms are closed for substitution and append, hence they are closed for β_1 and π, constituting a copy of $\lambda\mathbf{J}$ inside $\lambda\mathbf{Jm}$. However, they are not closed for μ – for this reason they do not form a subsystem of $\lambda\mathbf{Jm}$. But, if rule μ is the cause for this failure, it is also a remedy.

As known since [10], μ-reduction is terminating and confluent. So we can define a calculus on μ-normal forms by equipping them with $\to_R$, for $R\in\{\beta,\pi\}$, defined as $\to_R$ of $\lambda\mathbf{Jm}$ restricted to μ-normal forms, followed by reduction to μ-normal form of the contractum, if needed, to return to the restricted syntax. We denote by $\lambda\overline{\mathbf{n}}\mathbf{m}$ the resulting system.

Another basic isomorphism. Now, the map μ, defined on $\lambda\mathbf{Jm}$, that calculates the unique μ-normal form of a term, when restricted to unary

Figure 6: Reduction rules of $\lambda\mathbf{m}$

$$\begin{array}{rrcl}
(\beta_1) & (\lambda x.t)(u,[]) & \to & \mathbf{s}(u,x,t) \\
(\beta_2) & (\lambda x.t)(u,u'::l) & \to & \mathbf{s}(u,x,t)(u',l) \\
(\zeta) & (t(u,l))(u',l') & \to & t(u,l@(u'::l'))
\end{array}$$

terms (that is, to $\lambda\mathbf{J}$-terms), becomes a bijection between the latter and the μ-normal forms, indeed an isomorphism between R-reduction in $\lambda\mathbf{J}$ and R-reduction in $\lambda\overline{\mathbf{n}}\mathbf{m}$ as defined just now [10]. We consider $\lambda\overline{\mathbf{n}}\mathbf{m} \cong \lambda\mathbf{J}$ as another basic isomorphism.

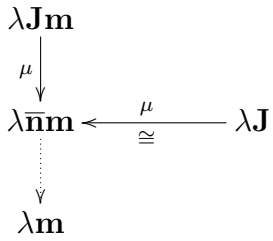
Subsystem $\lambda\mathbf{m}$. In the same way as $\lambda\mathbf{J}$ admits the subset of trivial terms (those with trivial continuations $(x)x$) closed under β, so does $\lambda\mathbf{Jm}$ has the subset of multiary trivial terms (again, those with trivial continuation) closed under β_1. Closure under β_2 is no problem either. As to the other two rules: such trivial terms have no μ-redexes; but π can go beyond trivial continuations, because $((x)x)@a = (x)xa$ is not trivial. However, $(x)xa$, with $x \notin a$, signals a μ-redex (and hence, non-triviality), so we are led to consider $\to_\pi$ followed by reduction to μ-normal form.

This leads to the definition of a subsystem $\lambda\mathbf{m}$, whose terms are those with m-arguments $(u,l,(x)x)$, with a trivial continuation, written simply (u,l) - so an application has the form tU.[9] These terms are equipped with β_1, β_2, and the mentioned combination of π and reduction to μ-normal forms, named ζ. These rules, written directly for this restricted syntax, are as shown in Fig. 6. Notice rule ζ can be written in the more compact form $tUU' \to t(U\underline{@}U')$. Logically, system $\lambda\mathbf{m}$ is a focused sequent calculus, the core of calculus LJT [7]. Computationally, $\lambda\mathbf{m}$ is the multiary λ-calculus, where application $t(u,l)$ handles non-empty lists of arguments.

As clear from the above description, the subsystem $\lambda\mathbf{m}$ is actually a

[9] Writting m-arguments as U can cause an ambiguity: two such arguments $U_i = (u_i, l_i, (x_i)x_i)$, $i = 1,2$, may be appended as arguments, resulting in $U_1@U_2 = (U_1,(x_1)x_1(U_2,(x_2)x_2)) = (U_1,U_2)$, or may be appended as non-empty lists, resulting in $U_1\underline{@}U_2 = (u_1, l_2@(u_2::l_2),(x_2)x_2)$. This explains the need for the new symbol $\underline{@}$.

subsystem of $\lambda\overline{\mathbf{n}}\mathbf{m}$, so we obtain the following picture:



This picture has to be integrated with Fig. 3, and the result will be as shown in Fig. 7, containing a concrete map $\lambda\overline{\mathbf{n}}\mathbf{m} \to \lambda\mathbf{m}$. The figure shows the basic isomorphism $\lambda \cong \vec{\lambda}\mathbf{n}$ (resp. $\lambda\mathbf{J} \cong \lambda\overline{\mathbf{n}}\mathbf{m}$) crossing the frontier that separates λ (resp. $\lambda\mathbf{J}$) from the wider world of $\lambda\mathbf{J}$ (resp. $\lambda\mathbf{Jm}$).

Some structural consequences via $\vec{\lambda}\mathbf{m}$. Now, we extract consequences of the basic isomorphism aforementioned, in order to progress towards Fig. 7. Given $\lambda\mathbf{J} \cong \lambda\overline{\mathbf{n}}\mathbf{m}$ and the fact that $\lambda\mathbf{n}$ is a subsystem of $\lambda\mathbf{J}$, there is a subsystem of $\lambda\overline{\mathbf{n}}\mathbf{m}$ isomorphic to $\lambda\mathbf{n}$. It turns out to be $\lambda\mathbf{m}$, with β (resp. π) reduction in $\lambda\mathbf{m}$ corresponding to β (resp. ζ) reduction in $\lambda\mathbf{n}$. At the level $\lambda\mathbf{n} \cong \lambda\mathbf{m}$, map μ just translates between applications $t(u_1, u_2, \cdots, u_m)$ and $t(u_1, [u_2, \cdots, u_m])$ (with $m \geq 1$). Next, in the same way there is a subsystem $\vec{\lambda}\mathbf{n}$ of $\lambda\mathbf{n}$ based on π-normal forms, there is a subsystem $\vec{\lambda}\mathbf{m}$ of $\lambda\mathbf{m}$ based on ζ-normal forms, such that $\vec{\lambda}\mathbf{n} \cong \vec{\lambda}\mathbf{m}$. At this level, map μ just translates between applications $V(u_1, u_2, \cdots, u_m)$ and $V(u_1, [u_2, \cdots, u_m])$.

