Permutability in proof terms for intuitionistic sequent calculus with cuts

José Espírito Santo
Centro de Matemática, Universidade do Minho, Portugal

Maria João Frade
HASLab/INESC TEC & Universidade do Minho, Portugal

Luís Pinto
Centro de Matemática, Universidade do Minho, Portugal

Abstract

This paper gives a comprehensive and coherent view on permutability in the intuitionistic sequent calculus with cuts. Specifically we show that, once permutability is packaged into appropriate global reduction procedures, it organizes the internal structure of the system and determines fragments with computational interest, both for the computation-as-proof-normalization and the computation-as-proof-search paradigms. The vehicle of the study is a λ-calculus of multiary proof terms with generalized application, previously developed by the authors (the paper argues this system represents the simplest fragment of ordinary sequent calculus that does not fall into mere natural deduction). We start by adapting to our setting the concept of *normal* proof, developed by Mints, Dyckhoff, and Pinto, and by defining *natural* proofs, so that a proof is normal iff it is natural and cut-free. Natural proofs form a subsystem with a transparent Curry-Howard interpretation (a kind of formal vector notation for λ-terms with vectors consisting of lists of lists of arguments), while searching for normal proofs corresponds to a slight relaxation of focusing (in the sense of LJT). Next, we define a process of permutative conversion to natural form, and show that its combination with cut elimination gives a concept of *normalization* for the sequent calculus. We derive a systematic picture of the full system comprehending a rich set of reduction procedures (cut elimination, flattening, permutative conversion, normalization, focalization), organizing the relevant subsystems and the important subclasses of cut-free, normal, and focused proofs.

2012 ACM Subject Classification Theory of Computation → Type Theory, Theory of Computation → Proof Theory

Keywords and phrases sequent calculus, permutative conversion, Curry-Howard isomorphism, vector of arguments, generalized application, normal proof, natural proof, cut elimination, flatenning, normalization, focalization

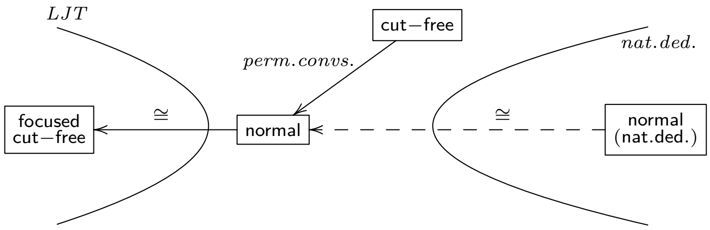
Digital Object Identifier 10.4230/LIPIcs.TYPES.2016.10

1 Introduction

Traditionally, the sequent calculus is associated with the computation-as-proof-search paradigm [16], but progress in the understanding of the Curry-Howard correspondence showed that sequent calculus has a lot to offer to the computation-as-proof-normalization paradigm as well, from alternative λ-term representations which are useful for machine handling [12, 2] to logical foundations for evaluation strategies [3, 24]. Nevertheless, the mentioned progress has been slow: even if we are not anymore in the situation where textbooks had almost nothing to report about Curry-Howard for sequent calculus [11, 22, 18], it seems basic discoveries are still being made after decades of investigation [1, 5].

22nd International Conference on Types for Proofs and Programs (TYPES 2016).
Editors: Silvia Ghilezan, Herman Geuvers, and Jelena Ivetić; Article No. 10; pp. 10:1–10:27
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Figure 1 The cut-free setting



One source of difficulties in completing the Curry-Howard interpretation of sequent calculus and cut-elimination is the phenomenon of permutability of inferences [14], which sometimes is dubbed "bureaucracy". Permutability can be faced with several attitudes: either by decreeing Curry-Howard for sequent calculus an outright impossibility [11]; or by regarding the sequent calculus as meta-notation for alternative, supposedly permutation-free formalisms, like natural deduction [19] or proof nets [10]; or by restricting one's attention to permutability-free fragments of sequent calculus - cf. the flourishing area of focusing [15, 21].

In this paper we face permutability squarely, in the context of intuitionistic propositional logic, for a simple and standard sequent calculus including cut, the latter system presented as a typed λ-calculus, and we show that the (perhaps dull) complexity engendered by permutability can be tamed and organized appropriately and meaningfully, in a way that enlightens the internal structure and the computational interpretation of the entire sequent calculus.

Our starting point is the familiar situation in the cut-free setting, depicted in Fig. 1: there is a set of permutation-free proofs, named *normal* by Mints [17], which are in 1-1 correspondence with normal natural deductions; in addition [4]: (i) normal derivations are normal (i.e. irreducible) w.r.t. a rewriting system of permutative conversions; (ii) normal derivations are in 1-1 correspondence with cut-free LJT-proofs (that is, cut-free $\overline{\lambda}$-terms [12]). So permutation-freeness has a privileged relationship with natural deduction (as we already knew since Zucker [25]); and, in this setting, permutation-freeness is indistinguishable from focusedness (in the sense of LJT).

What is the high-level lesson of this situation? Permutability can be organized into a reduction procedure determining a class of normal forms which are meaningful both for functional computation and for proof-search. Shorter: if permutability of inferences is packaged into a global reduction procedure, it becomes an organizing tool at the macro level that brings out meaning.

In this paper, guided by this heuristic, we move to the cut-full setting. Needless to say, the situation becomes rather more complex, as cut-elimination is present and potentially interacts with permutability, we have to deal with (sub)systems of the full rewriting system rather than classes of normal forms, and desirably the familiar cut-free situation falls out as a corollary of the cut-full picture.

In a nutshell, these are our results: we adapt to our setting the notion of *normal* proof [17, 4] and pin down the bottom-up proof-search procedure it determines, which is a slight relaxation of focusing; we introduce a permutation-free notion of *natural* proof so that a proof is normal iff it is natural and cut-free; we prove natural proofs are closed for cut-elimination, constituting a subsystem with a transparent Curry-Howard interpretation; we prove natural proofs are the normal forms w.r.t. a certain permutative conversion γ; we give a systematic description of the internal structure of the sequent calculus we consider in terms of the two "bureaucratic" conversions: the conversion γ, and another conversion named μ, which, among other things, is the bridge between natural proofs and focused proofs; we investigate the commutation between the (macro level) reduction procedures and this allows us to identify a *normalization* procedure on the set of all sequent calculus proofs, for which the normal proofs are the irreducible forms, and which is a combination of cut-elimination and permutative conversion.

Technical overview. In order to isolate the syntactic difficulties caused by permutability, we reduce the logical apparatus to a minimum: intuitionistic implication is the single connective studied, and the sequent calculus analyzed is designed to be the simplest one that goes beyond natural deduction with general elimination rules [23] (i.e. beyond the λ-calculus with generalized application ΛJ [13]). Quite conveniently, the resulting system is precisely the $\lambda\mathbf{Jm}$-calculus introduced by two of the authors [8, 9] and further studied in [6] - a system which may be seen as the "multiary" [20] version of ΛJ. Multiarity just means that the generalized application constructor handles a non-empty list of arguments, thus ΛJ may be recast as the *unary fragment* $\lambda\mathbf{J}$, where the list of arguments is singular [9]; but multiarity engenders the mentioned conversion μ, firstly introduced in [20] as a technical tool in a termination argument, which turns out to play a crucial role in the description of the internal structure of $\lambda\mathbf{Jm}$ and its subtle connection with natural deduction [8, 6].

In extending the situation in Fig. 1 to the cut-full setting, we have to avoid an immediate pitfall: to consider an excessively narrow class of derivations possibly containing cuts. Ordinary cut-free derivations may be seen as fully-normal natural deductions with general application [23]. So, if we merely "close under substitution" such derivations, we end up with natural deduction, or the ΛJ-calculus [13]. Similarly, if we merely add appropriate cut-rules to LJT, we end up with some variant of the $\overline{\lambda}$-calculus [12]. We do something different: we recast in $\lambda\mathbf{Jm}$ (a system designed to not fall in mere natural deduction) the situation in Fig. 1, and the result is illustrated in Fig. 2. In $\lambda\mathbf{Jm}$, natural deduction and LJT are captured internally[1], and the normal derivations of [17, 4] are just the unary case of a more general concept of normal derivation, which is studied here for the first time, as it escaped the catalogue of normal forms in [6].

In fact, we will rather develop Fig. 3, concerning the cut-full setting, and extract Fig. 2 as a corollary, given that cut-elimination links each system in Fig. 3 to a corresponding class in Fig. 2. Specifically: Section 3 defines and studies natural derivations and how they define a subsystem $\lambda\mathbf{nm}$ with clear computational interpretation. This includes studying the cut-free natural (=normal) derivations, in particular in their relation to focused proofs. Section 4 goes beyond the permutation-free fragment $\lambda\mathbf{nm}$, and studies permutative conversion γ, for which the natural proofs are the irreducible forms. This includes studying the interaction of γ with cut-elimination, which leads to the definition of normalization in $\lambda\mathbf{Jm}$. Section 2 recapitulates $\lambda\mathbf{Jm}$, while Section 5 concludes.

[1] This is in contrast with [4], where natural deduction, LJT and sequent calculus are three different systems - this is why in Fig. 1 we see the curved borders, while in Fig. 2 these borders disappear, their location being memorized with dotted lines. Beware that there are several inclusion that hold in Fig. 2, since all classes live in the same system: the class of unary cut-free (resp. unary normal) derivations is included in the class of cut-free (resp. normal) derivations; and normal natural deductions are a subclass of unary cut-free derivations ($\cong$ fully normal natural deductions with general eliminations). Such inclusions are not depicted to avoid clutter and because they are not witnessed by reduction rules of $\lambda\mathbf{Jm}$. The map denoted with a dashed line is not a mere inclusion, but is not studied in this paper. Similar remarks apply as well to Fig. 3.

Figure 2 The cut-free setting in the multiary calculus $\lambda\mathbf{Jm}$ (classes and maps)

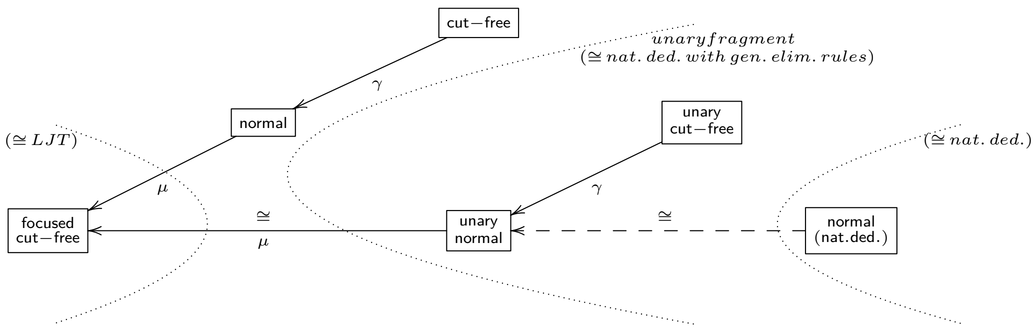
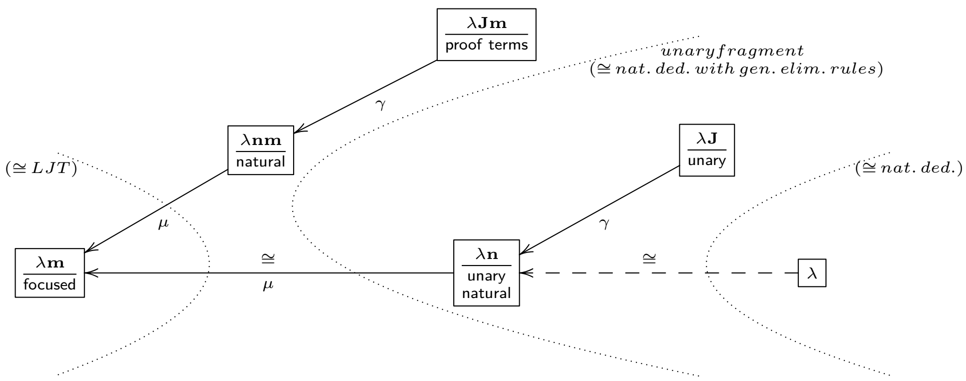


Figure 3 The cut-full setting in the multiary calculus $\lambda\mathbf{Jm}$ (calculi and morphisms)



2 The sequent calculus $\lambda\mathbf{Jm}$

In the first subsection we recall $\lambda\mathbf{Jm}$, while in the second we argue why $\lambda\mathbf{Jm}$ is a simple and standard presentation of the intuitionistic sequent calculus.

