Polarized proof-nets and $\lambda\mu$-calculus

Olivier LAURENT

Institut de Mathématiques de Luminy

163, avenue de Luminy - case 907

13288 MARSEILLE cedex 09 FRANCE

olaurent@iml.univ-mrs.fr

Abstract

We first define *polarized proof-nets*, an extension of MELL proof-nets for the polarized fragment of linear logic; the main difference with usual proof-nets is that we allow structural rules on any negative formula. The essential properties (confluence, strong normalization in the typed case) of polarized proof-nets are proved using a reduction preserving translation into usual proof-nets.

We then give a reduction preserving encoding of Parigot's $\lambda\mu$-terms for classical logic as polarized proof-nets. It is based on the intuitionistic translation: $A \to B \rightsquigarrow\ !A \multimap B$, so that it is a straightforward extension of the usual translation of λ-calculus into proof-nets. We give a reverse encoding which sequentializes any polarized proof-net as a $\lambda\mu$-term.

In the last part of the paper, we extend the σ-equivalence for λ-calculus to $\lambda\mu$-calculus. Interestingly, this new σ-equivalence relation identifies *normal* $\lambda\mu$-terms. We eventually show that two terms are equivalent iff they are translated as the same polarized proof-net; thus the set of polarized proof-nets represents the quotient of $\lambda\mu$-calculus by σ-equivalence.

Introduction

In the last ten years, much work has been done to solve the so-called *determinization* problem for classical logic: finding some computational interpretation of classical proofs, similar to the *Curry-Howard correspondence* for intuitionistic logic. We will be interested in two kinds of solutions that have been proposed.

- The sequent calculus approach has two main instances: Girard's LC [5] is a deterministic sequent calculus for classical logic based on a *polarization* of formulas, with a semantics of proofs in coherent spaces; the $\mathrm{LK^{tq}}$ system of Danos-Joinet-Schellinx [2] gives an extensive description of the deterministic reduction strategies that may be applied to LK. Both LC and $\mathrm{LK^{tq}}$ have translations into linear logic that preserve reductions (see also Quatrini-Tortora [11]).

- The λ-calculus approach consists in extending the λ-calculus with control operators typed by classical schemes. For example one adds a new constant *call/cc* typed by the Peirce law $((A \to B) \to A) \to A$. Unfortunately the reduction rules for this new constant depend on the reduction strategy (call-by-name or call-by-value), contradicting the Church-Rosser property for λ-calculus.

  The $\lambda\mu$-calculus of M. Parigot [9] on the other hand is based on a natural deduction with multiple conclusions and enjoys confluence. As before there are some good translations of $\lambda\mu$-calculus into linear logic. Furthermore it has been recently given a nice categorical semantics, the *control categories* of P. Selinger [14], which as a by-product, extends the language of types with a new disjunctive connective.

As in the λ-calculus where the σ-equivalence [12, 13] identifies terms that differ only in their sequential structure (e.g., $(\lambda x_1.\lambda x_2.u)v_1v_2$ and $(\lambda x_2.\lambda x_1.u)v_2v_1$), $\lambda\mu$-calculus terms contain pieces of information, which are unnecessary from the operational viewpoint. Indeed the control category semantics identifies distinct *normal* $\lambda\mu$-terms. So two questions naturally arise: *find the σ-equivalence for $\lambda\mu$-calculus*; *find some parallel syntax which identifies σ-equivalent terms.* In the λ-calculus, these two questions are answered by means of a translation of intuitionistic logic into proof-nets. However as to the present work, the translations of $\lambda\mu$-calculus into linear logic fail to solve these problems, essentially because they preserve the sequential information by translating it as exponential boxes (typically the t-translation in [2] defines the encoding of the classical arrow by: $A \to B \rightsquigarrow\ !?A \multimap ?B$).

In this paper we set up and study a translation of $\lambda\mu$-calculus into *polarized* proof-nets (PPN) for the fragment LLP of linear logic [7, 8]. LLP is a subsystem of LL dealing with polarized formulas where polarities are defined as a linear version of LC's polarities. Polarized proof-nets allow a finer use of exponential boxes: structural rules, which are reserved to formulas of the shape $?A$ in LL, are now applied to any *negative* formula (saving uses of the ? connective). Dually the cut elimination takes advantage of the geometrical properties of polarized proof-nets for duplicating any $\otimes$*-tree* (saving uses of the !-boxes). We prove the basic properties of PPN (confluence, normalization) by using a reduction preserving translation of polarized proof-nets into usual proof-nets which may be seen as an analogue of CPS-translations from $\lambda\mu$-calculus into λ-calculus.

We shall show that the translation of $\lambda\mu$-calculus preserves reductions and that it is surjective. Interestingly enough, it translates $A \to B$ as $!A \multimap B$ just like the usual encoding of intuitionistic logic[1]. As a consequence the translation is a straightforward extension of the λ-calculus translation [1, 12]. Furthermore, since there is very little difference between LLP and LC (the two systems are equivalent in the categorical sense), our framework conciliates LC and the $\lambda\mu$-calculus, allowing the use of LC as a typing system for $\lambda\mu$-terms and endowing $\lambda\mu$-calculus with LC structure (e.g., its denotational semantics). In other terms we have established a Curry-Howard correspondence between LC and $\lambda\mu$-calculus.

We shall also define the *σ-equivalence* for $\lambda\mu$-calculus as an extension of the σ-equivalence for λ-calculus. We show that it is operationally innocuous as it is included in the $\beta\eta$-equivalence of $\lambda\mu$-calculus. This result is slightly weaker than in the λ-calculus since it must make use of η-equivalence (whereas σ-equivalence for λ-terms is included in β-equivalence). We show that σ-equivalence is complete w.r.t. the translation: two terms are equivalent iff they are translated as the same proof-net.