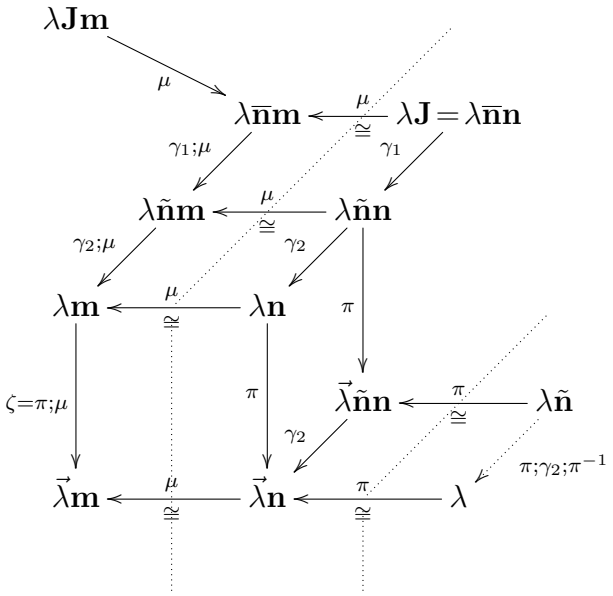
In the end, by composition of $\lambda \cong \vec{\lambda}\mathbf{n} \cong \vec{\lambda}\mathbf{m}$, we obtain the isomorphism $\lambda \cong \vec{\lambda}\mathbf{m}$, which essentialy goes back to [8]. Logically, it corresponds to a translation of natural deduction into sequent calculus, ultimately linked to the one proposed by Prawitz [16], rather than the one proposed by Gentzen [17]. Computationally, $\vec{\lambda}\mathbf{m}$ gives a formal vector notation for the λ-calculus – as opposed to the informal vector notation $Vu_1 \cdots u_n$, which is a meta-language device. Since $\lambda \cong \vec{\lambda}\mathbf{m}$ was known, the first basic isomorphism we stated above ($\lambda \cong \vec{\lambda}\mathbf{n}$) was expected, and only its extension $\lambda\tilde{\mathbf{n}} \cong \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$ deserved a proof.

4.2. Multiary naturality

Even if naturality was explained above in the context of $\lambda\mathbf{J}$, its original introduction in [13] was done directly for $\lambda\mathbf{Jm}$, as follows.

Definition 3 (Multiary natural lists). Multiary natural *arguments and lists and* multiary x-natural *terms are defined simultaneously as follows:*

Figure 7: The two frontiers



- *An argument $a = (u, l, c)$ is multiary natural if c is a multiary natural list.*
- *A continuation $c = (x)v$ is a multiary natural list if v is multiary x-natural.*
- *A term v is multiary x-natural if $v = x$ or $v = xa$ and $x \notin a$ and a is multiary natural.*

We may abbreviate "multiary natural" as m-natural. As in $\lambda\mathbf{J}$, we let L range over m-natural lists. Every m-natural list L is either $\mathbf{nil} := (x)x$ or $(u{+}l{+}L) := (x)x(u, l, L)$, with $x \notin u, l, L$. Given $U = (u, l)$, we may also write $(U{+}L)$ standing for $(u{+}l{+}L)$.

System $\lambda\mathbf{nm}$ and its subsystem $\vec{\lambda}\mathbf{nm}$. An expression of $\lambda\mathbf{Jm}$ is *multiary natural* if all continuations occurring in it are multiary natural lists. Alternatively, multiary natural expressions are generated by the same grammar as given in Section 2.2 for natural expressions, this time generating multiary natural proof terms t, u, v, Jm-arguments a and continuations L. The only difference is that multiary natural Jm-arguments have the form (U, L), and so we simultaneously also define multiary natural U's and lists l in the obvi-

Figure 8: The reduction rules of system $\lambda\mathbf{nm}$

$$\begin{array}{rrcl}
(\beta_{11}) & (\lambda x.t)(u,[],\mathbf{nil}) & \to & \mathbf{s}(u,x,t) \\
(\beta_{12}) & (\lambda x.t)(u,[],(u'{+}l{+}L)) & \to & \mathbf{s}(u,x,t)(u',l,L) \\
(\beta_2) & (\lambda x.t)(u,u'::l,L) & \to & \mathbf{s}(u,x,t)(u',l,L) \\
(\pi) & t(u,l,L)(u',l',L') & \to & t(u,l,L@(u'{+}l'{+}L')) \\
(\mu_1) & (u,l,(u'{+}l'{+}L)) & \to & (u,l@(u'::l'),L) \\
(\mu_2) & (u{+}l{+}(u'{+}l'{+}L)) & \to & (u{+}l@(u'::l'){+}L)
\end{array}$$

ous way: $U = (u, l)$ is multiary natural if u and l are multiary natural; [] is multiary natural; and $u::l$ is multiary natural if u and l are multiary natural.

In [13] one finds a very detailed discussion of the fact that this syntax is closed for the reduction rules of $\lambda\mathbf{Jm}$, constituting the subsystem $\lambda\mathbf{nm}$. A presentation of its reduction rules, also taken from [13], is in Fig. 8. Notice $\lambda\mathbf{nm}$ is not among the systems in Fig. 7.

Reduction rule β_{12} is not strictly needed because each $t_1 \to_{\beta_{12}} t_2$ implies $t_1 \to_{\mu_1} t' \to_{\beta_2} t_2$, for some t'. Reduction rules define *root* reduction in the syntactic classes they are defined. Rules μ_1 (resp. μ_2) is defined on arguments (resp. continuations). We define *root μ_1-reduction on terms* as the closure of rule μ_1 under: $a \to a'$ implies $ta \to ta'$; and *root μ_2-reduction on terms* as the closure of rule μ_2 under the previous rule and this one: $L \to L'$ implies $(U{+}L) \to (U{+}L')$. The compatible closure of root μ_1-reduction on terms is the same relation, on terms, as $\to_{\mu_1}$. Similarly for μ_2.

The π-normal forms of $\lambda\mathbf{nm}$ are characterized by requiring t in ta to be a value V. Such normal forms are still closed under μ_1 and μ_2. A β-reduction step of the kind found in $\lambda\mathbf{nm}$ may not preserve π-normal form, so π-normalization may be required afterwards. Alternatively, β-reduction in π-normal forms may be defined as being generated by these two rules