2.1 A recapitulation of $\lambda\mathbf{Jm}$

Proof expressions and typing. Expressions E are generated by the following grammar:

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

We will just say "term" instead of "proof term". A *value* V is a term of the form x or $\lambda x.t$. The word "continuation" is chosen for its intuitive appeal, with no connection with technical meanings of the word intended.[2]

[2] In the previous publications on $\lambda\mathbf{Jm}$, the system was presented with two syntactic classes only: terms and lists. In fact, since continuations are generated by a single constructor and used only once in the grammar (in the formation of gm-arguments), they could easily be dispensed with; and the very same holds of gm-arguments. However, the separation into finer classes gives more flexibility. This flexibility is a convenience, as quite often we can avoid writing the entire expression $t(u, l, (x)v)$ - see e.g. the simpler definition of reduction rules π and μ; but such flexibility is also a necessity - see the particular form of continuations (called pseudo-lists) extensively studied in the next section.

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

$$\frac{}{x\!:\!A,\Gamma\vdash x\!:\!A}\;Axiom \qquad \frac{x\!:\!A,\Gamma\vdash t\!:\!B}{\Gamma\vdash\lambda x.t\!:\!A\supset B}\;Right$$

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

$$\frac{}{\Gamma;C\vdash[]\!:\!C}\;Ax \qquad \frac{\Gamma\vdash u\!:\!A\quad \Gamma;B\vdash l\!:\!C}{\Gamma;A\supset B\vdash u::l\!:\!C}\;Lft \qquad \frac{x:C,\Gamma\vdash v\!:\!D}{\Gamma|C\vdash(x)v\!:\!D}\;Select$$

We identify simple types with formulas of intuitionistic, propositional, implicational logic. They are ranged over by A, B, C, D. If $B = B_1 \supset \cdots \supset B_n$ $(n \geq 1)$ then we say C is a *suffix* of B if $C = B_j \supset \cdots \supset B_n$, for some $1 \leq j \leq n$. Contexts Γ are sets of variable declarations $x : A$, with at most one declaration per variable. The typing rules are in Fig. 4. They handle four kinds of sequents, one per syntactic class:

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

In the sequents of kinds (ii) and (iii), the distinguished formula on the left hand side (the formula separated by ;) is main in the last inference, whereas in the sequents of kind (iv) the distinguished formula C is merely selected from the context. In addition, in a derivable sequent of kind (iii), C is a suffix of B.

Inference rule Lft is a special left-introduction rule, because its right premiss is a sequent of kind (iii): this implies that $B = B_1 \supset \cdots \supset B_m \supset C$, for some $m \geq 0$, and the referred premiss is the conclusion of a chain of m other Lft inferences. There is another primitive left introduction rule, $Leftm$, the single rule for typing gm-arguments. Since its middle premiss is a sequent of kind (iii), the main formula of $Leftm$ has the form $A \supset B_1 \supset \cdots \supset B_m \supset C$, for some $m \geq 0$, and is obtained after a sequence of $m+1$ left introductions. We call $Leftm$ a *multiary* left-introduciton rule, while its particular case where the middle premiss is the conclusion of Ax, $m = 0$, $l = []$, and $B = C$ may be called a *unary* left introduction.

In $\Gamma;A \supset B \vdash a : D$, with $a = (u,l,c)$, and $\Gamma;A \supset B \vdash l' : C$, the formula $A \supset B$ is introduced linearly, *i.e* without contraction, in the last inference; the difference between the two sequents is that C is a suffix of B, whereas the same is not true of D, unless $c = (x)x$. The trivial cut xa gives name x to the formula $A \supset B$: we have the admissible rules

$$\frac{\Gamma;A\supset B\vdash a\!:\!D}{\Gamma\vdash xa\!:\!D}\;Unselect \qquad \frac{\Gamma\vdash u\!:\!A\quad \Gamma;B\vdash l\!:\!C\quad \Gamma|C\vdash c\!:\!D}{\Gamma\vdash x(u,l,c)\!:\!D}$$

where $(x : A \supset B) \in \Gamma$. So xa represents simultaneously an inference that "unselects" an antecedent formula, and a form of left introduction without linearity constraint.

If $x \notin a$ and $t = xa$ we say x is *main and linear in the application* t (abbreviation $mla(x,t)$). In that case, $c = (x)xa$ represents an argument a coerced to a continuation. The admissible typing rule is

$$\frac{\Gamma;A\supset B\vdash a\!:\!D}{\Gamma|A\supset B\vdash(x)xa\!:\!D} \qquad \text{due to} \qquad \cfrac{\cfrac{\cfrac{\Gamma;A\supset B\vdash a\!:\!D}{x:A\supset B,\Gamma;A\supset B\vdash a\!:\!D}\;Weak}{x:A\supset B,\Gamma\vdash xa\!:\!C}\;Unselect}{\Gamma|A\supset B\vdash(x)xa\!:\!D}\;Select \tag{2}$$

where $(x : A \supset B) \notin \Gamma$. In the first figure we see that the distinguished position in the l.h.s. is changed, losing the information about linearity. In general, there is no coercion of a continuation to an argument or list. As hinted above, a non-empty list $u :: l$ can be coerced to an argument $(u, l, (x)x)$ (and then to a continuation). A direct "coercion" of a list to a continuation is given by $[]^\natural = (x)x$ and $(u :: l)^\natural = (x)x(u, [], l^\natural)$, with $x \notin u, l$. The admissible typing rule is

$$\frac{\Gamma;B \vdash l : C}{\Gamma|B \vdash l^\natural : C} \qquad (3)$$

Derived syntax. In order to formulate the reduction rules, we have to introduce some derived syntactic operations. A familiar one is ordinary substitution of variables by terms, denoted $\mathbf{s}(t, x, E)$. It is becoming increasingly clear [12, 5] (and this paper just confirms this) that mechanisms of vectorization of arguments for functional applications are at the heart of the computational interpretation of sequent calculus. Here is a careful definition of dixappend operations in $\lambda\mathbf{Jm}$:

▶ **Definition 1** (Append operations).

1. The term $t@a$ is defined by $V@a = Va$ if V is a value; and by $(ta')@a = t(a'@a)$.
2. The argument $a'@a$ is defined by $(u, l, c)@a = (u, l, c@a)$.
3. The continuation $c@a$ is defined by $((x)v)@a = (x)(v@_x a)$.
4. The term $v@_x a$ is defined by $(xa')@_x a = x(a'@a)$ if $x \notin a'$; and by $v@_x a = va$, otherwise.
5. The continuation $c@c'$ is defined by: $((x)x)@c' = c'$; $((x)x(u, l, c))@c' = (x)x(u, l, c@c')$, if $x \notin u, l, c$; and $((x)v)@c' = (x)(v@c')$, otherwise.
6. The term $t@c$ is defined by $t@(x)v = \mathbf{s}(t, x, v)$.
7. The list $l@l'$ is defined by $[]@l' = l'$ and $(u :: l)@l' = u :: (l@l')$.

Some immediate comments about these append operators: $t@a$ will be used in the definition of a special substitution operator (Def. 39 in Section 4); $v@_x a$ is used in the definition of $c@a$, and the idea goes back to [8]; $a@a'$ allows a very short definition of the reduction rule π; $c@a$ is used in the definition of $a@a'$; $c@c'$ will allow the definition of $L@L'$ in Section 3; $l@l'$ is necessary for the definition of reduction rule μ.

Recall that an argument a can be "coerced" to a continuation $(z)za$, if $z \notin a$. The next lemma shows $c@a$ could have been defined via $c@c'$.

▶ **Lemma 2** (Coherence of append). **1.** *$c@a = c@(z)za$, if $z \notin a$.*
2. *$(x)(v@_x a) = ((x)v)@(z)za$, if $x, z \notin a$.*

Proof. By simultaneous induction on c and v. It is interesting to see how the various definitions in Def. 1 cooperate to produce the result. ◀

▶ **Lemma 3** (Admissible typing rules). *The typing rules in Fig. 5 are admissible.*

Proof. Rule (i) follows immediately from rule (ii). Rules (ii), (iii) and (iv) are proved by simultaneous induction on a', c and v. Rule (vi) follows immediately from rule (vii). Rule (vii) is proved together with similar statements for a, l and c by simultaneous induction. Rule (v) follows by induction on c with the help of rule (vi). Rule (viii) is proved by induction on l'. ◀

Figure 5 Typing rules for derived syntactic operators

$$\frac{\Gamma \vdash t : A \supset B \quad \Gamma ; A \supset B \vdash a : C}{\Gamma \vdash t@a : C}\ (i) \qquad \frac{\Gamma ; A \supset B \vdash a' : C_1 \supset C_2 \quad \Gamma ; C_1 \supset C_2 \vdash a : D}{\Gamma ; A \supset B \vdash a'@a : D}\ (ii)$$

$$\frac{\Gamma | A \vdash c : B_1 \supset B_2 \quad \Gamma ; B_1 \supset B_2 \vdash a : C}{\Gamma | A \vdash c@a : C}\ (iii) \qquad \frac{x : D, \Gamma \vdash v : A \supset B \quad \Gamma ; A \supset B \vdash a : C}{x : D, \Gamma \vdash v@_x a : C}\ (iv)$$

$$\frac{\Gamma | C \vdash c : D \quad \Gamma | D \vdash c' : E}{\Gamma | C \vdash c@c' : E}\ (v) \qquad \frac{\Gamma \vdash t : A \quad \Gamma | A \vdash c : B}{\Gamma \vdash t@c : B}\ (vi) \qquad \frac{\Gamma \vdash t : A \quad \Gamma, x : A \vdash v : B}{\Gamma \vdash \mathbf{s}(t, x, v) : B}\ (vii)$$

$$\frac{\Gamma ; A \vdash l : B \quad \Gamma ; B \vdash l' : C}{\Gamma ; A \vdash l@l' : C}\ (viii)$$

Figure 6 Reduction rules of $\lambda\mathbf{Jm}$

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

So, every derived syntactic operator is typed with a corresponding variant of the cut rule, and each such operator is the term representation of the operation on derivations produced by the elimination of the corresponding cut. Such operations on derivations may be extracted from the proof of the previous lemma. All of them, except for the cuts (v), (vi) and (vii), consist in permuting the cut to the left, as long as this is made possible by the repetition of the cut formula; for cuts (vi) and (vii) the corresponding operation performs a similar permutation to the right; for cut (v) the operation is an hybrid of permutation to the left and to the right.

Reduction rules. The reduction rules of $\lambda\mathbf{Jm}$ are given in Fig. 6. All rules but μ are relations on terms, while μ is a relation on arguments. We let $\beta := \beta_1 \cup \beta_2$. Rule μ is the "abbreviation" conversion due to [20]. Rule π of this paper is not the "lazy" variant of [8, 9], where argument a' is appended to argument a in a stepwise fashion, but rather corresponds to the rule π' of the cited papers. This is due to the definition of $v@_x a$, which is not merely va, but instead triggers a new appending process in some cases.[3] See some remarks about the computational interpretation of these rules after Lemma 4.

The compatible closure $\to_R$ of a reduction rule R is obtained by closing R under the rules in Fig.7[4]. We use the notations $\to_R^=$, $\to_R^+$, and $\to_R^*$ to denote the reflexive, the transitive, and the reflexive-transitive closure of $\to_R$, respectively. If $R = R_1 \cup R_2$, $\to_R$ can be denoted $\to_{R_1R_2}$ (e.g. $\to_{\beta\pi}$). A *R-normal form* (or R-nf, for short) is an expression E such that $E \to_R E'$ for no E'. When existing, we write $\downarrow_R (E)$ to denote the unique R-nf of an

[3] In ΛJ [13] rule π is also of the "lazy" kind.
[4] This detailed naming of the closure rules will be intensively used in Section 3, where we will consider alternative notions of compatible closure.

Figure 7 Compatible closure

$$\frac{t \to t'}{\lambda x.t \to \lambda x.t'}\ (I) \qquad \frac{t \to t'}{ta \to t'a}\ (II) \qquad \frac{a \to a'}{ta \to ta'}\ (III)$$

$$\frac{u \to u'}{(u,l,c) \to (u',l,c)}\ (IV) \qquad \frac{l \to l'}{(u,l,c) \to (u,l',c)}\ (V) \qquad \frac{c \to c'}{(u,l,c) \to (u,l,c')}\ (VI)$$

$$\frac{u \to u'}{u :: l \to u' :: l}\ (VII) \qquad \frac{l \to l'}{u :: l \to u :: l'}\ (VIII) \qquad \frac{v \to v'}{(x)v \to (x)v'}\ (IX)$$

expression E.

The following result will be important later, and closes the discussion of derived syntax.

▶ **Lemma 4** (Associativity of append).

1. $(t@a)@a' \to_\pi^= t@(a@a')$ *and* $(t@_x a)@_x a' \to_\pi^= t@_x(a@a')$.
2. $(a@a')@a'' \to_\pi^= a@(a'@a'')$.
3. $(c@a)@a' \to_\pi^= c@(a@a')$.