For the sake of simplicity, we shall stick to simply typed $\lambda\mu$-calculus. However our translation is easily extendable to some richer language: pairing may be encoded by the & connective, Selinger's disjunctive connective may be encoded by the $\parr$ connective. As in the λ-calculus, we can use linear first and second order quantifiers to encode the classical ones. The proof-net technology needed for all these extensions has been developed in [7, 8]. Also, in the last section we use the same trick as in the λ-calculus for applying our results to the untyped $\lambda\mu$-calculus.

Note that as the σ-equivalence identifies normal $\lambda\mu$-terms, and since two equivalent terms correspond to the same proof-net, it is impossible to distinguish them by evaluation in a context. Thus $\lambda\mu$-calculus violates Böhm's theorem which enforces David and Py result, who found two

[1] In fact the t-translation may be factorized through ours: first use the LLP-translation, then use the translation of LLP into LL shown in section 1.4.

normal terms that are operationally indistinguishable [4]. It is worth noting that their two terms are not σ-equivalent, from which we may deduce that polarized proof-nets themselves do not satisfy Böhm's theorem.

1 Polarized proof-nets

We use a fragment of multiplicative exponential polarized proof-nets [7, 8] which has a particularly simple correctness criterion to encode $\lambda\mu$-calculus.

1.1 Definitions

Definition 1 (Polarized formula)
Starting with a set of atoms (denoted by X), we define *output* (denoted by N, M, ...) and *anti output* (denoted by P, Q, ...) formulas:

$$\begin{array}{rcl} N & ::= & X \quad | \quad ?P \parr N \\ P & ::= & X^\perp \quad | \quad !N \otimes P \end{array}$$

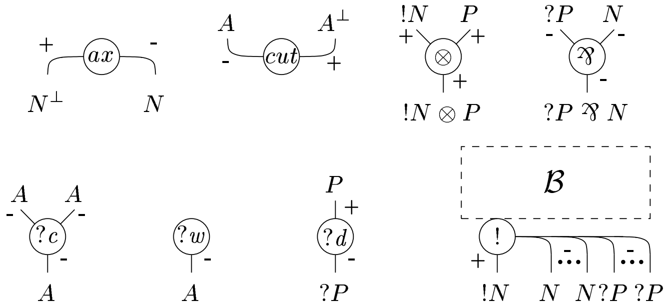
Formulas of the shape $?P$ (resp. $!N$) are called *input* (resp. *anti input*) formulas. *Negative formulas* (resp. *positive formulas*) are input and output (resp. anti input and anti output) formulas.

The negation is involutive with $(?P \parr N)^\perp = !P^\perp \otimes N^\perp$ and $(?P)^\perp = !P^\perp$.

The terminology "*input*" and "*output*" will become clear in section 2.2.

Definition 2 (Proof-structure)
A *proof-structure* is a finite acyclic oriented graph built over the alphabet of nodes represented below (where the orientation is the top-bottom one), i.e. respecting for each node: the orientation, the number of incident (top) edges (the *premises* of the node), the number of emergent (bottom) edges (the *conclusions* of the node), the typing of each edge by a polarized formula.

Each edge is conclusion of exactly one node and premise of at most one node. Edges which are not premise of any node are the conclusions of the proof-structure.



where A is a negative formula.

Additionally, to each !-node with conclusions $\{!N, \Gamma\}$ is associated a *box*, that is a proof-structure with conclusions $\{N, \Gamma\}$. We say that a node occurs at *depth* 0 in the proof-structure $\mathcal{R}$ if it is a node of $\mathcal{R}$, and that it occurs at depth $k+1$ in $\mathcal{R}$ if it occurs at depth k in some box associated to a !-node of $\mathcal{R}$. The depth of $\mathcal{R}$ is the maximal depth of the nodes occurring in $\mathcal{R}$. We assume it is always finite.

We distinguish two kinds of *contractions* ($?c$-nodes) (resp. *weakenings* ($?w$-nodes)): one for output types denoted by *co* (resp. *wo*) and the other one for input types denoted by *ci* (resp. *wi*). If the distinction is not needed we still use $?c$ (resp. $?w$).

Definition 3 (Edges and nodes)

- An edge is *positive* (resp. *negative*) if the associated formula is positive (resp. negative).
- A node is *positive* (resp. *negative*) if all its edges are positive (resp. negative), thus positive nodes are $\otimes$-nodes and negative nodes are $\parr$-, $?c$- and $?w$-nodes.
- A *cut*-node is a *structural* cut if its negative premise is conclusion of $?d$, $?c$, $?w$ or !. The other cuts, i.e. $\otimes/\parr$ and *ax*, are called *multiplicative* cuts.

1.2 Correctness criterion

Definition 4 (Correction graph)
Given a proof-structure, its *correction graph* is obtained by orienting upwardly (resp. downwardly) the positive (resp. negative) edges, i.e. by reversing the orientation of positive edges, and by erasing boxes (just keeping the !-node).

Since we have defined two orientations on proof-structures, we have to introduce some terminology to distinguish them. In the sequel, we will never mention the orientation coming from the definition of a proof-structure except through "geometrical" terms such as *above*, *below*, *down*, *up*, ... To talk about the orientation coming from the correction graph, we will use "ordering" terms such as *initial*, *final*, *maximal*, ...

Definition 5 (Proof-net)
A proof-structure is *correct* or is a *proof-net* if:

- its correction graph is an acyclic oriented graph;
- the number of positive conclusions plus $?d$-nodes is one;
- and recursively the boxes are also correct proof-structures.