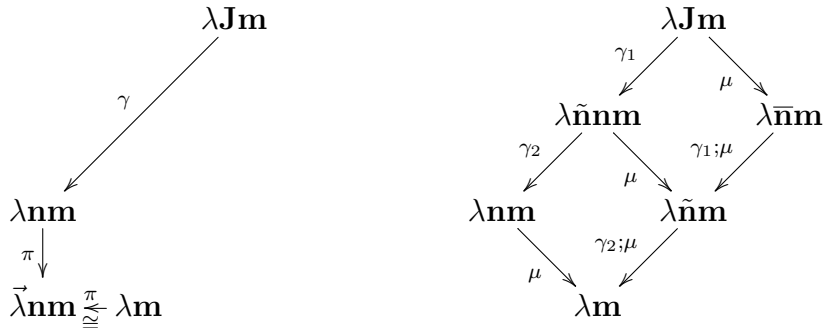
$$\begin{array}{rrcl}
(\beta_{11}) & (\lambda x.t)(u,[],\mathbf{nil}) & \to & \mathbb{S}(u,x,t) \\
(\beta_2) & (\lambda x.t)(u,u'::l,L) & \to & \mathbb{S}(u,x,t)@(u',l,L)
\end{array}$$

which employ the special substitution $\mathbb{S}(u, x, E)$ we met before, defined exactly as ordinary substitution, except that $\mathbb{S}(u, x, Va) = \mathbb{S}(u, x, V)@\mathbb{S}(u, x, a)$. The π-normal forms of $\lambda\mathbf{nm}$, equipped with β_{11}, β_2, μ_1 and μ_2 constitute the subsystem $\vec{\lambda}\mathbf{nm}$ of $\lambda\mathbf{nm}$. Notice we gave up β_{12}.

4.3. Another isomorphism

What happens next is as if we restarted with protagonists $\lambda\mathbf{m}$ and $\lambda\mathbf{Jm}$ what we have done before with λ and $\lambda\mathbf{J}$ (recall Fig. 7). The farthest we will

Figure 9: Left: another isomorphism. Right: reduction to (weak) natural form vs μ



go in this paper is to build the multiary version of the left diagram in Fig. 3, shown in the left diagram of Fig. 9.

As before, we introduce a map $\Phi : \vec{\lambda}\mathbf{nm} \to \lambda\mathbf{m}$ with inverse $(\_)^\circ$, as a multiary version of the maps between $\vec{\lambda}\mathbf{n}$ and λ. Hence, the idea behind $(\_)^\circ$ is to encode $VU_1 \cdots U_m$ as $V(U_1, \cdots, U_m)$, where $(U_1, \cdots, U_m) := a_1@\cdots@a_m$, with $a_i = (U_i, (x)x)$. Again map $(\_)^\circ$ is actually calculating π-normal forms.

We define by simultaneous recursion on $t, U, l \in \lambda\mathbf{m}$: the term $t^\circ \in \vec{\lambda}\mathbf{nm}$, as well as the term $(t, U, L)^\circ \in \vec{\lambda}\mathbf{nm}$, given $U, L \in \vec{\lambda}\mathbf{nm}$; the non-empty list $U^\circ \in \vec{\lambda}\mathbf{nm}$; and the list $l^\circ \in \vec{\lambda}\mathbf{nm}$ as follows:

- $x^\circ = x$ and $(\lambda x.t)^\circ = \lambda x.t^\circ$ and $(tU)^\circ = (t, U^\circ, \mathbf{nil})^\circ$
- $(V, U, L)^\circ = V^\circ(U, L)$ and $(tU, U', L)^\circ = (t, U^\circ, (U'{+}L))^\circ$
- $(u, l)^\circ = (u^\circ, l^\circ)$ and $[]^\circ = []$ and $(u::l)^\circ = u^\circ::l^\circ$.

We define by simultaneous recursion on $t, L, U, l \in \vec{\lambda}\mathbf{nm}$: the term $\Phi t \in \lambda\mathbf{m}$; the term $\Phi(t, U, L) \in \lambda\mathbf{m}$, given $t \in \lambda\mathbf{m}$ and $U \in \vec{\lambda}\mathbf{nm}$; the argument $\Phi U \in \lambda\mathbf{m}$; and the list $\Phi l \in \lambda\mathbf{m}$ as follows:

- $\Phi x = x$ and $\Phi(\lambda x.t) = \lambda x.\Phi t$ and $\Phi(V(U, L)) = \Phi(\Phi V, U, L)$
- $\Phi(t, U, \mathbf{nil}) = t(\Phi U)$ and $\Phi(t, U, (U'{+}L)) = \Phi(t(\Phi U), U', L)$
- $\Phi(u, l) = (\Phi u, \Phi l)$ and $\Phi([]) = []$ and $\Phi(u::l) = \Phi u::\Phi l$.

Theorem 3. $\lambda\mathbf{m} \cong \vec{\lambda}\mathbf{nm}$. *(See Appendix A.2 for details of the proof.)*

Example - part 2. Let us go back to the example presented at the end of Section 2, and develop briefly its multiary version. We start with a $\lambda\mathbf{m}$-term $t_1 := t_0U_1U_2$, with $t_0 := \lambda x.t$ and $U_i = (u_i, l_i)$. Notice $\lambda\mathbf{m} \subset \lambda\mathbf{nm}$ - indeed t_1 is the m-natural term $t_0(U_1, (x_1)x_1)(U_2, (x_2)x_2)$. A single π-reduction step leads us beyond the set of $\lambda\mathbf{m}$-terms: $t_1 \to_\pi t_0(U_1, U_2) = t_0(u_1, l_1, (x_1)x_1(u_2, l_2, (x_2)x_2)) =: t_2$. This latter term is an application with continuation $c_1 := (x_1)x_1(u_2, l_2, (x_2)x_2)$, which is a natural list (notice $x_1 \notin u_2, l_2$), also written $(u_2{+}l_2 {+}\mathbf{nil})$. If t_1 has type B, then t_0 has type of the form $\vec{A_1} \supset \vec{A_2} \supset B$, for vectors $\vec{A_1} = A_{11}, \cdots, A_{1n_1}$ and $\vec{A_2} = A_{21}, \cdots, A_{2n_2}$. The arguments u_i have type A_{i1} and $\vec{|A_2} \supset B \vdash c_1 : B$. Finally, instead of analyzing more general forms of the continuation c_1 going beyond m-naturality, we remark a new possibility brought by multiarity, which is the μ-reduction $t_2 \to t_0(u_1, l_1@(u_2 :: l_2)) = t_0(U_1 \underline{@} U_2)$. The new possibility is felt even if the start term t_1 is a λ-term, hence with $l_i = []$. In that case, the path $t_1 \to t_2 \to t_3$ leads us from λ to $\lambda\mathbf{n}$ to $\lambda\mathbf{m}$. In the general case, with arbitrary l_i, the path leads from $\lambda\mathbf{m}$ to $\lambda\mathbf{nm}$ and then back to $\lambda\mathbf{m}$.

4.4. Weak multiary naturality

Let us briefly mention weak naturality in the setting of $\lambda\mathbf{Jm}$. In order to define the multiary version of the concepts introduced in Def. 2, we just change the first item there, and say an argument $a = (u, l, c)$ is weakly m-natural if c a weak m-natural list - no constraint imposed on the list l. We thus obtain the concept of weak m-natural lists, denoted $\mathbb{L}$ as in $\lambda\mathbf{J}$.