Proof. By simultaneous induction in t, a and c. Everything follows from definitions and IHs, except in the single case where π-steps are generated, which is this: suppose t is neither x nor xa with $x \notin a$. Then $(t@_x a)@_x a' = (ta)a' \to_\pi t(a@a') = t@_x(a@a')$. ◀

Cut-elimination and computational interpretation. Rules β and π define a cut-elimination procedure in $\lambda\mathbf{Jm}$, whose purpose is not to eliminate all cuts ta, but rather to reduce them to the form xa: as seen above, xa represents a left introduction, not a cut to be eliminated. Still, we refer to $\beta\pi$-nfs as *cut-free*. A cut ta is necessarily principal on the right premiss, so its elimination starts by analyzing the left premiss t. If t is not a variable, then either it is another cut (in which case the original cut ta is permutable to the left, and a π-redex), or it is a λ-abstraction (in which case the cut is principal in both premisses, and a β-redex). Rule π performs left permutation, while rule β performs the key step of cut-elimination, breaking the cut into two cuts with simpler cut-formulas. If any of these two cuts is permutable to the right, it is not formed, but rather eliminated immediately, and represented by a substitution.

In a μ-redex we find a continuation $c = (x)xa'$ with $x \notin a'$, which represents a derivation of the form found in the right figure of (2), where a formula is selected immediately after being "unselected". The redex itself is a sequence of two $Leftm$ inferences, with the first, represented by a', being coerced to a continuation c, before being used in the second $Leftm$ inference (u, l, c). In addition, xa' represents a left introduction with the principal formula being introduced linearly, due to the proviso $x \notin a'$. The construction $u' :: l'$ found in the *contractum* of rule μ represents a linear left introduction by alternative and more primitive means, dispensing with the temporary name x, and eliminating the described sequence of inferences.

We also refer to ta as a *generalised, multiary application* (or gm-application for short), and think of $\lambda\mathbf{Jm}$ as a λ-calculus with the *multiarity* and *generality* features. In ta, t is the function expression, a is its gm-argument. A gm-argument consists of a first ordinary argument u, a list l of further ordinary arguments l (the multiarity feature), and a "continuation" c, indicating where to substitute the result of passing the last argument (the generality feature). This interpretation follows from the reduction rules β_1 and β_2. A π-redex is an iterated gm-application. Contrary to ordinary arguments in, say, the λ-calculus, gm-arguments can be appended and the function expression simplified - this is the effect of the π-reduction. In a μ-redex, the generality feature is being used just to "link" two lists of arguments. The effect of the μ-reduction is to append these two lists. In the sequel, these interpretation of π and μ will be specialized to a fragment of $\lambda\mathbf{Jm}$; there, it will become appropriate to call μ-nfs *flat* expressions, and to call μ-normalization *flattening*. We adopt such terminology for the entire $\lambda\mathbf{Jm}$. For instance, μ-nfs constitute a subsystem of $\lambda\mathbf{Jm}$ [8]; we call it the *flat subsystem*.

Properties. The meta-theory of $\lambda\mathbf{Jm}$ is well developed, we just recall the results we need below. Some proofs have to be adapted to cover the variant of π we employ here.

▶ **Theorem 5** (Confluence and SN). *In $\lambda\mathbf{Jm}$, $\beta\pi$- and $\beta\pi\mu$-reductions are confluent, and $\beta\pi\mu$-reduction is SN on typable expressions.*

Proof. The existing proofs are easily adapted. ◀

In isolation, μ-reduction is easily seen to be confluent and terminating [8, 9]. The μ-nf of an expression E, $\mu(E)$, is defined by recursion on E, with all clauses given homomorphicaly, except in the following case: if $\mu v = x(u', l', (y)v')$ and $x \notin u', l', v'$, then $\mu(t(u, l, (x)v)) = \mu t(\mu u, \mu l @(u' :: l'), (y)v')$.

▶ **Lemma 6** (Preservation of cut-freeness by μ-reduction). *In $\lambda\mathbf{Jm}$, if t is a $\beta\pi$-nf and $t \to_\mu t'$, then t' is a $\beta\pi$-nf.*

Proof. Easy induction on $t \to_\mu t'$. ◀

This lemma says μ-reduction preserves cut-freeness. Conversely, neither β-reduction nor π-reduction preserve μ-normality. Given a reduction rule R, by *R'-reduction* we will mean R-reduction followed by reduction to μ-nf.

▶ **Theorem 7** (Preservation of reduction by μ). *In $\lambda\mathbf{Jm}$:*

1. *If $t \to_\beta t'$ then there exists t'' s.t. $\mu(t) \to_\beta t'' \to_\mu^* \mu(t')$.*
2. *If $t \to_\pi t'$ then there exists t'' s.t. $\mu(t) \to_\pi t'' \to_\mu^* \mu(t')$.*

Proof. Statement 1 of the previous theorem is already used in [8] (Lemma 5), while statement 2 is also present in [8] (Lemma 7), but only for the terms in the $\lambda\mathbf{J}$-subsystem. ◀

Subsystems. A $\lambda\mathbf{m}$-expression is a $\lambda\mathbf{Jm}$-expression where all gm-applications have the form $t(u, l, (x)x)$, a form which we abbreviate as $t(u, l)$ and call *multiary application*. Based on such expressions one defines a subsystem $\lambda\mathbf{m}$ of $\lambda\mathbf{Jm}$: the expressions are μ-nfs; they are closed for β; they are not closed for π, but we return to the subsystem by post-composition with μ-normalization. The reduction rules of $\lambda\mathbf{m}$ are given in Fig. 8. The $\lambda\mathbf{m}$-calculus is a variant of the λ-calculus, called the *multiary λ-calculus*, or $\lambda\mathbf{m}$-calculus, where functions are applied to non-empty lists of arguments. The rules β_i pass to the function the first argument, adjusting the remainder of the list, while rule π' appends lists of arguments. The $\lambda\mathbf{m}$ is also a variant of the $\overline{\lambda}$-calculus [12]. The normal forms of $\lambda\mathbf{m}$ are either x, $\lambda x.t$ or $x(u, l)$, which are a variant of the cut-free $\overline{\lambda}$-terms, and represent the cut-free *LJT* derivations. For this reason $\lambda\mathbf{m}$ is also the *focused* subsystem of $\lambda\mathbf{Jm}$.

A $\lambda\mathbf{J}$-expression is a $\lambda\mathbf{Jm}$-expression where all gm-applications have the form $t(u, [], (x)v)$ (hence, just one argument u), a form which we abbreviate as $t(u, (x)v)$ and call *generalized application*. Such expressions define the *unary* subsystem $\lambda\mathbf{J}$ of $\lambda\mathbf{Jm}$, as they are closed for β_1 and π. This is a copy of natural deduction with general elimination rules [23], or

Figure 8 The reduction rules of the multiary λ-calculus $\lambda\mathbf{m}$

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

rather its presentation as the typed λ-calculus ΛJ [13], inside $\lambda\mathbf{Jm}$ [9]. Conversely, $\lambda\mathbf{Jm}$ is a generalization of $\lambda\mathbf{J}$ obtained by allowing the left-introduction rule Lft, or constructor $u::l$. This is a small difference with numerous consequences: reduction rules β and π of ΛJ have to be taken in a multiary form, and two new reduction rules, β_2 and μ, appear; lists are not restricted to $[]$, so the syntactic class of lists, as well as the third form of sequents $\Gamma;B\vdash l:C$, are not degenerate. So $\lambda\mathbf{Jm}$ is a system that goes slightly but decisively beyond natural deduction.

2.2 Why λJm?

We choose to base on $\lambda\mathbf{Jm}$ our study of permutability in the sequent calculus. Before we proceed, we would like to justify our choice. The justification has two parts. First, we give a fresh explanation of the place of $\lambda\mathbf{Jm}$ among possible formulations of the sequent calculus, trying to dissipate some misunderstandings. Second, we explain our methodology in the study of permutability, and why $\lambda\mathbf{Jm}$ is an adequate tool for that methodology.

Understanding λJm. Given that $\lambda\mathbf{Jm}$ captures several known systems as subsystems, one might have the impression that $\lambda\mathbf{Jm}$ is some *ad hoc* gluing. Of course we think otherwise, and would like to argue that $\lambda\mathbf{Jm}$ is rather a standard and important fragment of sequent calculus. Actually, this has been argued technically before [7], but the sceptical reader may object against the formulations of the sequent calculus with which $\lambda\mathbf{Jm}$ is compared in *op. cit.* So, here we formulate *ordinary* sequent calculus as a λ-calculus named $\lambda\mathbf{LJ}$, and show what fragment of this calculus $\lambda\mathbf{Jm}$ corresponds to.

The proof expressions of $\lambda\mathbf{LJ}$ are given by the following grammar:

$$\begin{array}{rrcl}(LJ\text{-proof terms}) & t,u,v & ::= & x \mid \lambda x.t \mid x\hat{}(u;c) \mid tc\\ (LJ\text{-continuations}) & c & ::= & (x)v\end{array}$$

The various term forms represent, respectively, the inference rules axiom, right introduction, left introduction, and cut; the continuation $(x)v$ represents a selection. Separating the class of continuations is convenient, as they are used twice in the grammar of terms. The formulation of the system as a typing system is quite obvious, here are most of the rules:

$$\frac{\Gamma\vdash u:A \quad \Gamma|B\vdash c:C}{\Gamma\vdash x\hat{}(u;c):C}\ ((x:A\supset B)\in\Gamma) \qquad \frac{\Gamma\vdash t:A \quad \Gamma|A\vdash c:B}{\Gamma\vdash tc:B} \qquad \frac{x:B,\Gamma\vdash v:C}{\Gamma|B\vdash (x)v:C}$$

We will specify a subset of the set of LJ-terms, whose elements are called *Jm-terms*, by imposing two restrictions. The first restriction is that no Jm-term has the third form, corresponding to a left introduction. One reason for this is that we want a term to be either a value (variable or abstraction) or a single other form: having to sacrifice either left introduction or cut, there is no doubt the first form is the chosen to be sacrificed, since cuts represent computation, and can mimic left introductions.

This first restriction on terms determines three subsets of the set of LJ-continuations: (i) *Jm-continuations* $(x)v$, where v is a Jm-term; (ii) LJ-continuations of the form $(x)x\hat{}(u;c')$, to be called *Jm-arguments*, where $x \notin u, c'$, and u is a Jm-term, and c' is to be specified soon; (iii) the union of these two subsets, to be ranged over by k, whose elements are to be called *Jm-contexts*. Notice a Jm-argument $(x)\hat{x}(u; c')$ is not a Jm-continuation, because $\hat{x}(u; c')$ is not a Jm-term.

We have to specify which class c' in Jm-arguments belongs to; and the same is true of Jm-cuts, terms of the form tc'' with t a Jm-term and c'' to be specified now. We impose (and this is the second restriction on terms) c'' to be a Jm-argument: this implies that a Jm-cut is right-principal and a generalized form of function application, the cut-formula is an implication; this also justifies the terminology "Jm-argument". As to c': (i) imposing it to be a Jm-argument is not an option, otherwise the inductive definition of Jm-arguments would not have a base case; (ii) imposing it to be a Jm-continuation is not an option either, as otherwise Jm-terms would be isomorphic to ΛJ-terms, and the fragment would be equivalent to natural deduction; (iii) so we have to choose c' to be a Jm-context. Therefore, a Jm-argument is a LJ-continuation of the form $(x)\hat{x}(u; k)$, where $x \notin u, k$, and u is a Jm-term and k is a Jm-context. A Jm-argument $(x)\hat{x}(u; k)$ is abbreviated (u, k).

Summing up: the sets of Jm-terms, arguments, contexts, and continuations are given by

$$t, u, v ::= x \mid \lambda x.t \mid ta \qquad a ::= (u, k) \qquad k ::= a \,|\, c \qquad c ::= (x)v \tag{4}$$

We now see this syntax is a formulation of $\lambda\mathbf{Jm}$, let us call it the first formulation.

In fact, the syntax of $\lambda\mathbf{Jm}$ has many equivalent formulations. From (4) we can dispense with the class of arguments: cuts become $t(u, k)$ and contexts are given by $k ::= (u, k) \,|\, c$. This second formulation was used in [7]. Alternatively, from (4) we can dispense with the class of contexts: in this third formulation, which has never been used, arguments are given by $a ::= (u, a) \,|\, (u, c)$. In this paper we are using a fourth formulation: in (4), it is equivalent to take contexts as given by $k ::= (u, k) \,|\, c$; then arguments have the general form $(u_1, (u_2, (\cdots (u_m, c) \cdots)))$ for some $m \geq 1$; finally, we bring c to the surface of arguments, rearranging them as: $(u_1, (u_2 :: \cdots :: (u_m :: []) \cdots), c)$. To have c at the surface of arguments will be important precisely for the formulation of the process of permutative conversion[5].

So, $\lambda\mathbf{Jm}$ has several formulations, we are using one that suits better the purpose of this paper; but, independently of the several formulations, $\lambda\mathbf{Jm}$ has a special status, as it is a syntax that follows necessarily from $\lambda\mathbf{LJ}$ by imposing proof terms to be either values or cuts, and cuts to be restricted to a form of function application.