Remarks:

- The complexity of the verification of correctness is linear in the size of the proof-structure (i.e. its number of nodes).
- The correction graph of a proof-structure without cut is always acyclic. A path in such a correction graph can start either on a negative edge and in this case it goes towards a conclusion or on a positive edge and in this case it can only go to an axiom and then towards a conclusion, then the path ends because the only way to continue is to use a cut.
- A polarized proof-net has one non $?w$ initial node at depth 0 called the *main* initial node which is either a positive conclusion or a $?d$-node.
- A polarized proof-net with no anti input conclusion has at least one output conclusion. Indeed a polarized proof-net without anti input conclusion has an anti output edge at depth 0; starting from this edge, it is possible to go through an oriented path (of the correction graph) containing only output and anti output edges yielding to an output conclusion.

The orientation defined by the correction graph and the acyclicity of this graph induce a partial order on the set of the nodes of a proof-net. A node is *final* if it is maximal at depth 0 for this order. A *cut*-node is a *maximal cut-node* if it is at depth 0 and maximal inside the subset of *cut*-nodes.

Remark: A negative node the conclusion of which is conclusion of the proof-structure is final. This would not be true anymore with $\forall$ and & because the correctness criterion for such proof-nets introduces particular edges starting from these nodes [8].

1.3 Cut elimination

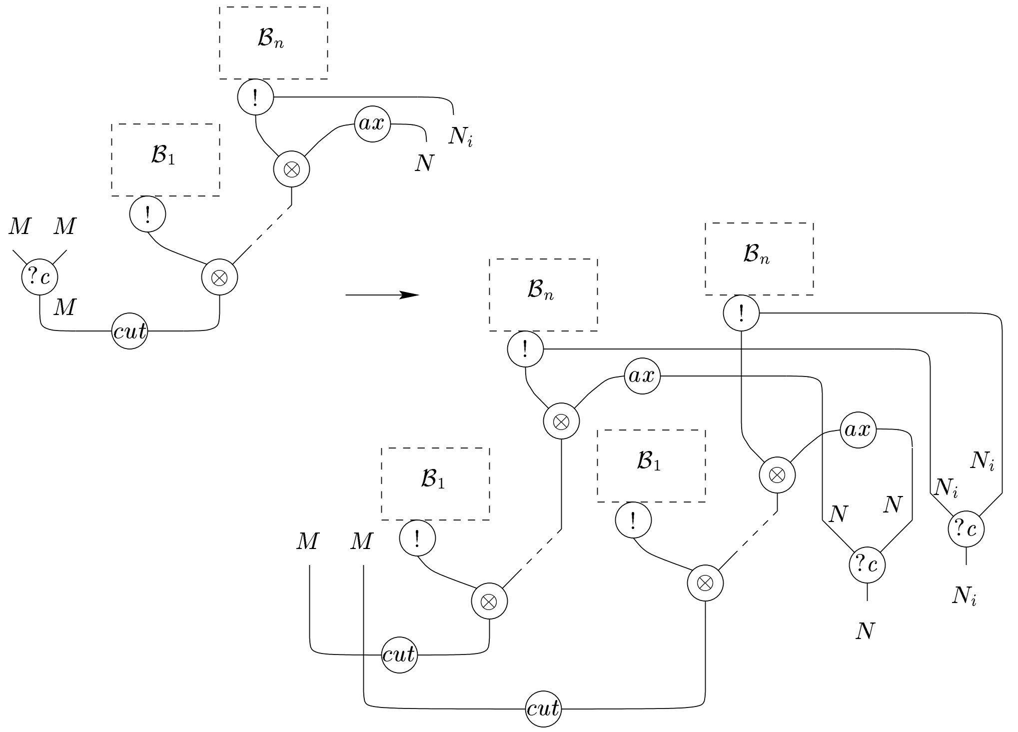
Definition 6 ($\otimes$-tree)
The set of nodes above a positive edge e is called its $\otimes$*-tree* and e is called the *root* of the tree (the $\otimes$-tree of a $\otimes$-node is the $\otimes$-tree of its conclusion). It has a very particular structure: either it is just a box or just an axiom or it is a $\otimes$-node with a box above one of its premises and another $\otimes$-tree above the other.

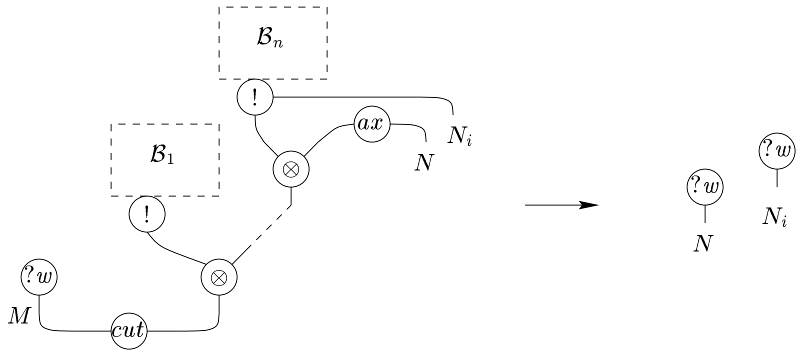
If the $\otimes$-tree is a box, it is said to be *flat*. A non flat $\otimes$-tree contains exactly one axiom the negative conclusion of which is called the *main* conclusion of the $\otimes$-tree. All the other non-root conclusions of a $\otimes$-tree are the *auxiliary* conclusions.