System $\lambda\tilde{\mathbf{n}}\mathbf{nm}$. The weakly m-natural expressions of $\lambda\mathbf{Jm}$ are those where every continuation is one such $\mathbb{L}$. Alternatively, they can be given by the grammar after Def. 2, provided arguments a have the form $(u, l, \mathbb{L})$ and lists l are generated by the usual two clauses $[]$ and $u :: l$. By considerations similar to those given before, the weakly m-natural expressions constitute the subsystem $\lambda\tilde{\mathbf{n}}\mathbf{nm}$ of $\lambda\mathbf{Jm}$. The contents of Subsection 3.3 is routinely adapted to the multiary setting, providing a map $\gamma : \lambda\mathbf{Jm} \to \lambda\mathbf{nm}$, that can be restricted to $\gamma_2 : \lambda\tilde{\mathbf{n}}\mathbf{nm} \to \lambda\mathbf{nm}$, and a map $\gamma_1 : \lambda\mathbf{Jm} \to \lambda\tilde{\mathbf{n}}\mathbf{nm}$.

Combination with μ-normality. Consider the right diagram of Fig. 9. Let us elucidate the form of arguments a in the lower range of systems, where only μ-normal forms are allowed (hence continuations $(x)xa$ with $x \notin a$ are forbidden). In $\lambda\mathbf{m}$, arguments have the form $U = (u, l, L')$, where L' is a natural list which does not destroy μ-normality - hence L' cannot in fact be a natural list except for the trivial continuation $(x)x$. Going upwards, this pattern repeats. In $\lambda\tilde{\mathbf{n}}\mathbf{m}$, arguments have the form $\mathbb{U} = (u, l, \mathbb{L}')$, where $\mathbb{L}'$ is a weakly natural list that is not a natural list different from the trivial

continuation. In $\lambda\overline{\mathbf{n}}\mathbf{m}$, arguments have the form $\mathcal{U} = (u, l, \mathcal{L}')$, where $\mathcal{L}$ is any continuation that is not a natural list different from the trivial continuation.

Consider again the right diagram of Fig. 9, and compose its two squares to obtain a square with four vertices $\lambda\mathbf{Jm}$, $\lambda\overline{\mathbf{n}}\mathbf{m}$, $\lambda\mathbf{nm}$ and $\lambda\mathbf{m}$. In [13], the same square, with edges labelled with γ (or $\gamma;\mu$) and μ was shown to be commutative. We leave similar commutation questions about the two composed squares for future work.

5. Discussion

In this section we discuss, with the help of Fig. 10,[10] what we have achieved in this paper. In the first subsection we focus on proof-theoretical aspects; in the second, on aspects relative to the λ-calculus.

5.1. Weakly general natural deduction

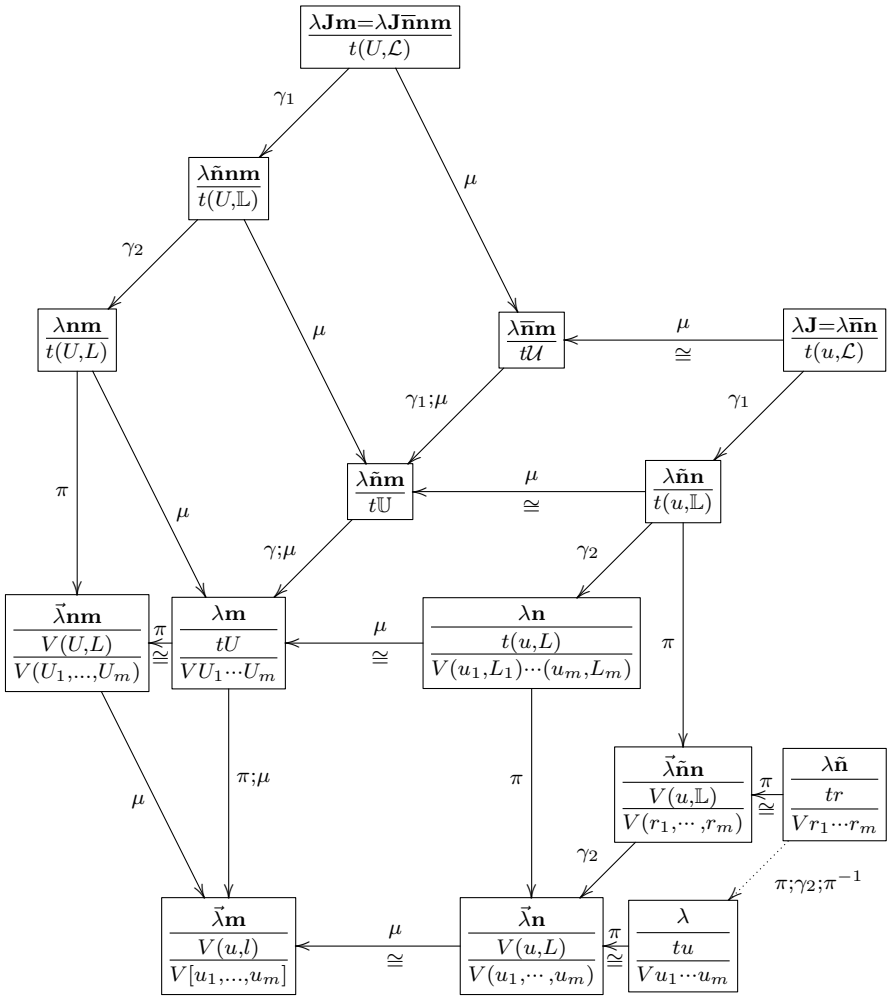
In the diagram of Fig. 10 there are 3 regions corresponding to 3 families of proof systems: (1) the region below $\lambda\tilde{\mathbf{n}}$, corresponding to natural deduction; (2) the region below $\lambda\mathbf{J}$, corresponding to natural deduction with generalized applications; (3) the region below $\lambda\mathbf{Jm}$, corresponding to sequent calculus. Here "below" means accessible by descending arrows. Region (1) has a copy inside region (2): each of the two systems in the former has a matching subsystem of $\lambda\mathbf{J}$ in the latter, with isomorphisms π (Theorems 1 and 2) mediating between a system in (1) and its match. Similarly, region (2) has a partial copy inside region (3), given by the isomorphism $\mu : \lambda\mathbf{J} \to \lambda\overline{\mathbf{n}}\mathbf{m}$ and its restrictions - we say "partial" because we did not develop the system of π-normal forms of $\lambda\tilde{\mathbf{n}}\mathbf{m}$ that would complete the image of the region (2). Isomorphisms π (resp. μ) cross the frontier between regions (1) and (2) (resp. (2) and (3)).