Methodology. Our methodology in the study of permutability in the sequent calculus is modular: we want to isolate and highlight the syntactic intricacies of permutability, avoiding to mix them with other issues that a wrong choice of system could bring. So, we need in the background a system as simple and as close to the ordinary λ-calculus as possible - but without falling into mere natural deduction or ΛJ (which would be undesirable in a study about the sequent calculus).

The system $\lambda\mathbf{Jm}$ has a number of characteristics appropriate to this aim (some of which were stressed by the reconstruction of $\lambda\mathbf{Jm}$ inside $\lambda\mathbf{LJ}$ given above). First, the logic we consider is the simplest one (intuitionistic implication as sole connective). Second, the cut=redex paradigm [12, 3] is not followed, so that variables in proof terms can be treated as ordinary term variables, and substitution can be treated as ordinary term substitution [5]. Third, the primitive cut of the system is right-principal, hence a cut-formula is always an implication, hence the cut can be interpreted as some sort of function application;

[5] See equation (7) below.

concomitantly, substitution is treated as a meta-operation (no explicit substitution), with the corresponding cut-rule treated as an admissible typing rule. Fourth, the immediate call of substitution(s) in the β-rules induces the call-by-name character of cut-elimination [3], which is the approach closest to the ordinary λ-calculus.

3 Permutation-freeness in the sequent calculus $\lambda\mathbf{Jm}$

In this section we study natural proofs, which are a generalization of normal proofs to the cut-full setting. They are introduced in the second subsection, after a technical subsection which develops the concept of pseudo-list. After natural proofs are proved to be closed for typing and reduction, they are given a computational interpretation in the third subsection, through the calculus $\lambda\mathbf{nm}$, which we prove to be isomorphic to the natural subsystem. In the final subsection we investigate the relationship between natural and focused proofs, paying particular attention to the search for normal proofs.

3.1 Pseudo-lists

The notion of x-normality goes back to [4] and was used in the context of $\lambda\mathbf{Jm}$ in [6]. Here we rename the notion as x-naturality, since we are not restricted to the cut-free setting. The concept of *pseudo-list* arises from the particular syntactic organization of $\lambda\mathbf{Jm}$ we employ in this paper, which includes the syntactic class of continuations c. In the remainder of the paper, pseudo-lists will be crucial in the study of naturality. In this subsection we see some of their basic properties, and their use in the analysis of continuations and gm-applications.

▶ **Definition 8** (Pseudo-lists). *x-natural* terms and arguments and *pseudo-lists* are defined simultaneously as follows:

- v is x-natural if $v = x$ or $v = xa$ and a is x-natural.
- a is x-natural if $a = (u, l, c)$ and $x \notin u, l, c$ and c is a pseudo-list.
- c is a pseudo-list if $c = (x)v$ with v x-natural.

Pseudo-lists are ranged over by L. We introduce the following abbreviations for pseudo-lists:

$$L ::= \mathbf{nil} \,|\, (u{+}l+L) \tag{5}$$

- $\mathbf{nil}$ abbreviates $(x)x$
- $(u{+}l+L)$ abbreviates $(x)x(u, l, c)$ if L abbreviates c and $x \notin u, l, c$.

▶ **Lemma 9** (Typing of pseudo-lists). **1.** *In $\lambda\mathbf{Jm}$ a typing derivation of $\Gamma|C \vdash L : D$ ends with an application of the Select inference rule which has one of two forms:*
  - *either the inference selects the left-principal formula of an Axiom inference (with the whole derivation consisting of the two mentioned inferences);*
  - *or the inference ends a derivation of the form of the right figure in (2) - which entails that the Select inference selects a formula which had just been unselected, and the latter, being the distinguished formula in the l.h.s. of a sequent of kind (ii), is the principal formula of a $Leftm$ inference.*

2. *The typing rules for pseudo-lists in Fig. 9 are admissible typing rules of $\lambda\mathbf{Jm}$.*

Proof. 1. is by case analysis on L. The case Axm of 2. uses 1. and the case $multi - Lft$ of 2. uses admissibility of weakening for pseudo-lists. ◀

Figure 9 Typing rules for pseudo-lists

$$\frac{}{\Gamma|A\vdash \mathbf{nil}:A}\,Axm \qquad \frac{\Gamma\vdash u:A \quad \Gamma;B\vdash l:C \quad \Gamma|C\vdash L:D}{\Gamma|A\supset B\vdash (u{+}l+L):D}\,multi-Lft$$

Figure 10 Closure rules for pseudo-lists

$$\frac{u\to u'}{(u{+}l+L)\to(u'{+}l+L)}\,(a) \qquad \frac{l\to l'}{(u{+}l+L)\to(u{+}l'+L)}\,(b) \qquad \frac{L\to L'}{(u{+}l+L)\to(u{+}l+L')}\,(c)$$

▶ **Lemma 10** (Derived substitution rules). *$\mathbf{s}(u,x,L)$ is a pseudo-list and satisfies $\mathbf{s}(u,x,\mathbf{nil})=\mathbf{nil}$ and $\mathbf{s}(u,x,(v{+}l+L))=(\mathbf{s}(u,x,v){+}\mathbf{s}(u,x,l)+\mathbf{s}(u,x,L))$.*

Proof. First one proves $\mathbf{s}(u,x,v)$ z-natural, for v z-natural, $z\neq x$ and $z\notin u$. Then, the statement of the lemma is proved by case analysis of L. ◀

▶ **Lemma 11** (Derived append rules). **1.** *$L@a$ is a continuation and satisfies: $\mathbf{nil}@a=(z)za$, if $z\notin a$; and $(u{+}l+L)@a=(z)z(u,l,L@a)$, if $z\notin u,l,L,a$.*
2. *$L@c$ is a continuation and satisfies: $\mathbf{nil}@c=c$; and $(u{+}l+L)@c=(z)z(u,l,L@c)$, if $z\notin u,l,L,c$.*
3. *$L@L'$ is a pseudo-list and satisfies: $\mathbf{nil}@L'=L'$ and $(u{+}l+L)@L'=(u{+}l+(L@L'))$.*

Proof. 1. (resp. 2.) Immediate by definition of $c@a$ (resp. $c@c$) 3. Particular case of 2. ◀

Notice that $L@c$ is the continuation obtained by replacing $\mathbf{nil}$ by c in L.

▶ **Lemma 12** (Derived closure rules). *The closure rules for pseudo-lists in Fig. 10 are derived closure rules of $\to_R$, for any R.*

Proof. The derivations are easy. ◀

Pseudo-lists allow a useful representation of continuations:

▶ **Lemma 13** (Unique decomposition). *Every continuation c can be written in a unique way as $L@(x)v$ with $\neg mla(x,v)$.*

Proof. Existence of decomposition: we prove that, for all $t\in\lambda\mathbf{Jm}$, there are L and v such that $\neg mla(x,v)$ and $(x)t=L@(x)v$. The proof is by induction on t. Uniqueness of decomposition: we prove that, for all $t\in\lambda\mathbf{Jm}$, if $(z)t=L@(x)v=L'@(y)v'$, with $\neg mla(x,v)$ and $\neg mla(y,v')$, then $L=L'$ and $(x)v=(y)v'$. The proof is by induction on t. ◀

▶ **Definition 14.** When we write $\langle u,l,L,(x)v\rangle$ we mean $(u,l,L@(x)v)$ with $\neg mla(x,v)$.

In the argument $\langle u,l,L,(x)v\rangle$ the continuation is analyzed into its unique decomposition as given by Lemma 13. Of course we can write a gm-application as $t\langle u,l,L,(x)v\rangle$.

▶ **Corollary 15** (Pseudo-lists). *A continuation c is a pseudo-list iff $c=L@(x)x$.*

Proof. $L@(x)x=L$ is a pseudo-list. Conversely, suppose c is a pseudo-list and $c=L@(x)v$ with $\neg mla(x,v)$. The only case of v where the replacement of $\mathbf{nil}$ by $(x)v$ in L yields a pseudo-list is $v=x$. ◀

▶ **Lemma 16** (Associativity of append). **1.** $(L@c)@c'=L@(c@c')$.

2. $(L@c)@a = L@(c@a)$.
3. $(L@a)@a' = L@(a@a')$. *(Compare with the third statement in Lemma 4.)*

Proof. Each by easy induction on L. Alternatively, the second (resp. third) statement follows from Lemma 2 and the first (resp. second) statement. ◀

Pseudo-lists can be used to give an handy alternative presentation of reduction rule π: $t\langle u, l, L, (x)v\rangle a \to t(u, l, L@(x)va)$.

Pseudo-lists also allow an alternative characterisation of the mapping μ for generalised multiary applications. For that, we need a flattening operation on pseudo-lists, denoted by $L^\flat$, and defined by: (i) $\mathbf{nil}^\flat := []$; (ii) $(u{+}l{+}L)^\flat := (u :: l)@L^\flat$. We also need μ extended to pseudo-lists homomorphically, that is: (i) $\mu(\mathbf{nil}) := \mathbf{nil}$; (ii) $\mu((u{+}l{+}L)) := (\mu(u){+}\mu(l){+}\mu(L))$.

▶ **Lemma 17.** $\mu(t\langle u, l, L, (x)v\rangle) = \mu t(\mu u, \mu l@(\mu L)^\flat, (x)\mu v)$.

Proof. By induction on L. The base case requires the fact that if $\neg mla(x, v)$, then also $\neg mla(x, \mu(v))$. The inductive case follows from the IH and uses associativity of the append operation on lists. ◀

3.2 Naturality

In this subsection we will introduce the concept of natural expression, and observe that this class of expressions is closed both for the reduction and the typing relations of $\lambda\mathbf{Jm}$, thus constituting the *natural subsystem* of $\lambda\mathbf{Jm}$.

▶ **Definition 18** (Natural and normal expressions)**.** An expression of $\lambda\mathbf{Jm}$ is *natural* if all continuations occurring in it are pseudo-lists. An expression of $\lambda\mathbf{Jm}$ is *normal* if it is both natural and cut-free.[6]

A normal expression corresponds to a typing derivation where the inference rule *Select* is constrained to be of the two forms described in item 1 of Lemma 9.

Natural expressions are generated by the following grammar:

$$\begin{array}{rrcl} \text{(natural proof terms)} & t, u, v & ::= & x \mid \lambda x.t \mid ta \\ \text{(natural gm-arguments)} & a & ::= & (u, l, L) \\ \text{(natural lists)} & l & ::= & u{::}l \mid [] \\ \text{(natural continuations)} & L & ::= & (x)v, \text{ with } v \text{ } x\text{-natural} \end{array} \tag{6}$$

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

When one coerces a natural argument $a = (u, l, L)$ to the natural continuation $(z)za$, with $z \notin a$, one obtains the natural pseudo-list $(u{+}l{+}L)$.

In view of Corollary 15, a natural application ta has the form $t\langle u, l, L, (x)x\rangle$; the last component is $\mathbf{nil}$ and so this representation does not give more information than $t(u, l, L)$.

The natural expressions of $\lambda\mathbf{Jm}$ are closed for typing in the following sense: in a typing derivation of a natural expression, every expression occurring in the derivation is natural itself. This is easily seen: the axioms of the typing system of $\lambda\mathbf{Jm}$ type natural expressions; in every other typing rule, the expressions in the premisses are subexpressions of the expression in the conclusion; and every subexpression of a natural expression is natural.

6 Natural proofs were called "normal proofs" in [6].

Figure 11 Some rules for the restricted closure

$$\frac{L \to L'}{L@c \to L'@c}\ (d) \qquad \frac{v \to v' \quad \neg mla(x,v)}{L@(x)v \to L@(x)v'}\ (e)$$

We now see the natural expressions of $\lambda\mathbf{Jm}$ are also closed for reduction. This is harder. The following lemma establishes that natural expressions are closed for the operations of substitution and append of gm-arguments.

▶ **Lemma 19.** **1.** *If u, E are natural expressions, then $\mathbf{s}(u,x,E)$ is a natural expression.*
2. *If a, a' are natural gm-arguments, then $a@a'$ is also a natural gm-argument.*
3. *If l, l' are natural lists, then $l@l'$ is also a natural list.*
4. *If L, L' are natural continuations, then $L@L'$ is also a natural continuation.*

Proof. Part 1 is proved by simultaneous induction on $E = v, a, l, c$. Part 2 follows from the fact that, given a natural continuation L and a natural gm-argument a', $L@a'$ is a natural continuation - and this is easily proved by induction on L. Parts 3 and 4 are proved by straightforward induction on l and L respectively. ◀

▶ **Definition 20.** A relation ρ on expressions of $\lambda\mathbf{Jm}$ *preserves naturality* if $E\rho E'$ and E natural implies E' natural.

We will see that $\to_R$ preserves naturality. For the reduction rules R this is done directly.