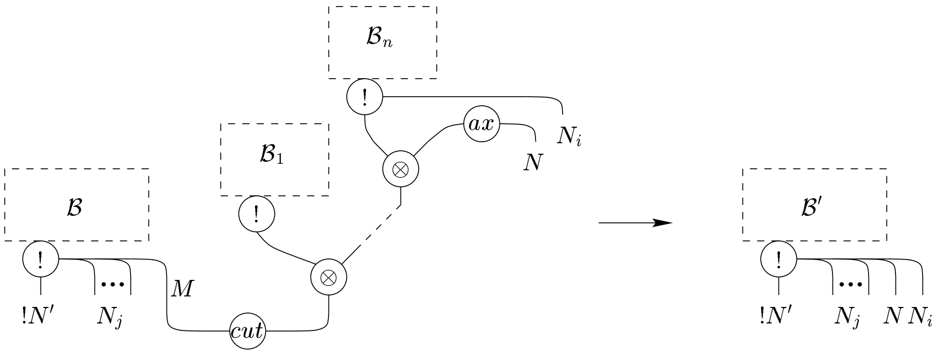
Remark: The $\otimes$-tree above a positive edge e is the smallest sub-proof-net containing e, also called the *kingdom* of e. Note that the kingdom of an edge e typed by $!N$ is the whole box of which e is a conclusion, so that $\otimes$-trees may be considered as generalizations of boxes for all positive edges.

A *cut*-node always has a $\otimes$-tree above its positive premise. Maximality of the *cut*-node entails that there is no other *cut*-node below the conclusions of the $\otimes$-tree.

The cut elimination steps for ax, $\otimes/\parr$, $?d/!$, $ci/!$, $wi/!$ and $!/!$ reductions are the same as in usual proof-nets [1, 12]. We add new steps for the new structural rules which are similar to the $ci/!$, $wi/!$ and $!/!$ if we consider $\otimes$-trees as boxes.







where $\mathcal{B}'$ is obtained by a cut between the M conclusion of $\mathcal{B}$ and the $\otimes$-tree.

We use the notation $\mathcal{R} \to \mathcal{R}'$ if $\mathcal{R}$ can be reduced in $\mathcal{R}'$ by one step of cut elimination.

Proposition 1
Correctness is preserved by reduction.

PROOF: There are two interesting cases:

- For a cut between a $?d$-node and a box, the main initial node was the $?d$ one which is replaced by the main initial node of the box. As for the acyclicity, if a cycle is created, it must go through the box but it is impossible because all its conclusions are negative.
- For a cut on a $?c$-node (or $?w$), we have to remark that all the conclusions of the $\otimes$-tree are negative thus there is no problem to propagate the $?c$-node (or $?w$) on them. $\square$

1.4 Translation into usual proof-nets

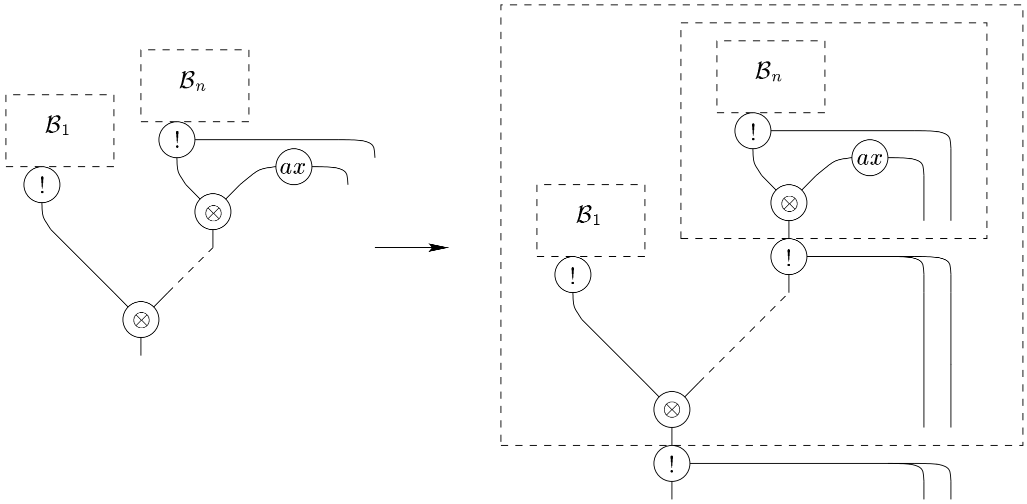
There is an encoding of polarized proof-nets into usual multiplicative exponential proof-nets which preserves reduction. This gives a simple way to prove different properties of polarized proof-nets.

We prefix output formulas with a ? so that the encoding of each negative formula begins with a ?:

$$\begin{array}{ccc} \overline{X} & = & ?X \\ \overline{?P \parr N} & = & ?(?\overline{P} \parr \overline{N}) \\ \overline{?P} & = & ?\overline{P} \end{array}$$

The translation of positive formulas is obtained by duality.

We translate proof-nets by replacing each $\parr$-node by the sequence $\parr$-$?d$ and by putting a box around the $\otimes$-tree of each $\otimes$-node. This translation emphasizes the fact that $\otimes$-trees behave like boxes.



Remark: This translation is the linear counterpart of CPS-translations. Indeed, very informally, CPS-translations act by adding $\neg\neg$ in suitable places which amounts to adding ? in the corresponding places in linear logic.

If we consider the usual translation of the intuitionistic arrow $A \to B \rightsquigarrow\ !A \multimap B$ (as done in section 2.2) and apply the translation $\overline{()}$, we obtain $!?A \multimap ?B$ which is the basis of the t-translation used in [2] to translate $\lambda\mu$-calculus into linear logic.