By accepting this organization, we find in region (1) a system of natural deduction other than the ordinary system corresponding to the λ-calculus - the system corresponding to $\lambda\tilde{\mathbf{n}}$, which we name *weakly general natural deduction*. In Section 3, this system is identified as a subclass of expressions of $\lambda\mathbf{J}$ closed under β-reduction of the latter system. Let us give a direct and self-contained definition - but still as a λ-calculus.

A term M, N, P is either a value, or an ordinary application MN, or a new form of application $M(N, (y, z)P)$. The latter is typed thus: if $\vdash M : A \supset B$ and $\vdash N : A$ and $y : B, z : C \vdash P : D$ then $\vdash M(N, (y, z)P) : C \supset D$

[10] For the reader of [13], the final diagram obtained here should be compared with the upper half of the final diagram obtained there. Notice we are not concerned with notions of cut-free proof terms in the present paper.

Figure 10: Alternative structuring of application



(here we omitted the common context Γ). Since the type of P is not $C \supset D$, the new form of application does not generate commuting conversions. But there are two β-rules, the ordinary and this one: $(\lambda x.M)(N,(y,z)P) \to \lambda z.\mathbf{s}(\mathbf{s}(N,x,M),y,P)$.

5.2. Computational interpretation

Let us recapitulate the computational interpretation of the (multiary) natural subsystems, that is, the systems in Fig. 10 below $\lambda\mathbf{nm}$ and $\lambda\mathbf{n}$. In $\lambda\mathbf{n}$, every application has the form $t(u,L)$, and every n-argument (u,L) can be written as $(u_1,\cdots,u_m) = u_1@\cdots@u_m$. So an n-argument is suitable to collect the arguments of an application $Vu_1\cdots u_m$ in the λ-calculus, and this is what the isomorphism $\lambda \cong \vec{\lambda}\mathbf{n}$ does. Now we move to the multiary version of this story. In $\lambda\mathbf{nm}$, every application has the form $t(U,L)$, and every nm-argument (U,L) can be written as $(U_1,\cdots,U_m) = U_1@\cdots@U_m$. So an nm-argument is suitable to collect the arguments of an application $VU_1\cdots U_m$ in the $\lambda\mathbf{m}$-calculus, and this is what the new isomorphism $\lambda\mathbf{m} \cong \vec{\lambda}\mathbf{nm}$ does. An n-argument (u,L) is a non-empty list of ordinary arguments, with u being the head and the natural list L being the tail of such non-empty list. An nm-argument (U,L) is a non-empty list of multiary arguments, with U being the head and the multiary natural list L being the tail of such non-empty list. In short, (multiary) natural lists L are lists of (multiary) arguments.

Now we want to move beyond the (multiary) natural subsystems. Why? Because the isomorphism $\lambda\tilde{\mathbf{n}} \cong \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$ repeats a similar story: in $\lambda\tilde{\mathbf{n}}\mathbf{n}$, application has the form $t(u,\mathbb{L})$, and the continuations $\mathbb{L}$ are lists of weakly natural arguments r. We conjecture the same happens in $\lambda\tilde{\mathbf{n}}\mathbf{nm}$, with every $\mathbb{L}$ there being a list of multiary weakly natural arguments R. So we now take the final step, and extend the interpretation to $\lambda\mathbf{J}$ and $\lambda\mathbf{Jm}$ respectively.

We will work out the extension of the interpretation from $\lambda\mathbf{nm}$ to $\lambda\mathbf{Jm}$ (for the extension from $\lambda\mathbf{n}$ to $\lambda\mathbf{J}$, replace U's by u's). Let us write multiary natural lists in $\lambda\mathbf{nm}$ as follows:

$$L ::= \mathbf{NIL} \,|\, \mathbf{CONS}a \qquad a ::= (U,L)$$

The operations $L@a$ and $a'@a$, when written with this syntax, read

$$\mathbf{NIL}@a = \mathbf{CONS}a \qquad (\mathbf{CONS}a')@a = \mathbf{CONS}(a'@a) \qquad (U,L)@a = (U,L@a)$$

This just looks like a pedantic way of writing lists of U's, with a strange separation of the pair (U,L) out of which **CONS** constructs a list. But all of this is just preparation for the general case, dealing with general continuations c.

A continuation c of $\lambda\mathbf{Jm}$ is either $(x)V$ or $(x)ta$, with $a = (U,c)$. We now see the two forms of c as generalizations of **NIL** and **CONS**, respectively.

For emphasis, we use $\mathcal{L}$ instead of c, and think of $\mathcal{L}$ as a *general list*:

$$\mathcal{L} ::= \mathbf{NIL}_{x.V} \mid \mathbf{CONS}_{x.t}a \qquad a ::= (U, \mathcal{L})$$

The operation $\mathcal{L}@a$, when written with this syntax, reads

$$(\mathbf{NIL}_{x.V})@a = \mathbf{CONS}_{x.V}a \quad (\mathbf{CONS}_{x.t}a')@a = \mathbf{CONS}_{x.t}(a'@a)$$

with $a'@a$ again given by $(U, \mathcal{L})@a = (U, \mathcal{L}@a)$. We know $\mathbf{NIL} = \mathbf{NIL}_{x.x}$ and $\mathbf{CONS} = \mathbf{CONS}_{x.x}a$ with $x \notin a$. The general indexes $x.V$ and $x.t$ added to **NIL** and **CONS** seem innocuous in these operations. In fact, their use (and their computational interpretation) is seen elsewhere, in reduction rule β_1, which (as seen before in $\lambda\mathbf{nm}$) must be separated into two:

$$\begin{array}{rcl} (\lambda x.t)(u, \mathbf{NIL}_{y.V}) & \to & V' \\ (\lambda x.t)(u, \mathbf{CONS}_{y.v}a) & \to & v'a' \end{array}$$

where $E' = \mathbf{s}(\mathbf{s}(u, x, t), y, E)$, for $E = V, v, a$.

Therefore $\mathbf{NIL}_{x.V}$ is not just a token to finish a list: it provides a value V, an instance of which will be the result of the computation. $\mathbf{CONS}_{y.v}a$ does not merely construct a list out of the pair a, it provides a term v, an instance of which will be applied to a similar instance of a – it is from this application that the computation proceeds. The application of $\lambda x.t$ to u just determines the instance of the parameter y that needs to be provided.