▶ **Lemma 21.** *For each $R \in \{\beta_1, \beta_2, \pi, \mu\}$, R preserves naturality.*

Proof. The cases $R = \beta_1$ and $R = \beta_2$ (resp. $R = \pi$, $R = \mu$) follow from Part 1 (resp. Part 2, Part 3) of Lemma 19. ◀

For the compatible closure $\to_R$, preservation of naturality is proved in an easier way with the help of a restricted notion of closure.

▶ **Definition 22** (Restricted closure)**.** The *restricted closure* of a relation on expressions of $\lambda\mathbf{Jm}$ is defined by replacing closure rule (IX) in Fig. 7 by the rules (a), (b) and (c) in Fig. 10, and the rules (d) and (e) in Fig. 11. If R is a reduction rule, the closure of R under the restricted closure is denoted $\rightsquigarrow_R$.

▶ **Lemma 23.** *If R preserves naturality, so does $\rightsquigarrow_R$.*

Proof. Suppose R preserves naturality. We prove by simultaneous induction four statements. The first three are: if $E \rightsquigarrow_R E'$ and E natural then E' natural, for terms, arguments and lists. The last is: if $L \rightsquigarrow_R L'$ and $L@c$ natural then $L'@c$ natural. ◀

We now must relate $\to_R$ and $\rightsquigarrow_R$. We will see that the two closures coincide for $R \in \{\beta_1, \beta_2, \pi\}$, but there are small differences for $R = \mu$, which, nonetheless, allow to conclude preservation of naturality by $\to_\mu$ from preservation of naturality by $\rightsquigarrow_\mu$.

▶ **Lemma 24** (Admissible closure rules of $\to_R$)**.** *Let $R \in \{\beta_1, \beta_2, \pi, \mu\}$. Closure rules (d) and (e) in Fig. 11 are admissible closure rules of $\to_R$.*

Proof. Case closure rule (d). One proves:
(i) if $t \to_R t'$, with t and t' x-natural, then $((x)t)@c \to_R ((x)t')@c$.

(ii) if $a \to_R a'$, with a and a' x-natural, then $((x)xa)@c \to_R ((x)xa')@c$.
(iii) if $c_1 \to_R c_1'$, with c_1 and c_1' pseudo-lists, then $c_1@c_2 \to_R c_1'@c_2$.

Case closure rule (e). In fact, one proves that the following are admissible closure rules of $\to_R$:

$$\frac{c \to c'}{t@c \to t@c'}\ (i) \qquad \frac{c_2 \to c_2'}{c_1@c_2 \to c_1@c_2'}\ (ii) \qquad \frac{v \to v'}{L@(x)v \to L@(x)v'}\ (iii)$$

◀

Putting together the previous lemma and Lemma 10, we conclude $\rightsquigarrow_R \subseteq \to_R$ for the reduction rules of $\lambda\mathbf{Jm}$. For the converse inclusion, to address the case $R = \mu$ we will need the followig new μ-rule on pseudo-lists:

$$(\mu_2) \quad (u{+}l + (u'{+}l' + L)) \to (u{+}(l@(u' :: l')) + L) \ .$$

▶ **Lemma 25** (Admissible closure rules of $\rightsquigarrow_R$). **1.** *For any reduction rule R, the following are admissible closure rules of $\rightsquigarrow_R$:*

$$\frac{L_1 \rightsquigarrow L_1'}{L_1@L_2 \rightsquigarrow L_1'@L_2}\ (i) \qquad \frac{L_2 \rightsquigarrow L_2'}{L_1@L_2 \rightsquigarrow L_1@L_2'}\ (ii) \qquad \frac{c \rightsquigarrow c'}{L@c \rightsquigarrow L@c'}\ (iii)$$

2. *Let $R \in \{\beta_1, \beta_2, \pi\}$. Closure rule (IX) of Fig. 7 is an admissible closure rule of $\rightsquigarrow_R$.*
3. *Let $R = \mu \cup \mu_2$. Closure rule (IX) of Fig. 7 is an admissible closure rule of $\rightsquigarrow_R$.*

Proof. The closure rule (iii) of part 1 is used in the proof of part 2. The new rule (μ_2) is needed to fix the base case of the inductive proof of part 3. ◀

▶ **Corollary 26.** *For each $R \in \{\beta_1, \beta_2, \pi, \mu\}$, $\to_R$ and $\rightsquigarrow_{R'}$ are the same relation, where $R' = R$ if $R \neq \mu$, and $R' = \mu \cup \mu_2$ otherwise.*

Proof. We had seen that $\rightsquigarrow_R \subseteq \to_R$. For $R \neq \mu$, part 2 of Lemma 25 completes the proof that $\to_R$ and $\rightsquigarrow_R$ are the same relation. In the case of μ, part 3 of Lemma 25 gives $\to_\mu \subseteq \rightsquigarrow_{R'}$, with $R' = \mu \cup \mu_2$. One still has to argue for $\rightsquigarrow_{R'} \subseteq \to_\mu$. Observe that μ_2 is a subset of the closure of $\rightsquigarrow_\mu$ under (IX). Hence $\rightsquigarrow_{R'}$ is a subset of the same closure. But such closure is a subset of $\to_\mu$, since $\rightsquigarrow_\mu \subseteq \to_\mu$ and $\to_\mu$ is closed under (IX). ◀

With this characterization of $\to_R$ in terms of the restricted closure, we can now show that the natural expressions of $\lambda\mathbf{Jm}$ are closed for reduction.

▶ **Theorem 27** (Preservation of naturality). *$\to_R$ preserves naturality, for each $R \in \{\beta_1, \beta_2, \pi, \mu\}$.*

Proof. By the previous corollary $\to_R = \rightsquigarrow_{R'}$, where $R' = R$ if $R \neq \mu$, and $R' = \mu \cup \mu_2$ otherwise. By Lemma 21, each R preserves naturality. It is clear that also μ_2 preserves naturality. So, in each case, the reduction rule R' preserves naturality; by Lemma 23, so does $\rightsquigarrow_{R'}$. ◀

Given that the natural expressions are closed for typing and reduction, we define:

▶ **Definition 28** (Natural subsystem). The *natural subsystem* of $\lambda\mathbf{Jm}$ is obtained by restriction to the natural expressions of the typing and reduction relations of $\lambda\mathbf{Jm}$. That is:

- given a natural term t, $\Gamma \vdash t : A$ in the natural subsystem if $\Gamma \vdash t : A$ in $\lambda\mathbf{Jm}$; and similarly for gm-arguments, lists, and continuations.
- given natural terms t, t', $t \to_R t'$ in the natural subsystem if $t \to_R t'$ in $\lambda\mathbf{Jm}$; and similarly for gm-arguments, lists, and continuations.

▶ **Corollary 29** (Confluence, SN, and uniqueness of normal form). *In the natural subsystem, $\beta\pi$- and $\beta\pi\mu$-reductions are confluent, and $\beta\pi\mu$-reduction is SN on typable expressions. In particular, every typable natural expression has a unique $\beta\pi$-nf, which is a normal expression.*

Proof. By the same properties of $\lambda\mathbf{Jm}$ (Theorem 5). ◀

3.3 Computational interpretation

The natural subsystem was defined by restricting the typing and reduction relations of $\lambda\mathbf{Jm}$. We now give a direct, self-contained, equivalent definition of the natural subsystem. The advantage is that the alternative definition has a transparent computational interpretation.

The key idea is to handle the abbreviations for pseudo-lists as if they were first-class expressions. In the resulting system, named $\lambda\mathbf{nm}$, pseudo-lists L behave properly as lists of non-empty lists of ordinary arguments; and arguments (u, l, L) may be seen as (and coerced to) non-empty pseudo-lists $(u{+}l{+}L)$. If we call lists of lists *multi-lists*, $\lambda\mathbf{nm}$ is then a *multi-multiary* λ-calculus, in the sense of a λ-calculus where functions are applied to multi-lists of arguments. The reduction rules of $\lambda\mathbf{nm}$ will confirm this interpretation.

Definition of $\lambda\mathbf{nm}$. The expressions of $\lambda\mathbf{nm}$ are the natural expressions, given by grammar (6). It is easy to prove that the same expressions are generated if, in the grammar, the class L is generated by $L ::= \mathbf{nil} \,|\, (u{+}l{+}L)$. These are the abbreviations (in the meta-language) we adopted to denote pseudo-lists - recall (5). Now we define typing and reduction rules for the natural expressions, alternative to those of Def. 28. The idea is to treat these abbreviations as if they were object syntax, and handle them with the derived rules contained in Lemmas 9, 10, 11, and 12, together with reduction rules that can be proved to be derived rules as well. Since the new system $\lambda\mathbf{nm}$ is built with derived rules of the natural subsystem given by Def. 28, the former will be immediately "contained" in the latter. We will check that the two systems are actually isomorphic.

▶ **Definition 30** (Typing system of $\lambda\mathbf{nm}$). The typing rules of $\lambda\mathbf{nm}$ are all the typing rules in Fig. 4 except *Select*, plus the typing rules in Fig. 9 (of course, in both cases with meta-variables t, a, u, l, c ranging over expressions of $\lambda\mathbf{nm}$).

Recall the four kinds of sequent of $\lambda\mathbf{Jm}$, displayed in (1). Observing the typing rules in Fig. 9 we conclude that, in $\lambda\mathbf{nm}$, sequents $\Gamma | C \vdash c : D$ of kind (iv) are such that D is a suffix of C; and sequents $\Gamma; A \supset B \vdash a : D$ of kind (ii) are such that D is a suffix of B.

The reduction rules of $\lambda\mathbf{nm}$ are given in Fig. 12. We let $\beta_1 := \beta_{11} \cup \beta_{12}$ and $\mu := \mu_1 \cup \mu_2$. Observe that reduction rule β_{12} can be derived as μ_1 followed by β_2. However, if we would omit β_{12}, the wanted 1-1 correspondence of reduction steps with the natural subsystem would be lost. The meta-operations used in the reduction rules of $\lambda\mathbf{nm}$ are as follows:

- $\mathbf{s}(u, x, E)$ denotes ordinary substitution on $\lambda\mathbf{nm}$ expression E, with $E = t, a, l, L$. In the case $E = L$, the operation is defined by the equations in Lemma 10.
- $L@L'$ denotes the append of two pseudo-lists of $\lambda\mathbf{nm}$ and is defined by the same equations as those in Lemma 11.
- $l@l'$ denotes the append of two lists of $\lambda\mathbf{nm}$ and is defined by the same equations as those in Definition 1.

▶ **Definition 31** (Compatible closure for $\lambda\mathbf{nm}$-expressions). A *compatible* relation on $\lambda\mathbf{nm}$-expressions is one closed for the closure rules in Fig. 7 except (IX), plus the closure rules in Fig. 10 (with meta-variables ranging over expressions of $\lambda\mathbf{nm}$). The *compatible closure* of a rule R of $\lambda\mathbf{nm}$, denoted $\to_R$, is the smallest compatible relation containing R.

Figure 12 The reduction rules of the multi-multiary λ-calculus $\lambda\mathbf{nm}$

$$
\begin{array}{rrcl}
(\beta_{11}) & (\lambda x.t)(u,[],\mathbf{nil}) & \to & \mathbf{s}(u,x,t) \\
(\beta_{12}) & (\lambda x.t)(u,[],(u'{+}l+L)) & \to & \mathbf{s}(u,x,t)(u',l,L) \\
(\beta_{2}) & (\lambda x.t)(u,u'::l,L) & \to & \mathbf{s}(u,x,t)(u',l,L) \\
(\pi) & t(u,l,L)(u',l',L') & \to & t(u,l,L@(u'{+}l'+L')) \\
(\mu_1) & (u,l,(u'{+}l'+L)) & \to & (u,l@(u'::l'),L) \\
(\mu_2) & (u{+}l+(u'{+}l'+L)) & \to & (u{+}l@(u'::l')+L)
\end{array}
$$

Having completed the definition of the system $\lambda\mathbf{nm}$, we pause to observe its **computational interpretation**: $\lambda\mathbf{nm}$ *is a lambda-calculus where functions are applied to non-empty multi-lists, where a multi-list is a list of non-empty lists of arguments. The reduction rules have a transparent meaning in terms of these multi-lists:* β*-rules pass to the applied function the first element of the first list of arguments in the multi-list, while* π *and* μ *append and flatten multi-lists of arguments, respectively.*

▶ **Proposition 32** (Natural subsystem $\cong \lambda\mathbf{nm}$)**.** **1.** $\Gamma \vdash t : A$ *in the natural subsystem iff* $\Gamma \vdash t : A$ *in* $\lambda\mathbf{nm}$*. Similarly for gm-arguments, lists and continuations.*

2. *Let* $R \in \{\beta_1, \beta_2, \pi, \mu\}$*.* $t \to_R t'$ *in the natural subsystem iff* $t \to_R t'$ *in* $\lambda\mathbf{nm}$*. Similarly for gm-arguments, lists and continuations.*

Proof. 1. There are four "if" statements (one for each $E = t, a, l, L$) proved by simultaneous induction. The only interesting point is that the typing rules in Fig. 9 are derived typing rules of the natural subsystem. Similarly, there are four "only if" statements, proved by simultaneous induction.