We have the following correspondence between reductions:

PPN reductions ($\mathcal{R}$)		PN reductions ($\overline{\mathcal{R}}$)
ax	$\rightsquigarrow$	$(!/!)^*; ax$
$\otimes/\parr$	$\rightsquigarrow$	$?d/!; \otimes/\parr$
structural output	$\rightsquigarrow$	exponential
structural input (exponential)	$\rightsquigarrow$	exponential

Theorem 1 (Simulation)
If $\mathcal{R} \to_{LLP} \mathcal{R}'$ then $\overline{\mathcal{R}} \to^+_{LL} \overline{\mathcal{R}'}$. Conversely if $\overline{\mathcal{R}} \to^*_{LL} \mathcal{S}$, let $\mathcal{R}'$ be the proof-net obtained from $\mathcal{R}$ by applying the corresponding reduction steps, $\mathcal{S}$ gives $\overline{\mathcal{R}'}$ by reducing all its $\otimes/\parr$-cuts and its $!/!$-cuts and ax-cuts inside boxes not containing $\otimes$-trees (in particular if $\mathcal{S} = \overline{\mathcal{R}_1}$ then $\mathcal{R} \to^*_{LLP} \mathcal{R}_1$).

PROOF: The first statement is immediate. As to the converse, the reduction $\overline{\mathcal{R}} \to^* \mathcal{S}$ may have reduced some $?d/!$-cuts (resp. $!/!$-cuts) but omitted the resulting $\otimes/\parr$'s (resp. $!/!$'s and ax's). This is why we have to reduce the $\otimes/\parr$-cuts (resp. some $!/!$-cuts and ax-cuts) in $\mathcal{S}$ to obtain $\overline{\mathcal{R}'}$. ◻

Lemma 1 (Injectivity)
The $\overline{()}$-translation is injective.

Corollary 1.1 (Confluence)
Reduction of polarized proof-nets is confluent.

PROOF: If $\mathcal{R} \to^* \mathcal{R}_1$ and $\mathcal{R} \to^* \mathcal{R}_2$ then, by theorem 1, $\overline{\mathcal{R}} \to^* \overline{\mathcal{R}_1}$ and $\overline{\mathcal{R}} \to^* \overline{\mathcal{R}_2}$ thus, by confluence of usual proof-nets, there exists $\mathcal{S}$ such that $\overline{\mathcal{R}_1} \to^* \mathcal{S}$ and $\overline{\mathcal{R}_2} \to^* \mathcal{S}$. Let $\mathcal{R}'_1$ and $\mathcal{R}'_2$ be the corresponding reducts of $\mathcal{R}_1$ and $\mathcal{R}_2$, we have by theorem 1 that $\overline{\mathcal{R}'_1}$ and $\overline{\mathcal{R}'_2}$ are obtained from $\mathcal{S}$ by reducing the same cuts, thus $\overline{\mathcal{R}'_1} = \overline{\mathcal{R}'_2}$ and we can conclude by lemma 1. ◻

Corollary 1.2 (Strong normalization)
There is no infinite sequence of reductions in polarized proof-nets.

These properties can also be obtained with usual methods such as those used in [1, 12].

2 The $\lambda\mu$-calculus

The $\lambda\mu$-calculus has been introduced by M. Parigot in [9] as an extension of λ-calculus which gives an algorithmic interpretation of classical proofs. We will see how it can be interpreted into polarized proof-nets and how this induces new identifications in $\lambda\mu$-calculus.

2.1 Definitions

Definition 7 ($\lambda\mu$-term)
Given two disjoint denumerable sets of variables: called λ-variables (denoted by x, y, z, ...) and μ-variables (denoted by α, β, γ, ...), the *$\lambda\mu$-terms* are defined by:

$$u \quad ::= \quad x \quad | \quad \lambda x.u \quad | \quad \mu\beta[\alpha]u \quad | \quad (u)u$$

The λ and μ constructions are called *abstractions* and $(u)v$ is the *application*; λ and μ are binders. We consider terms modulo α-conversion on λ- and μ-variables. We use the notations x (resp. α) $\in u$ for "x (resp. α) free in u" and $(u)v_1...v_n$ for $(...((u)v_1)v_2...)v_n$.

For the simply typed $\lambda\mu$-calculus, a typing judgment for the term u is $\Gamma \vdash u : N \mid \Delta$ where Γ (resp. Δ) contains typing declarations for λ-variables (resp. μ-variables). Each variable appears at most once in a context. The derivation rules are:

$$\frac{}{x : N \vdash x : N \mid}\ \text{var}$$

$$\frac{\Gamma, x : N \vdash u : M \mid \Delta}{\Gamma \vdash \lambda x.u : N \to M \mid \Delta}\ \text{abs}$$

$$\frac{\Gamma \vdash u : N \mid \beta : M, \Delta}{\Gamma \vdash \mu\beta[\alpha]u : M \mid \alpha : N, \Delta}\ \mu$$

$$\frac{\Gamma \vdash u : N \to M \mid \Delta \qquad \Gamma' \vdash v : N \mid \Delta'}{\Gamma, \Gamma' \vdash (u)v : M \mid \Delta, \Delta'}\ \text{app}$$

The (*abs*) (resp. (μ)) rule contains an implicit weakening if x (resp. β) does not appear in the context. The (*app*) rule contains implicit contractions if variables appear in both Γ and Γ' or both Δ and Δ'. There is also an implicit contraction in the (μ) rule if α appears in Δ.