Example - part 3. Let us conclude the example at the end of Section 2. First we have the λ-term $t_1 = t_0u_1u_2$, with $t_0 = \lambda x.t$. Then $t_1 \to t_0(u_1, (x_1)v_1(u_2, (x_2)x_2)) = t_2$, with $v_1 = x_1$ and $x_1 \notin u_2$. If we let $a = (u_2, \mathbf{NIL})$, then $t_2 = (\lambda x.t)(u_1, \mathbf{CONS}a) \to_{\beta_1} t'a$, where $t' = \mathbf{s}(u_1, x, t)$. Here, the continuation $c_1 = \mathbf{CONS}a$ just consists of the list with the next argument u_2. In the reduction step, the symbol **CONS** is erased, so that the list becomes a and can act as the argument of the next application, whose function term t' is calculated in a separate process, as a substitution. However, if we allow v_1 to be arbitrary and x_1 to occur in v_1 and u_2, then the reduction step is richer. We have $t_2 = (\lambda x.t)(u_1, \mathbf{CONS}_{x_1.v_1}a) \to_{\beta_1} v_1'a'$, where $E' = \mathbf{s}(t', x_1, E)$, for $E = v_1, a$. The separate substitution provides the parameter t' with which we instantiate the parametrized application $(x_1)v_1a$. As the symbol **CONS** is erased, an instance a' (rather than a) becomes the argument of the next application, whose function term v_1' is an instance of the term attached to **CONS** (rather than the parameter t' itself).

6. Final remarks

This paper may be seen as an upgrade of our previous paper [13], and its main results are the various isomorphisms, the concept of weak naturality, and the final computational interpretation. In addition, the isomorphisms become visible when we combine (weak) naturality and normality w. r. t. the commutative conversion π, and the study of this combination is also a novelty. So, while in [13] we structured the space of calculi between the λ-calculus and $\lambda\mathbf{Jm}$ mainly by means of the combination of γ (reduction to permutation-free form) and μ, here the combination of these two with π produces a more detailed picture showing all the nuances and alternatives in structuring λ-terms, and containing, at last, true subsystems ($\vec{\lambda}\mathbf{n}$ and $\vec{\lambda}\mathbf{m}$) isomorphic to the λ-calculus.

There is an immediate and direct question that deserves attention. The isomorphism $\vec{\lambda}\mathbf{nm} \cong \lambda\mathbf{m}$ (Thm. 3) is a multiary version of the isomorphism $\vec{\lambda}\mathbf{n} \cong \lambda$ (Thm. 1). But the latter was firstly generalized as $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n} \cong \lambda\tilde{\mathbf{n}}$ (Thm. 2). Do these two extensions join as $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{nm} \cong \lambda\tilde{\mathbf{n}}\mathbf{m}$? Here $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{nm}$ is a system not yet in the picture, but whose place is clear.

In the older papers about normality in the cut-free setting [3, 14], such normal form corresponds to irreducibility in a well behaved rewriting system of permutative conversions. In this paper, as in [13], we deal with normality in the presence of cut, but reduction to such normal form is done naively by a map γ. This lacuna remains. At the same time, the present paper opens new lines of inquiry: Are there other interesting, weakened notions of normality? Is the upgrade from $\vec{\lambda}\mathbf{n} \cong \lambda$ to $\vec{\lambda}\mathbf{nm} \cong \lambda\mathbf{m}$ suggesting an iterative expansion of the space of calculi? Indeed, $\vec{\lambda}\mathbf{n} \cong \lambda$ can be prolonged to $\vec{\lambda}\mathbf{m} \cong \lambda$: can $\vec{\lambda}\mathbf{nm} \cong \lambda\mathbf{m}$ be prolonged to a system where the natural lists $(U_1, \cdots, U_m)$ of $\vec{\lambda}\mathbf{nm}$ are represented as $[U_1, \cdots, U_m]$? Such representation would perhaps live in a system $\lambda\mathbf{Jm}^2$, with lists of U's instead of lists of u's.

A truly systematic development of the space of calculi between the λ-calculus and $\lambda\mathbf{Jm}$ is still far beyond what we can reach with this paper, or perhaps with any paper. The final picture we obtained is only sketchy, in that its coherence (commutativity) is still waiting for a verification, some systems are still missing, and even parts of it still have tentative definitions, like what concerns γ – let alone the consideration of the variety of notions of normal form, seen in [11, 13] but out of scope in this paper. Maybe the systematic study of λ-calculi with generalized applications, even if we constrain ourselves to the call-by-name setting as we do here, can only be achieved with the help of machine assistance.

Acknowledgements. The authors were financed by Portuguese Funds

through FCT (Fundação para a Ciência e Tecnologia) within the projects UIDB/00013/2020, UIDP/00013/2020 and UIDB/50014/2020.

References

[1] F. Joachimski, R. Matthes, Standardization and confluence for a lambda calculus with generalized applications, in: Proc. of RTA'00, Vol. 1833 of Lecture Notes in Computer Science, Springer, 2000, pp. 141–155.

[2] J. von Plato, Natural deduction with general elimination rules, Annals of Mathematical Logic 40 (7) (2001) 541–567.

[3] R. Dyckhoff, L. Pinto, Permutability of proofs in intuitionistic sequent calculi, Theoretical Computer Science 212 (1999) 141–155.

[4] J. Zucker, The correspondence between cut-elimination and normalization, Annals of Mathematical Logic 7 (1974) 1–112.

[5] J.-Y. Girard, Y. Lafont, P. Taylor, Proofs and Types, Cambridge Univ. Press, 1989.

[6] G. Mints, Normal forms for sequent derivations, in: P. Odifreddi (Ed.), Kreiseliana, A. K. Peters, Wellesley, Massachusetts, 1996, pp. 469–492.

[7] H. Herbelin, A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure, in: Proc. of CSL'94, Vol. 933 of Lecture Notes in Computer Science, Springer, 1995, pp. 61–75.

[8] J. Espírito Santo, Revisiting the correspondence between cut-elimination and normalisation, in: Proceedings of ICALP'2000, Vol. 1853 of Lecture Notes in Computer Science, Springer-Verlag, 2000.

[9] J. Espírito Santo, L. Pinto, Permutative conversions in intuitionistic multiary sequent calculus with cuts, in: M. Hoffman (Ed.), Proc. of TLCA'03, Vol. 2701 of Lecture Notes in Computer Science, Springer, 2003, pp. 286–300.

[10] J. Espírito Santo, L. Pinto, Confluence and strong normalisation of the generalised multiary λ-calculus, in: Revised selected papers from TYPES'03, Vol. 3085 of Lecture Notes in Computer Science, Springer, 2004, pp. 286–300.

[11] J. Espírito Santo, M. J. Frade, L. Pinto, Structural proof theory as rewriting, in: Proc. of RTA'06, Vol. 4098 of Lecture Notes in Computer Science, Springer, 2006, pp. 197–211.