2. The "if" statement for $E = t$ is proved together with similar statements for $E = a, l, L$, by simultaneous induction on $E \to_R E'$ in $\lambda\mathbf{nm}$. The "only if" statement for $E = t$ is proved together with similar statements for $E = a, l, L$, by simultaneous induction on $E \to_R E'$ in the natural subsystem.

◀

The natural subsystem of $\lambda\mathbf{Jm}$ benefits largely from this isomorphism. The presentation of its typing and reduction rules as in Def. 30 and Fig. 12 is much more perspicuous than through Def. 28: think of the sequent invariants noted after Def. 30, or the computational interpretation of $\lambda\mathbf{nm}$, that the natural subsystem inherits. The isomorphism lets us see that the natural subsystem corresponds to a multi-multiary λ-calculus, where the generality feature is reduced to a mechanism to form lists of lists of arguments for functional application.

3.4 Naturality and focusedness

Natural proofs are a generalization of focused proofs (in the sense of LJT). We will show this both for the computation-as-cut-elimination and computation-as-proof-search paradigms. In the former case, we show the relationship between the calculi $\lambda\mathbf{nm}$ and $\lambda\mathbf{m}$; in the latter, we explain how normal(=natural and cut-free) proofs can be searched by a procedure that is a relaxed form of focusing.

Recall that the map μ calculates the unique μ-nf of a $\lambda\mathbf{Jm}$ expression. Its restriction to $\lambda\mathbf{nm}$ has a recursive description in which the single interesting clause is given by $\mu(t(u,l,L)) = \mu t(\mu u, \mu l@(\mu L)^\flat)$, thanks to Lemma 17. So we see μ maps natural proofs to focused proofs;

Figure 13 Proof system for normal proofs (in the atomized system, D is atomic)

$$\boxed{\mathrm{I}} \qquad \frac{}{x:D,\Gamma\vdash x:D}\ Axiom \qquad \frac{x:A,\Gamma\vdash t:B}{\Gamma\vdash \lambda x.t:A\supset B}\ Right$$

$$\boxed{\mathrm{II}} \qquad \frac{\Gamma;A\supset B\vdash a:D}{\Gamma\vdash xa:D}\ Unselect\ ((x:A\supset B)\in\Gamma)$$

$$\boxed{\mathrm{III}} \qquad \frac{\Gamma\vdash u:A \quad \Gamma;B\vdash l:C \quad \Gamma|C\vdash L:D}{\Gamma;A\supset B\vdash (u,l,L):D}\ Outer\text{-}multi\text{-}Lft$$

$$\frac{}{\Gamma|D\vdash \mathbf{nil}:D}\ Axm \qquad \frac{\Gamma\vdash u:A \quad \Gamma;B\vdash l:C \quad \Gamma|C\vdash L:D}{\Gamma|A\supset B\vdash (u{+}l+L):D}\ Inner\text{-}multi\text{-}Lft$$

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

the case $t = x$ also gives that μ maps normal proofs to cut-free, focused proofs[7]. The latter is also a consequence of the fact that μ-reduction in $\lambda\mathbf{nm}$ preserves cut-freeness, a particular case of Lemma 6.

▶ **Theorem 33** (Preservation of reduction on natural proofs by μ)**.**

1. *If* $t \to_\beta t'$ *in* $\lambda\mathbf{nm}$ *then* $\mu(t) \to_\beta \mu(t')$ *in* $\lambda\mathbf{m}$.
2. *If* $t \to_\pi t'$ *in* $\lambda\mathbf{nm}$ *then* $\mu(t) \to_{\pi'} \mu(t')$ *in* $\lambda\mathbf{m}$..

Proof. By Theorem 7 and the following two facts: (i) $\lambda\mathbf{m}$ is closed for β-reduction; (ii) $\to_{\pi'}$ in $\lambda\mathbf{m}$ is the same as $\to_\pi$ followed by μ-reduction to μ-nf in $\lambda\mathbf{nm}$. ◀

This theorem says μ is a morphism between the natural and the focused subsystems of $\lambda\mathbf{Jm}$.

In Fig. 13 we recapitulate the typing system for normal expressions[8]. The rule $Leftm$ has been renamed to $outer - multi - Lft$ to reflect its resemblance with $multi - Lft$, which in turn has been renamed to $inner - multi - Lft$. Cut inferences are restricted to the $Unselect$ form, which behaves as a focusing inference.

We will now see in detail how the good properties enjoyed by focused proof systems (invertibility, completeness w.r.t. provability, disciplined proof search) apply to the proof system for normal proofs.

One observation used several times below is that *weakening* is an admissible rule for the various forms of sequents in the proof system for normal proofs. Let us look first into invertibility of rules $multi - Lft$, which is not immediate because of the foreign formula C.

---

7 Note that mapping μ restricted to the class of *unary* normal expressions is a 1-1 correspondence with cut-free, focused proofs (which are the cut-free LJT proofs, or the cut-free $\overline{\lambda}$-terms, as already shown in [4] - but there the name used for the mapping is $\overline{\varphi}$).

8 The division into groups of rules will be useful later.

▶ **Proposition 34** (Invertibility of *multi-Lft* rules)**.** *If* $\Gamma;A \supset B \vdash a : D$ *or* $\Gamma|A \supset B \vdash L : D$ *and* D *is an atomic formula, then there exists* u_0 *s.t.* $\Gamma \vdash u_0 : A$*, and for all* C *suffix of* B*, there exist* l_0, L_0 *s.t.* $\Gamma;B \vdash l_0 : C$*, and* $\Gamma|C \vdash L_0 : D$*.*

Proof. Case $\Gamma;A \supset B \vdash a : D$ with $a = (u_0, l_0, L_0)$, we must have $\Gamma \vdash u_0 : A$, and, for some C_0, $\Gamma;B \vdash l_0 : C_0$, and $\Gamma|C_0 \vdash L_0 : D$. The result follows then with the help of the following *suffix lemma*: for D the atomic suffix of B, if, for some C_0, l_0, L_0, $\Gamma;B \vdash l_0 : C_0$ and $\Gamma|C_0 \vdash L_0 : D$, then, for all C suffix of B there exist l, L s.t. $\Gamma;B \vdash l : C$ and $\Gamma|C \vdash L : D$. (This lemma follows by induction on B.) Case $\Gamma|A \supset B \vdash L : D$, as D is atomic, the derivation cannot be solely an axiom Axm. So, we must have $L = (u_0 + l_0 + L_0)$, and proceed as in the previous case. ◀

Invertibility of the *multi-Lft* rules is guaranteed only if the RHS formula of the conclusion is atomic, but this is in line with LJT, where typically proof search imposes atomic RHS in the conclusion of Lft inferences (see e.g. [2] for a system corresponding to LJT with this atomic restriction). Next, we consider a restriction of the proof system for normal proofs, for which invertibility of the *multi-Lft* rules holds and a focused proof search discipline can be followed.

▶ **Definition 35** (Atomized normal system)**.** The *atomized system for normal proofs* is the system obtained from the proof system for normal proofs in Fig. 13 by imposing that at the rules $Axiom$ and $Unselect$ the RHS formula is atomic. We denote these restricted versions of the rules by $Axiom_{atom}$ and $Unselect_{atom}$. We write $\vdash_{atom}$, instead of $\vdash$, to mean that a sequent has a derivation in the atomized system.

Before we describe proof search in the atomized system, we will show that nothing is lost in the atomized system regarding provability of sequents $\Gamma \vdash t : A$.

▶ **Definition 36** (η-expansion)**.** The η-expansion rules for normal expressions are

$$y \to \lambda x.y(x, [], \mathbf{nil}) \qquad y(u, l, L) \to \lambda x.y(u, l, \eta exp_x L)$$

where $x \neq y$ and $x \notin u, l, L$, and $\eta exp_x L$ is defined by: $\eta exp_x \mathbf{nil} = (x + [] + \mathbf{nil})$ and $\eta exp_x(u + l + L) = (u + l + \eta exp_x L)$. The compatible closure of these rules is denoted $\to_{\eta exp}$.

▶ **Lemma 37** (Admissibility of $Axiom$ and $Unselect$)**.** *For any* A*:*
1. *There exists* t *s.t.* $x \to^*_{\eta exp} t$ *and* $x : A, \Gamma \vdash_{atom} t : A$*.*
2. *If* $\Gamma;B \supset C \vdash_{atom} a : A$ *and* $x : B \supset C \in \Gamma$*, there exists* t *s.t.* $xa \to^*_{\eta exp} t$ *and* $\Gamma \vdash_{atom} t : A$*.*
3. *If* $\Gamma|C \vdash_{atom} L : A \supset B$*, there exists* L' *s.t.* $\eta exp_y L \to^*_{\eta exp} L'$ *and* $y : A, \Gamma|C \vdash_{atom} L' : B$*.*

Proof. Proved simultaneously by induction on A. ◀

▶ **Theorem 38** (Completeness of the atomized system)**.** *If* $\Gamma \vdash t : A$*, then there exists* t' *s.t.* $t \to^*_{\eta exp} t'$ *and* $\Gamma \vdash_{atom} t' : A$*. Similarly for gm-arguments, lists, and continuations.*

Proof. The proof of the four statements is done by simultaneous induction. All cases follow routinely, except for the cases $t = x$ and $t = xa$. The case $t = x$ follows by 1. of the lemma before, whereas the case $t = xa$ is by IH and 2. of the lemma before. ◀

Proof search in the atomized system. Proof search in the atomized system will find a derivation of $\Gamma \vdash t : A$, if one exists, following a disciplined alternation between *asynchronous* and *synchronous* phases which we now explain. In this explanation, *bottom-up* application of inference rules is meant; we also refer to the groups of rules in Fig. 13.

The asynchronous phase searches for proofs of sequents $\Gamma \vdash t : A$ by applying rules of group I. Rule *Right* decomposes implications until an atomic formula is reached. If this atom is in the l.h.s. of the sequent, rule $Axiom_{atom}$ ends the search with success. Otherwise, the only rule in group II picks a formula from the context, and a synchronous phase starts.

The synchronous phase searches for proofs of sequents $\Gamma;A \supset B \vdash a:D$ or $\Gamma|C \vdash L:D$, by applying rules of group III. This phase consists of a chain of $multi-Lft$ inferences, starting with an $Outer-multi-Lft$ inference, continuing with $n \geq 0$ $Inner-multi-Lft$ inferences, and ending with an application of Axm when successful.

Each application of a $multi-Lft$ inference (either an outer or an inner one) transforms the distinguished formula $A \supset B$ in the l.h.s. of the sequent to be proved into a formula C, which is not necessarily the immediate positive subformula B, but rather some suffix of B which has to be chosen (provability is not affected by this choice - recall Proposition 34), triggering a subprocess of proof search for $\Gamma \vdash u:A$, and a subsidiary search for $\Gamma;B \vdash l:C$. The search for $\Gamma;B \vdash l:C$ is done by *focusing* on B, through application of rules in group IV.

So focusing is a subsidiary process of the synchronous phase. In fact, we may say the chain of $n+1$ $multi-Lft$ inferences that constitutes the synchronous phase that started with sequent $\Gamma;A \supset B \vdash a:D$ breaks into a succession of $n+1$ focusing proofs (that can be conducted independently and in parallel) what in a focused system like LJT or $\lambda\mathbf{m}$ would rather be a single focusing proof leading from $A \supset B$ to D.[9]

4 Permutability in the sequent calculus $\lambda\mathbf{Jm}$

In this section we study permutative conversions in $\lambda\mathbf{Jm}$ such that the proofs irreducible by such conversions are the natural proofs studied in the previous section. This justifies our description of natural proofs as "permutation-free". Our approach to permutative conversions is the simplest one: we introduce a map γ that translates any $\lambda\mathbf{Jm}$ proof into a natural one (and leaves natural proofs invariant); in addition, it maps cut-free proofs to normal ones, as required [4]. Map γ, studied in the second subsection, is defined in terms of a special substitution operator over natural proofs, which is introduced in the first subsection. Such an operator is an essential ingredient of the computational process involved in γ. In the third subsection, we prove that permutative conversion to natural form commutes with cut-elimination. Hence, the two immediate senses for the concept of *normalization*, either permutative conversion of cut-free proofs to normal form, or cut-elimination in the natural subsystem, are coherent and have a common generalization to $\lambda\mathbf{Jm}$. In the final fourth subsection we systematize the internal structure of $\lambda\mathbf{Jm}$ with the help of γ.

4.1 Special substitution

The special substitution operation on $\lambda\mathbf{nm}$ that we will introduce now is the key element in the permutative conversion of $\lambda\mathbf{Jm}$ expressions to natural form.