Alternatively, the μ-rule may be decomposed by introducing the notion of *named term* $[\alpha]u$. A named term has no type (or has type $\perp$) and appears in a judgment $\Gamma \vdash [\alpha]u \mid \Delta$. The ($\mu$) rule is split into:

$$\frac{\Gamma \vdash u : N \mid \Delta}{\Gamma \vdash [\alpha]u \mid \alpha : N, \Delta}\ \mu\text{-name}$$

$$\frac{\Gamma \vdash [\alpha]u \mid \beta : M, \Delta}{\Gamma \vdash \mu\beta[\alpha]u : M \mid \Delta}\ \mu\text{-abs}$$

Remark: If a $\lambda\mu$-term u is typable in this system, the conclusion of the typing derivation is $\Gamma \vdash u : N \mid \Delta$ where Γ and Δ exactly contain the free variables of u. In particular talking about a term or about a typing judgment is equivalent since there is at most one typing judgment for each term. Moreover for this typing judgment there exists at most one typing derivation.

The two reduction rules are the usual ones for $\lambda\mu$-calculus:

$$\begin{array}{lll} (\lambda x.u)v & \rightarrow_\beta & u[{}^{v}/_{x}] \\ (\mu\alpha.u)v & \rightarrow_\mu & \mu\alpha.u[{}^{[\alpha](w)v}/_{[\alpha]w}] \end{array}$$

We use the notation $u \rightarrow v$ if u can be reduced in v by one step of β- or μ-reduction.

2.2 Translation into proof-nets

We now give a first translation, denoted by $()^\circ$, of $\lambda\mu$-terms into proof-nets. Simple types are mapped to output formulas by:

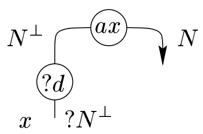
$$\begin{array}{rcccl} X^\circ & = & X & & \\ (N \rightarrow M)^\circ & = & !N^\circ \multimap M^\circ & = & ?N^{\circ\perp} \parr M^\circ \end{array}$$

The translation of typed $\lambda\mu$-terms is defined by induction on the typing: a proof of the sequent $\Gamma \vdash u : N \mid \Delta$ is interpreted by a proof-net with conclusions $?\Gamma^{\circ\perp}, N^\circ, \Delta^\circ$. There lies the justification for the input/output terminology: λ-variables are typed by input formulas; μ-variables (continuation variables) and the term are typed by output formulas.

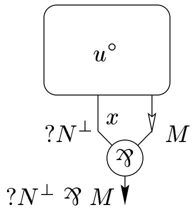
For the purpose of the translation, we add to proof-nets a labelling of their conclusions: each input conclusion in $?\Gamma^{\circ\perp}$ is labelled by the name of the corresponding λ-variable in Γ, each output conclusion in Δ° is labelled by the name of the corresponding μ-variable in Δ. The conclusion N° is said to be *distinguished* and is the only one which has no associated name.

In the pictures, we use $\longrightarrow$ for the distinguished conclusion, $\longrightarrow$ for an old distinguished conclusion and (β) for the old name of the distinguished conclusion.

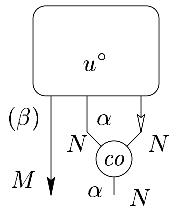
- $(x^N)^\circ$



- $(\lambda x^N.u^M)^\circ$

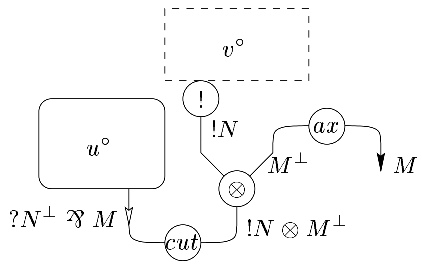


- $(\mu\beta^M[\alpha]^N u^N)^\circ$



This is for the particular case where $\alpha \in u$ and $\beta \in u$. If $\alpha \notin u$ we do not need the *co*-node, if $\beta \notin u$ we need a *wo*-node. Note that if $\alpha \notin u$ and $\beta \in u$, the only effect of the translation is to swap the status of the two conclusions, that is to associate the name α to the distinguished conclusion (which is no more distinguished) and forget the name of the conclusion labelled by β (which becomes the new distinguished conclusion) leaving the proof-net unchanged.

- $((u^{N \to M}) v^N)^\circ$



We can separate in two parts the case of $\mu\beta[\alpha]u$ if we translate named terms $[\alpha]u$ as proof-nets without distinguished conclusion. The $[\alpha]$ construction on u introduces a contraction node if $\alpha \in u$. The $\mu\alpha$ construction introduces a weakening node if $\alpha \notin u$.

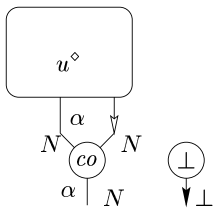
Another way to deal with named terms is to put explicitly the type $\perp$. The two μ-rules become:

$$\frac{\Gamma \vdash u : N \mid \Delta}{\Gamma \vdash [\alpha]u : \perp \mid \alpha : N, \Delta}\ \mu\text{-name'}$$

$$\frac{\Gamma \vdash [\alpha]u : \perp \mid \beta : M, \Delta}{\Gamma \vdash \mu\beta[\alpha]u : M \mid \Delta}\ \mu\text{-abs'}$$

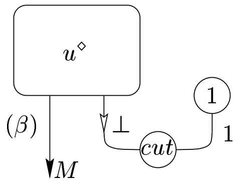
These two rules can be translated into proof-nets[2] through the translation $()^\diamond$:

- $([\alpha]^N u^N)^\diamond$



- $(\mu\beta^M . u^\perp)^\diamond$

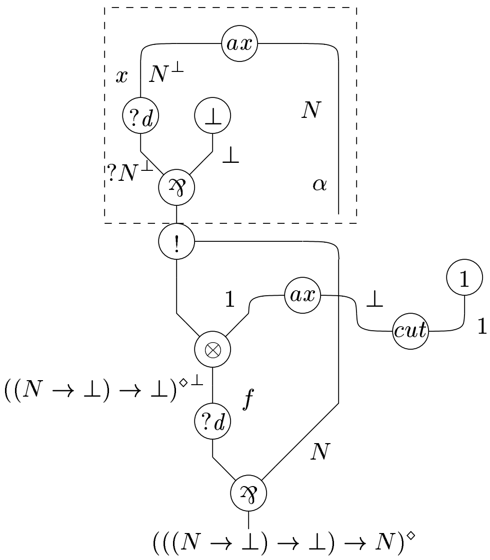
[2] The introduction of 1- and $\perp$-nodes does not modify the correctness criterion.



The translation $()^\diamond$ is the same as $()^\circ$ for the other rules. We can recover $()^\circ$ from $()^\diamond$ by reducing the cuts on constants. Indeed, by definition of $\lambda\mu$-calculus, $\mu\beta$ and $[\alpha]$ always come together, so the constant cuts are always between 1-nodes and $\perp$-nodes which vanish by reduction.

The $()^\diamond$ translation allows to account with a slight generalization of $\lambda\mu$-calculus with a new rule for $\perp$ corresponding to the construction $\mu\beta^M.u^\perp$ for any u of type $\perp$ (not only named terms). In this case 1-nodes are not always cut against $\perp$-ones. Since $\perp$ is an output formula, the $\perp$-node is just a particular case of the *wo*-node. On the other hand, the 1-node introduces a new kind of leaves for $\otimes$-trees.

With this extension, we can, for example, translate the $\mathcal{C}$ operator of Felleisen of type $\neg\neg N \to N$ defined in $\lambda\mu$-calculus by $\lambda f^{\neg\neg N}.\mu\alpha^N.(f)\lambda x^N.[\alpha]^N x$:



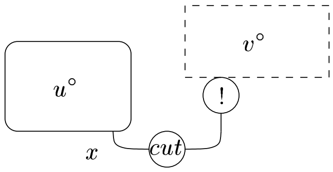
2.3 Simulation of reduction

The translation $()^\circ$ has also a dynamic meaning: it simulates $\beta\mu$-reduction by cut elimination in proof-nets.

Until the end of the paper, we have to identify all the binary trees of contractions with the same number of nodes, the sequence $?w$-$?c$ with a simple edge and a *ci*-node on two input conclusions of a box with the same node inside the box. Another solution has been proposed in [3, 12] to get rid of this problem with generalized contractions.

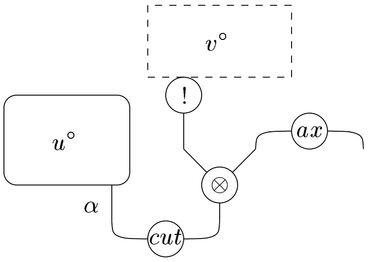
Final $?w$-nodes of a proof-net correspond to variables in the context not free in the term. As done for typing derivations, we want to ignore them. In the sequel we will not care about such final $?w$-nodes. In particular we say that two proof-nets are equal if they differ only by final $?w$-nodes. This is the counterpart of the fact that, in various systems, two typing proofs that differ only in useless variable declarations may be considered equal.

Lemma 2 (λ-substitution)
Modulo final ?w-nodes and structure of contraction trees:



$$\to^+ \quad (u[{}^{v}/_{x}])^\circ$$

Lemma 3 (μ-substitution)
Modulo final ?w-nodes and structure of contraction trees:



$$\to^* \quad (u[{}^{[\alpha](w)v}/_{[\alpha]w}])^\circ$$

PROOF: These two lemmas are proved by induction on u. $\square$

Theorem 2 (Simulation)
If $u \to v$ then $u^\circ \to^* v^\circ$.

PROOF: We just have to apply lemmas 2 and 3. In the case of β-reduction, the simulation is strict: one step in $\lambda\mu$-calculus corresponds to at least one step in proof-nets. On the other hand, a μ-reduction may be translated by identity, typically in the case: $(\mu\alpha[\alpha]u)v \to_\mu \mu\alpha[\alpha](u)v$ with $\alpha \notin u$. $\square$

Remark: Due to the σ-equivalence (next section), we cannot hope for any converse result. For example, if $u = \mu\gamma[\alpha]\lambda x.\mu\alpha'[\beta]\lambda y.(\mu\beta'[\gamma](y)x)t$ and $v = \mu\gamma[\beta]\lambda y.\mu\beta'[\alpha]\lambda x.\mu\alpha'[\gamma](y)x$, $u^\circ \to^* v^\circ$ but v is not a reduct of u.

We now consider proof-nets up to multiplicative (i.e. $\otimes/\parr$ and ax) reductions. The justification for doing so is that these reductions are operationally simple: they strictly decrease the size of the net and they are local, so that the heart of the dynamics may be thought of as lying in structural reductions.

Definition 8 (Translation $()^\bullet$)
The translation $u^\bullet$ of a $\lambda\mu$-term u is the multiplicative normal form of the proof-net u°.