[12] J. Espírito Santo, L. Pinto, A calculus of multiary sequent terms, ACM Transactions on Computational Logic 12 (3) (2011) 22.

[13] J. Espírito Santo, M. J. Frade, L. Pinto, Permutability in proof terms for intuitionistic sequent calculus with cuts, in: Proc. of TYPES'16, Vol. 97 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 10:1–10:27.

[14] H. Schwichtenberg, Termination of permutative conversions in intuitionistic Gentzen calculi, Theoretical Computer Science 212(1-2) (1999) 247–260.

[15] J. Espírito Santo, The call-by-value lambda-calculus with generalized applications, in: M. Fernández, A. Muscholl (Eds.), 28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain, Vol. 152 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 35:1–35:12.

[16] D. Prawitz, Natural Deduction. A Proof-Theoretical Study, Almquist and Wiksell, 1965.

[17] G. Gentzen, Investigations into logical deduction, in: M. E. Szabo (Ed.), The collected papers of Gerhard Gentzen, North Holland, 1969.

Appendix A. Proofs of the new isomorphisms

Appendix A.1. Proof of Theorem 2 ($\lambda\tilde{\mathbf{n}} \cong \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$)

Proof. First, we need some bookkeeping about the maps.

1. $(t, R)^\circ = t^\circ @ R$.
2. $\mathbf{s}(u, x, t)^\circ = \mathbb{S}(u^\circ, x, t^\circ)$.
3. $\Phi(t@R) = \Phi(\Phi t, R)$.
4. $\Phi(\mathbb{S}(u, x, t)) = \mathbf{s}(\Phi u, x, \Phi t)$.

Item 1 is proved by induction on t and uses associativity of @. Item 2 is proved together with $(\mathbf{s}(u, x, t), \mathbb{S}(u^\circ, x, R))^\circ = \mathbb{S}(u^\circ, x, (t, R)^\circ)$, by induction on t (the proof of case $t = x$ requires item 1). Item 3 is proved by case analysis of t and uses $\Phi(\Phi(t, R'), R) = \Phi(t, R'@R)$, with the latter proved by induction on R'. Item 4 is proved together with $\mathbf{s}(\Phi u, x, \Phi(t, R)) = \Phi(\mathbf{s}(\Phi u, x, t), \mathbb{S}(u, x, R))$, by simultaneous induction on t and R.

Next, we prove the maps are each other's inverse. One proves, for all $t \in \lambda\tilde{\mathbf{n}}$, $\Phi(t^\circ) = t$ and $\Phi((t, R)^\circ) = \Phi(t, R)$, by induction on t; and proves,

for all $t, R \in \vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$, $(\Phi t)^\circ = t$ and $(\Phi(t', R))^\circ = (t', R)^\circ$, by simultaneous induction on t and R.

Finally we prove the isomorphism of reduction relations:

(a) $t_1 \to_\beta t_2$ in $\lambda\tilde{\mathbf{n}}$ iff $t_1^\circ \to_\beta t_2^\circ$ in $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$.

(b) $t_1 \to_\beta t_2$ in $\vec{\lambda}\tilde{\mathbf{n}}\mathbf{n}$ iff $\Phi t_1 \to_\beta \Phi t_2$ in $\lambda\tilde{\mathbf{n}}$.

The "only if" statements follow from the "if" statements and the fact that the maps are inverse. The "if" statements are proved by induction on $t_1 \to_\beta t_2$. The base case in (a) follows immediately by a calculation and using the bookkeeping fact 2 above. The base case in (b) is slightly more interesting, when the β-redex $(\lambda x.t)(u, \mathbb{L})$ is such that $\mathbb{L} = (y)VR$, with $y \notin R$. Then

$$\begin{array}{rcll}
\Phi((\lambda x.t)(u,(y)VR)) & = & \Phi((\lambda x.\Phi t)(\Phi u,(y)\Phi V),R) & \text{(by def. } \Phi) \\
& \to_\beta & \Phi(\mathbf{s}(\mathbf{s}(\Phi u,x,\Phi t),y,\Phi V),R) & (*) \\
& = & \Phi(\Phi(\mathbb{S}(\mathbb{S}(u,x,t),y,V)),R) & \text{(by fact 4 above, twice)} \\
& = & \Phi(\mathbb{S}(\mathbb{S}(u,x,t),y,V)@R) & \text{(by fact 3 above)} \\
& = & \Phi(\mathbb{S}(\mathbb{S}(u,x,t),y,VR)) & \text{(by def. of } \mathbb{S} \text{ and } y \notin R)
\end{array}$$

where in $(*)$ we use the fact: $t_1 \to_\beta t_2$ implies $\Phi(t_1, R) \to_\beta \Phi(t_2, R)$, easily proved by induction on R.

The inductive cases are routine. This ends the proof.

Appendix A.2. Proof of Theorem 3 ($\lambda\mathbf{m} \cong \vec{\lambda}\mathbf{nm}$)

PROOF. First, we need some bookkeeping about the maps.

1. $(t, U, L)^\circ = t^\circ@(U, L)$.
2. $\mathbf{s}(u, x, t)^\circ = \mathbb{S}(u^\circ, x, t^\circ)$.
3. $\Phi(t@(U, L)) = \Phi(\Phi t, U, L)$.
4. $\Phi(\mathbb{S}(u, x, t)) = \mathbf{s}(\Phi u, x, \Phi t)$.

Item 1 is proved by induction on t and uses associativity of @. Item 2 is proved together with similar statements for U and l in the place of t, by simultaneous induction on t, U, and l. The statement for t also includes the statement $(\mathbf{s}(u, x, t), \mathbb{S}(u^\circ, x, U), \mathbb{S}(u^\circ, x, L))^\circ = \mathbb{S}(u^\circ, x, (t, U, L)^\circ)$ (the proof of case $t = x$ requires item 1). Item 3 is proved by case analysis of t and uses $\Phi(\Phi(t, U', L'), U, L) = \Phi(t, U', L'@(U, L))$, with the latter proved by induction on L'. Item 4 is proved together with similar statements for U and l in the place of t, and also with $\mathbf{s}(\Phi u, x, \Phi(t, U, L)) = \Phi(\mathbf{s}(\Phi u, x, t), \mathbb{S}(u, x, U), \mathbb{S}(u, x, L))$, by simultaneous induction on t, U, l, and R.