▶ **Definition 39** (Special substitution of $\lambda\mathbf{nm}$)**.** Given $t \in \lambda\mathbf{nm}$, we define $\mathbb{S}(t,x,u)$, $\mathbb{S}(t,x,a)$, $\mathbb{S}(t,x,l)$ and $\mathbb{S}(t,x,L)$ (for $u,a,l,L \in \lambda\mathbf{nm}$) by simultaneous recursion:

$$\begin{array}{ll}
\mathbb{S}(t,x,x) = t & \mathbb{S}(t,x,(u,l,L)) = (\mathbb{S}(t,x,u),\mathbb{S}(t,x,l),\mathbb{S}(t,x,L)) \\
\mathbb{S}(t,x,y) = y \;\text{ if } x \neq y & \mathbb{S}(t,x,[]) = [] \\
\mathbb{S}(t,x,\lambda y.v) = \lambda y.\mathbb{S}(t,x,v) & \mathbb{S}(t,x,(u::l)) = \mathbb{S}(t,x,u)::\mathbb{S}(t,x,l) \\
\mathbb{S}(t,x,xa) = t@\mathbb{S}(t,x,a) & \mathbb{S}(t,x,\mathbf{nil}) = \mathbf{nil} \\
\mathbb{S}(t,x,t'a) = \mathbb{S}(t,x,t')\mathbb{S}(t,x,a) \;\text{ if } t' \neq x & \mathbb{S}(t,x,(u+l+L)) = (\mathbb{S}(t,x,u)+\mathbb{S}(t,x,l)+\mathbb{S}(t,x,L))
\end{array}$$

[9] This has nothing to do with multifocusing, where the focus contains simultaneously several formulas.

The difference to ordinary substitution is seen in the fourth clause, with $t@\mathbb{S}(t,x,a)$ instead of $t\mathbb{S}(t,x,a)$. The precise relation between ordinary and special substitution is:

▶ **Lemma 40** (Subst. vs special subst.). $\mathbf{s}(u,x,E) \to_\pi^* \mathbb{S}(u,x,E)$*, for all* $E \in \lambda\mathbf{nm}$.

Proof. By simultaneous induction on $E = v, a, l, L$. ◀

▶ **Lemma 41** (Typing of special substitution). *The following rules are admissible in* $\lambda\mathbf{nm}$.

$$\frac{\Gamma \vdash t : A \quad x : A, \Gamma \vdash u : B}{\Gamma \vdash \mathbb{S}(t,x,u) : B} \qquad \frac{\Gamma \vdash t : A \quad x : A, \Gamma ; B \supset C \vdash a : D}{\Gamma ; B \supset C \vdash \mathbb{S}(t,x,a) : D}$$

$$\frac{\Gamma \vdash t : A \quad x : A, \Gamma ; B \supset C \vdash l : D}{\Gamma ; B \supset C \vdash \mathbb{S}(t,x,l) : D} \qquad \frac{\Gamma \vdash t : A \quad x : A, \Gamma | B \supset C \vdash L : D}{\Gamma | B \supset C \vdash \mathbb{S}(t,x,L) : D}$$

Proof. By simultaneous induction on u, a, l, L. The case $u = xa$ uses first the IH to type $\mathbb{S}(t,x,a)$, and then uses admissibility of the first rule of Fig. 5 to type $t@\mathbb{S}(t,x,a)$. ◀

From this proof we extract the *operation on typing/logical derivation of* $\lambda\mathbf{nm}$ *whose term representation is* $\mathbb{S}(t,x,u)$, performing the elimination of the cut which types this substitution. In general, such operation performs the permutation to the right as long as the repetition of the cut formula permits, supplemented in the exceptional case $u = xa$ by the operation associated with the operation $t@a'$ (recall discussion after Lemma 3).

▶ **Lemma 42** (Substitution Lemma). *Let* $t, u \in \lambda\mathbf{nm}$*,* $x \neq y$*, and* $y \notin u$*. For all* $E \in \lambda\mathbf{nm}$*:*
1. $\mathbf{s}(u,x,\mathbb{S}(t,y,E)) \to_\pi^* \mathbb{S}(\mathbf{s}(u,x,t),y,\mathbf{s}(u,x,E))$*;*
2. $\mathbb{S}(u,x,\mathbb{S}(t,y,E)) = \mathbb{S}(\mathbb{S}(u,x,t),y,\mathbb{S}(u,x,E))$*;*
3. $\mathbf{s}(\mathbb{S}(u,x,t),y,\mathbb{S}(u,x,E)) \to_\pi^* \mathbb{S}(u,x,\mathbf{s}(t,y,E))$.

Proof. By simultaneous induction on $E = v, a, l, L$. ◀

4.2 Permutative conversion to natural form

Now we introduce the map that realises conversion to natural form, and, in particular, show that it preserves typing, leaves invariant natural expressions, and preserves reduction.

▶ **Definition 43** (Conversion to natural form map). For $t, a, c, l \in \lambda\mathbf{Jm}$ and $t' \in \lambda\mathbf{nm}$, we define $\gamma(t)$, $\gamma(t', a)$, $\gamma(t', c)$, and $\gamma(l)$, by simultaneous recursion on t, a, c, and l:

$$\begin{array}{rclrcl}
\gamma(x) & = & x & \gamma(t',(u,l,c)) & = & \gamma(t'(\gamma u,\gamma l,\mathbf{nil}),c) \\
\gamma(\lambda x.t) & = & \lambda x.\gamma(t) & \gamma(t',(x)v) & = & \mathbb{S}(t',x,\gamma v) \\
\gamma(ta) & = & \gamma(\gamma t, a) & \gamma([]) & = & [] \\
 & & & \gamma(u :: l) & = & \gamma(u) :: \gamma(l)
\end{array}$$

This is summarized in the following equation:

$$\gamma(t(u,l,(x)v)) = \mathbb{S}(\gamma t(\gamma u,\gamma l,\mathbf{nil}),x,\gamma v) \tag{7}$$

▶ **Proposition 44** (Preservation of typing by γ). *The following typing rules are admissible (where* $\vdash$ *and* $\vdash'$ *denote derivability in* $\lambda\mathbf{Jm}$ *and* $\lambda\mathbf{nm}$ *resp., and so* $t, a, l, c \in \lambda\mathbf{Jm}$ *and* $t' \in \lambda\mathbf{nm}$*).*

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash' \gamma t : A} \qquad \frac{\Gamma \vdash' t' : A \supset B \quad \Gamma ; A \supset B \vdash a : C}{\Gamma \vdash' \gamma(t',a) : C} \qquad \frac{\Gamma ; A \vdash l : B}{\Gamma ; A \vdash' \gamma l : B} \qquad \frac{\Gamma \vdash' t' : A \qquad \Gamma | A \vdash c : B}{\Gamma \vdash' \gamma(t',c) : B}$$

Proof. By simultaneous induction on t, a, l, c. The case $a = (u', l', c')$ needs to first show $\gamma(t')(\gamma(u'), \gamma(l'), \mathbf{nil})$ is typable, using the IH relative to u' and l', and then use the IH relative to c'. The case $c = (x)v$ needs the typing rule of the special substitution on terms (Lemma 41). The other cases are routine. ◀

From this proof we extract the *operation on typing/logical derivation of* $\lambda\mathbf{Jm}$ *associated with* γ: it is an innermost-outermost application of the operation associated with transformation (7), and the latter, in turn, is the operation on derivations of $\lambda\mathbf{nm}$ associated with special substitution (see discussion after Lemma 41), applied after the transformation of the given sub-derivations (represented by t, u, l, v).

γ is extended to pseudo-lists:

$$\gamma(\mathbf{nil}) := \mathbf{nil} \qquad\qquad \gamma((u{+}l\,{+}L)) := (\gamma(u){+}\gamma(l)\,{+}\gamma(L)) \ , \tag{8}$$

▶ **Proposition 45** (Invariance of natural expressions under γ)**.** *For* $E = t, l, L \in \lambda\mathbf{nm}$*,* $\gamma E = E$*.*

Proof. By simultaneous induction on t, l, L. The interesting case is where $t = t'(u', l', L')$, which follows by IH and the fact $\gamma(t_0(u, l_0, L_0@(x)v)) = \mathbb{S}(\gamma t_0(\gamma u, \gamma l_0, \gamma L_0), x, \gamma v)$, for any $t_0, u, l_0, L_0, v \in \lambda\mathbf{Jm}$, which, in turn, uses the following auxiliary result: given $t', u', l', L' \in \lambda\mathbf{nm}$, $\gamma(t'(u', l', L'), L@(x)v) = \mathbb{S}(t'(u', l', L'@\gamma L), x, \gamma v)$ (proved by induction on L). ◀

This means that, if we want to see γ as defining the naive, long-step reduction rule $E \to \gamma(E)$, we have to require the redex E not to be normal, and so the normal expressions are the irreducible expressions for this rule.

The following result says γ sends cut-free proofs to normal proofs.

▶ **Lemma 46** (γ preserves cut-freeness)**.** *If* t *is a* $\beta\pi$*-nf of* $\lambda\mathbf{Jm}$*,* $\gamma(t)$ *is a* $\beta\pi$*-nf of* $\lambda\mathbf{nm}$*.*

Proof. Proved together with analogue statements for gm-arguments, lists and pseudo-lists. The case $t = xa$ requires an auxiliary result about preservation of $\beta\pi$-nfs by substitutions of the form $\mathbb{S}(x(u, l, L), y, t)$. ◀

▶ **Theorem 47** (Preservation of reduction by conversion to natural form)**.**

1. *If* $t \to_\beta t'$ *in* $\lambda\mathbf{Jm}$ *then* $\gamma(t) =_{\beta\pi} \gamma(t')$ *in* $\lambda\mathbf{nm}$*.*
2. *If* $t \to_R t'$ *in* $\lambda\mathbf{Jm}$ *then* $\gamma(t) \to_R^* \gamma(t')$ *in* $\lambda\mathbf{nm}$*, for* $R \in \{\pi, \mu\}$*.*

Proof. We use the inductive characterisation of reduction in $\lambda\mathbf{Jm}$ given by Corollary 26. Notice $=_{\beta\pi}$ in statement 1. ◀

4.3 Normalisation

We have so far two processes of obtaining a normal(=natural and cut-free) proof: either by cut-elimination on a natural proof (as natural proofs are closed for cut-elimination, recall Theorem 27), or by permutative conversion of a cut-free proof (as γ preserves cut-freeness, recall Lemma 46). We may call such processes *normalization* processes. The question is whether there is a normalization procedure defined on arbitrary $\lambda\mathbf{Jm}$ proofs which generalizes both these two processes. The answer is positive, due to the following result.

▶ **Theorem 48** (Commutation between cut-elim. and conversion to natural form)**.** *For all typable* $t \in \lambda\mathbf{Jm}$*,* $\gamma(\downarrow_{\beta\pi}(t)) = \downarrow_{\beta\pi}(\gamma(t))$*.*

Proof. Firstly observe that all the required nfs exist since the starting terms are typable and the map γ preserves typing. By Theorem 47, $\gamma(t) =_{\beta\pi} \gamma(\downarrow_{\beta\pi}(t))$. By Lemma 46, $\gamma(\downarrow_{\beta\pi}(t))$ is a $\beta\pi$-nf. Hence, by confluence of $\to_{\beta\pi}$ in $\lambda\mathbf{nm}$, $\gamma(t) \to_{\beta\pi}^* \gamma(\downarrow_{\beta\pi}(t))$. ◀

▶ **Definition 49** (Normalisation map). $\rho(E) := \gamma(\downarrow_{\beta\pi}(E))$, for all typed $\lambda\mathbf{Jm}$ expression E.

If E is cut-free, then $\rho(E) = \gamma(E)$, which is the permutative conversion of E; if E is natural, then $\rho(E) = \downarrow_{\beta\pi}(\gamma(E)) = \downarrow_{\beta\pi}(E)$, which is the result of cut elimination from E in $\lambda\mathbf{nm}$.

4.4 The taming of "bureaucracy"

The permutative conversion γ and the reduction process μ are the "bureaucratic" processes of $\lambda\mathbf{Jm}$, as opposed to $\beta\pi$-reduction, which represents cut-elimination. We are now in position to converge to a systematic picture of the internal organization of $\lambda\mathbf{Jm}$, fulfilling the promise made in the introduction of linking Figs. 3 and 2. The final result we want to achieve is in Fig. 14.

Recall μ preserves "γ-normality", as the range of γ is $\lambda\mathbf{nm}$, which is closed for μ.

▶ **Theorem 50** (Commutation between μ and γ). $\gamma(t) \to_\mu^* \gamma(\mu t) \to_\mu^* \mu(\gamma t)$.