Next, we prove the maps are each other's inverse. One proves, for all $E = t, U, l$ in $\lambda\mathbf{m}$, $\Phi(E^\circ) = E$, by simultaneous induction on t, U, and l. The statement for t is proved together with $\Phi(t, U, L)^\circ = \Phi(t, U, L)$. Next one proves, for all $E = t, U, L$ in $\vec{\lambda}\mathbf{nm}$, $(\Phi E)^\circ = E$ and, for all $L \in \vec{\lambda}\mathbf{nm}$, $(\Phi(t', U, L))^\circ = (t', U, L)^\circ$, by simultaneous induction on t, U, l, and L.

Finally we prove the isomorphism of reduction relations. We define a bijection between the reduction rules of $\lambda\mathbf{m}$ and $\vec{\lambda}\mathbf{nm}$: if $\rho = \beta_1$, (resp. β_2, ζ) then $\rho' = \beta_{11}$ (resp. β_2, μ).

(a) $t_1 \to_\rho t_2$ in $\lambda\mathbf{m}$ iff $t_1^\circ \to_{\rho'} t_2^\circ$ in $\vec{\lambda}\mathbf{nm}$.

(b) $t_1 \to_{\rho'} t_2$ in $\vec{\lambda}\mathbf{nm}$ iff $\Phi t_1 \to_\rho \Phi t_2$ in $\lambda\mathbf{m}$.

The "only if" statements follow from the "if" statements and the fact that the maps are inverse. The "if" statements are proved by induction on $t_1 \to_\rho t_2$ and $t_1 \to_{\rho'} t_2$, for each ρ, ρ'.

Let us analyze the base cases in (a). The case $\rho = \beta_1$ follows immediately by a calculation and using the bookkeeping fact 2 above. For the case $\rho = \beta_2$, we reason as follows:

$$\begin{array}{rcll}
(\lambda x.t)(u, v::l)^\circ & = & (\lambda x.t^\circ)(u^\circ, v^\circ::l^\circ, \mathbf{nil}) & \text{(by def. of } (_)^\circ) \\
& \to_{\beta_2} & \mathbb{S}(u^\circ, x, t^\circ)@((v^\circ, l^\circ), \mathbf{nil}) & \\
& = & \mathbf{s}(u, x, t)^\circ@((v^\circ, l^\circ), \mathbf{nil}) & \text{(by fact 2 above)} \\
& = & (\mathbf{s}(u, x, t), (v^\circ, l^\circ), \mathbf{nil})^\circ & \text{(by fact 1 above)} \\
& = & (\mathbf{s}(u, x, t)(v, l))^\circ & \text{(by def. of } (_)^\circ)
\end{array}$$

For the case $\rho = \zeta$, we reason as follows:

$$\begin{array}{rcll}
(t(u, l)(u', l'))^\circ & = & (t, (u^\circ, l^\circ), ((u'^\circ, l'^\circ)+\mathbf{nil}))^\circ & \text{(by def. of } (_)^\circ) \\
& \to_\mu & (t, (u^\circ, l^\circ@(u'^\circ::l'^\circ)), \mathbf{nil})^\circ & (*) \\
& = & (t, (u^\circ, (l@(u'::l'))^\circ), \mathbf{nil})^\circ & (**) \\
& = & (t(u, l@(u'::l')))^\circ & \text{(by def. of } (_)^\circ)
\end{array}$$

In $(**)$ we need the easily proved $(l_1@l_2)^\circ = l_1^\circ@l_2^\circ$. In $(*)$ we need

$$(t, (u, l), ((u', l')+L))^\circ \to_\mu (t, (u, l@u'::l'), L)^\circ$$

with all expressions in $\vec{\lambda}\mathbf{nm}$, except $t \in \lambda\mathbf{m}$. The proof is a case analysis of t. In the case t is not a value, one needs the following observation: $L_1 \to L_2$ implies $(t, U, L_1)^\circ \to (t, U, L_2)^\circ$. The latter is easily proved by induction on $t \in \lambda\mathbf{m}$.

Now the base cases in (b). The case $\rho' = \beta_{11}$ follows immediately by a calculation and using the bookkeeping fact 4 above. For the case $\rho' = \beta_2$ we

reason as follows:

$$\begin{array}{rcll}
\Phi((\lambda x.t)(u, v::l, L)) & = & \Phi((\lambda x.\Phi t), (u, v::l), L) & \text{(by def. } \Phi\text{)} \\
& \to_{\beta_2} & \Phi(\mathbf{s}(\Phi u, x, \Phi t), (v, l), L) & (*) \\
& = & \Phi(\Phi(\mathbb{S}(u, x, t)), (v, l), L) & \text{(by fact 4 above)} \\
& = & \Phi(\mathbb{S}(u, x, t)@((v, l), L)) & \text{(by fact 3 above)}
\end{array}$$

where in $(*)$ we use the fact: $t_1(\Phi U_1) \to_\rho t_2(\Phi U_2)$ implies $\Phi(t_1, U_1, L) \to_\rho \Phi(t_2, U_2, L)$, for $\rho = \beta_2$, easily proved by induction on L.

For the base case $\rho' \in \{\mu_1, \mu_2\}$, recall the definition of root ρ'-reduction on terms. We need the following:

$$\Phi(t(\Phi u, \Phi l), (u', l'), L) \to_\zeta \Phi(t, (u, l@(u'::l')), L) \qquad (*)$$

where $t \in \lambda\mathbf{m}$ and the remaining parameters are in $\vec{\lambda}\mathbf{nm}$. This is proved by case analysis of L and requires an observation about Φ analogous to the one used in base case β_2 with $\rho = \zeta$, and the easily proved $\Phi(l@l') = (\Phi l)@(\Phi l')$.

Φ maps a root μ_1-reduction step on terms as follows:

$$\begin{array}{rcll}
\Phi(V, (u, l), ((u', l')+L)) & = & \Phi(V(\Phi u, \Phi l), (u', l'), L) & \text{(by def. of } \Phi\text{)} \\
& \to_\zeta & \Phi(V, (u, l@(u'::l')), L) & \text{(by } (*)\text{)} \\
& = & \Phi(V, (u, l@(u'::l'), L) & \text{(by def. of } \Phi\text{)}
\end{array}$$

Φ maps a root μ_2-reduction step on terms as follows:

$$\begin{array}{rcll}
\Phi(t, U, (u+l+(u'+l'+L))) & = & \Phi(t(\Phi U)(\Phi u, \Phi l), u'::l', L) & \text{(by def. of } \Phi\text{)} \\
& \to_\zeta & \Phi(t(\Phi U), (u, l@(u'::l')), L) & \text{(by } (*)\text{)} \\
& = & \Phi(t, U, (u+l@(u'::l')+L)) & \text{(by def. of } \Phi\text{)}
\end{array}$$

The inductive cases are routine. This ends the proof.