Proof. First observe that part 3 of Theorem 47 gives: if $t \to_\mu^* t'$ in $\lambda\mathbf{Jm}$, then $\gamma(t) \to_\mu^* \gamma(t')$ in $\lambda\mathbf{nm}$. From this, together with $t \to_\mu^* \mu t$, we get $\gamma(t) \to_\mu^* \gamma(\mu t)$. From this, together with $\gamma t \to_\mu^* \mu(\gamma t)$, we conclude that $\mu(\gamma t)$ is the μ-nf of $\gamma(\mu t)$. ◀

In general, $\gamma(\mu t) = \mu(\gamma t)$ does not hold, as γ does not preserve μ-normality. By the theorem, we only have $\mu(\gamma(\mu t)) = \mu(\gamma t)$. Therefore, we define the combination of γ and μ to be $\mu \circ \gamma$, denoted γ'. This defines a map from $\lambda\mathbf{Jm}$ to $\lambda\mathbf{m}$, sending an arbitrary proof to a focused one. In this sense, this map may be called a *focalization* process.

▶ **Theorem 51** (Commutation). *Every face of the two cubes in Fig. 14 commutes.*

Proof. The equality $\mu(\gamma(\mu t)) = \mu(\gamma t)$ is the commutativity of face NL in Fig. 14. We now argue the commutativity of every other face, with faces named according to the explanation given below the figure. Face SL: particular case of face NL, as γ and μ preserve cut-freeness. Face BW: Theorem 48. Face BE: $\mu(\downarrow_{\beta\pi}(t))$ and $\downarrow_{\beta'\pi'}(\mu t)$ are $\beta\pi\mu$-nfs of t, hence are the same term by confluence of $\beta\pi\mu$-reduction. Face B: by the isomorphism between $\lambda\mathbf{J}$ and the flat subsystem [8], which links β, π with β', π', respectively. Face F: by the isomorphism between $\lambda\mathbf{n}$ and $\lambda\mathbf{m}$ [6]. Face N: the unary particular case of face NL. This may be seen as extending to γ, γ' the isomorphism between $\lambda\mathbf{J}$ and the flat subsystem. Face S: the unary particular case of face SL, or particular case of face N, as γ' and μ preserve cut-freeness. Face E: the unary particular case of face BW. Face FE: by the isomorphism between $\lambda\mathbf{J}$ and the flat subsystem, which means that face E is isomorphic to face FE. That the "diagonal" map of face FE is ρ' (*i.e.* $\mu \circ \rho$) follows from the commutativity of faces SL and BE. ◀

To conclude, Fig. 14 says that $\lambda\mathbf{Jm}$ consists of two levels linked by cut-elimination, each level organized by the "bureaucratic" conversions γ and μ - and we see that the organization is quite tidy. Above the line (a) the maps are "morphisms" of λ-calculi: in addition to the isomorphisms that cross the line (b), recall the properties of μ and γ, namely Theorems 7, 33, and 47. The permutation-free fragment $\lambda\mathbf{nm}$ and its sub-fragment $\lambda\mathbf{m}$ have clear computational meaning: (multi-)multiary λ-calculi whose normal forms can be found by a (relaxed) focusing proof-search strategy.

5 Final remarks

This paper is a study of the computational interpretation of the sequent calculus that deals with the permutability phenomenon, hence distinguished either from the approaches that avoid permutability altogether by staying in some permutation-free fragment, or from approaches that simplify the problem by staying in the cut-free fragment or in some fragment that is indistinguishable from natural deduction. Our contribution is two-staged: first we studied the permutation-free fragment, then we mediated the full and the permutation-free systems by means of permutative conversions. In the permutation-free level, the novelty is in the computational interpretation: the "multi-multiary" λ-calculus is the transparent Curry-Howard interpretation of natural proofs, and normal proofs can be searched by a new, relaxed form of focusing. Beyond the permutation-free level, the novelty is in the permutative conversion γ and how, with its help, a complete picture of the internal structure of the sequent calculus $\lambda\mathbf{Jm}$ is achieved, as seen in Fig. 14: two halves mediated by cut-elimination and organized by the "bureaucracy" conversions γ and μ. To measure the progress achieved, this picture should be compared with the wisdom established long ago [4] for the cut-free setting and depicted in Fig. 1.

On the way to such a complete picture, numerous side contributions were made, including: the technicalities involving append operators and pseudo-lists that permitted a smooth handling of natural proofs; the always surprising richness of "abbreviation" conversion μ, this time promoted to a morphism between the natural and the focused fragments; the concept of special substitution on natural proofs, which is the computational process behind permutative conversion γ; and the indirect contributions to ΛJ *qua* unary fragment $\lambda\mathbf{J}$.

Among the previous papers on $\lambda\mathbf{Jm}$ [8, 6, 9], the present is closer to [6] in its attempt to refine the naive view that $\lambda\mathbf{Jm}$ is obtained from the λ-calculus by the addition of the multiarity and generality dimensions. But the purpose of [6] was to catalogue classes of normal forms (and rewriting systems giving rise to them). Curiously, the class of normal proofs studied here escaped that catalogue; and even if we find there the statement that natural proofs form a subsystem, no computational interpretation was developed. In addition, a conversion γ was proposed in [6], but it employed ordinary substitution, which does not preserve cut-freeness, hence does not preserve normality. In the present paper, we backtrack, employ special substitution in the definition of γ, and start afresh.

It is important to notice that the purpose of this paper is just to identify computational meaning: to assess whether that meaning is useful in practice is out of scope. For instance, we are happy to pin down the relaxation of focusing that constitutes the proof-search procedure for normal proofs. Such variation on focusing seems to be new, and seems useful in practice, allowing some parallelism in the synchronous phase - but we do not say more. Also the Curry-Howard interpretation of the natural subsystem (a λ-calculus where functions are applied to a vector of vectors of arguments) is perhaps not exciting, but is transparent and illuminating: it means that, in the natural fragment, the generality feature is reduced to a second-level vectorization mechanism.

Only space limitation prevented us from developing the study of other reduction procedures inside $\lambda\mathbf{Jm}$ like focalization (captured by the combination of γ and μ) and its combination with cut-elimination or normalization. On the other hand, further work is needed if one is interested in rewriting systems of permutative conversions, like those in [4, 20]. The present concept of special substitution gives a hint of what global operation the local rewrite steps should be calculating; but a generalization of that operation from natural proofs to arbitrary proofs is required, and this is on-going work.

References

1 T. Brock-Nannestad, N. Guenot, and D. Gustafsson. Computation in focused intuitionistic logic. In *Proc. of PPDP'15*, pages 43–54. ACM, 2015.

2 I. Cervesato and F. Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.

3 P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of ICFP'00*, pages 233–243. IEEE, 2000.

4 R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212:141–155, 1999.

5 J. Espírito Santo. Curry-Howard for sequent calculus at last! In *Proc. of TLCA'15*, volume 38 of *LIPIcs*, pages 165–179. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

6 J. Espírito Santo, M. J. Frade, and L. Pinto. Structural proof theory as rewriting. In *Proc. of RTA'06*, volume 4098 of *Lecture Notes in Computer Science*, pages 197–211. Springer, 2006.

7 J. Espírito Santo, R. Matthes, and L. Pinto. Continuation-passing style and strong normalisation for intuitionistic sequent calculi. *Logical Methods in Computer Science*, 5(2), 2009.

8 J. Espírito Santo and L. Pinto. Confluence and strong normalisation of the generalised multiary λ-calculus. In *Revised selected papers from TYPES'03*, volume 3085 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2004.

9 J. Espírito Santo and L. Pinto. A calculus of multiary sequent terms. *ACM Trans. Comput. Log.*, 12(3):22, 2011.

10 J-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

11 J-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge Univ. Press, 1989.

12 H. Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In *Proc. of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1995.

13 F. Joachimski and R. Matthes. Standardization and confluence for a lambda calculus with generalized applications. In *Proc. of RTA'00*, volume 1833 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2000.

14 S. Kleene. Permutability of inferences in gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 10:1–26, 1952.

15 C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logic. *Theoretical Computer Science*, 410:4747–4768, 2009.

16 D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1-2):125–157, 1991.

17 G. Mints. Normal forms for sequent derivations. In P. Odifreddi, editor, *Kreiseliana*, pages 469–492. A. K. Peters, Wellesley, Massachusetts, 1996.

18 S. Negri and J. von Plato. *Structural proof theory.* Cambridge University Press, 2001.

19 D. Prawitz. *Natural Deduction. A Proof-Theoretical Study.* Almquist and Wiksell, 1965.

20 H. Schwichtenberg. Termination of permutative conversions in intuitionistic gentzen calculi. *Theoretical Computer Science*, 212(1-2):247–260, 1999.

21 R. J. Simmons. Structural focalization. *ACM Transactions on Computational Logic*, 15(3):21:1–21:33, 2014.

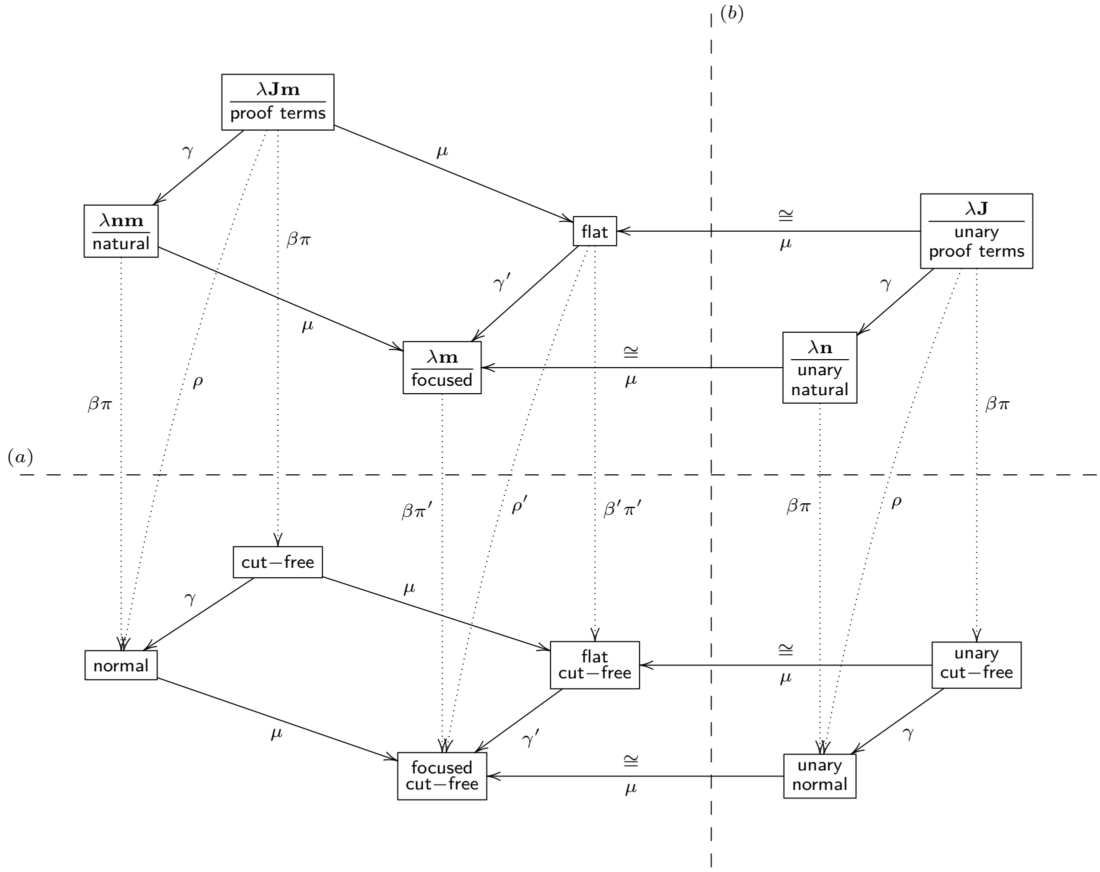
22 A. Troelstra and H. Schwitchtenberg. *Basic Proof Theory.* Cambridge Univ. Press, 2000.

23 J. von Plato. Natural deduction with general elimination rules. *Annals of Mathematical Logic*, 40(7):541–567, 2001.

24 N. Zeilberger. On the unity of duality. *Annals of Pure and Applied Logic*, 153(1-3):66–96, 2008.

25 J. Zucker. The correspondence between cut-elimination and normalization. *Annals of Mathematical Logic*, 7:1–112, 1974.

Figure 14 The internal structure of the sequent calculus $\lambda\mathbf{Jm}$



- Cut-free classes below line (a).
- Unary fragments ($\cong$ to natural deduction with gen. elimination rule) to the right of line (b).
- The class of unary proof terms (resp. unary natural terms, unary cut-free terms, unary normal terms) contained in the class of proof terms (resp. natural terms, cut-free terms, normal terms).
- $\beta\pi$ - cut-elimination; γ, μ - "bureaucracy" conversions; ρ - normalization.
- Naming of faces in the right cube: N, S, E, W, F(=Front), B(=Back)
- Naming of faces in the left cube: NL(=North face of the Left cube), SL, FE (=Front East), FW(=Front West), BE(=Back East), BW(=Back West)