Remark: We have defined three translations with the following relations:

$$u^\diamond \xrightarrow{1/\perp} u^\circ \xrightarrow{\otimes/\parr, ax} u^\bullet$$

With the translation $()^\bullet$, the last theorem is no longer correct, for example:

$$((\mu\alpha.u)v_1)v_2 \quad \to_\mu \quad (\mu\alpha.u[{}^{[\alpha](w)v_1}/_{[\alpha]w}])v_2 \quad \to_\mu \quad \mu\alpha.u[{}^{[\alpha](w)v_1 v_2}/_{[\alpha]w}]$$

but in the corresponding proof-net $(((\mu\alpha.u)v_1)v_2)^\bullet$ these two steps are done in just one, so that the translation of the middle term is not reachable from the translation of the first one. Asking for "consecutive" μ-redexes to be reduced in one step by modifying the μ-reduction: $(\mu\alpha.u)v_1...v_n \to_{\mu'}$

$\mu\alpha.u[^{[\alpha](w)v_1...v_n}/_{[\alpha]w}]$ (where $(\mu\alpha.u)v_1...v_n$ is not applied to another term v_{n+1}) does not solve the problem since as shown in the following example the sequence $v_1...v_n$ may be hidden in the $\lambda\mu$-term:

$$
\begin{array}{lll}
((\lambda x.(\mu\alpha.u)v_1)t)v_2 & \rightarrow_{\mu'} & ((\lambda x.\mu\alpha.u[^{[\alpha](w)v_1}/_{[\alpha]w}])t)v_2 \\
& \rightarrow_{\beta} & (\mu\alpha.u[^{[\alpha](w)v_1}/_{[\alpha]w}][^t/_x])v_2 \quad \rightarrow_{\mu'} \quad \mu\alpha.u[^{[\alpha](w)v_1v_2}/_{[\alpha]w}][^t/_x]
\end{array}
$$

On the other hand $()^\bullet$ translates normal terms as cut-free proof-nets, β-redexes as $!/?$ cuts and μ-redexes as $\otimes/co$, $\otimes/wo$ or $\otimes/!$ cuts (except in some particular cases like $(\mu\alpha[\alpha]x)v$).

2.4 Sequentialization

We now address the question of surjectivity of the translation $()^\bullet$.

Definition 9 ($\lambda\mu$-proof-structure)
A *$\lambda\mu$-proof-structure* is a proof-structure with no positive conclusion, no final *wi*-node and only structural cuts.

The "*no final wi-node*" constraint is not needed for theorem 3 if we want to sequentialize proof-nets as typing derivations with explicit structural rules because they just add variables in the context not free in the $\lambda\mu$-term (see page 11).

Remark: When a proof-structure has no positive conclusion, the correctness condition on the number of *?d*-nodes can be replaced by: *exactly one dereliction at depth* 0 *and in each box.*

Lemma 4
Let $\mathcal{R}$ be a proof-structure with a final negative node n, $\mathcal{R} \setminus n$ is the graph obtained from $\mathcal{R}$ by erasing the node n. Then $\mathcal{R} \setminus n$ is a proof structure and if $\mathcal{R}$ is correct, $\mathcal{R} \setminus n$ is correct.

Lemma 5
Let u be a term with two free λ- (resp. μ-) variables x_1 and x_2 (resp. α_1 and α_2), $(u[^x/_{x_1}, {}^x/_{x_2}])^\bullet$ (resp. $(u[^\alpha/_{\alpha_1}, {}^\alpha/_{\alpha_2}])^\bullet$) is obtained from $u^\bullet$ by adding a ci-node (resp. co-node) between the two conclusions corresponding to x_1 and x_2 (resp. α_1 and α_2).

Lemma 6
Let $\mathcal{R}$ be a cut-free proof-net, if e is a negative edge either it is a conclusion of $\mathcal{R}$ or moving downward from e yields to a final negative node.

Lemma 7
Let $\mathcal{R}$ be a proof-net without any final negative node. If c is a maximal[3] *cut-node in $\mathcal{R}$ then c is* splitting, *that is $\mathcal{R}$ is obtained by cutting the negative conclusion A of a proof-net $\mathcal{R}^-$ with the positive conclusion $A^\perp$ of a proof-net $\mathcal{R}^+$. Furthermore $\mathcal{R}^+$ is a $\otimes$-tree.*

PROOF: The last four results are immediate. □

Theorem 3 (Sequentialization)
A proof-structure with a distinguished conclusion is the translation $()^\bullet$ of a $\lambda\mu$-term if and only if it is a $\lambda\mu$-proof-net.

[3]see section 1.2

PROOF: The "only if" part is immediate. Conversely, we sequentialize any $\lambda\mu$-proof-net $\mathcal{R}$ with a distinguished conclusion as a $\lambda\mu$-term. We start by associating distinct λ-variables to input conclusions and distinct μ-variables to output conclusions of $\mathcal{R}$. We then sequentialize $\mathcal{R}$ as a named term $[\beta]u$ and we define the *complete* sequentialization of $\mathcal{R}$ to be $\mu\alpha[\beta]u$ where α is the name associated to the distinguished output conclusion. The construction of $[\beta]u$ is done by induction on $\mathcal{R}$:

- If $\mathcal{R}$ has a final $⅋$-node n whose conclusion $?N^\perp ⅋ M$ has name α, let u be a sequentialization of $\mathcal{R} \setminus n$ (or of $(\mathcal{R} \setminus n) \setminus n'$ if $?N^\perp$ is introduced by a *wi*-node n') with names x for $?N^\perp$ and β for M. We sequentialize $\mathcal{R}$ as $[\alpha]^{N \to M}\lambda x^N.\mu\beta^M.u$.
- If $\mathcal{R}$ has a final *co*-node n whose conclusion N has name α, let u be a sequentialization of $\mathcal{R} \setminus n$ with names α_1 and α_2 for the premises N of n. We sequentialize $\mathcal{R}$ as $u[{}^{\alpha}/_{\alpha_1}, {}^{\alpha}/_{\alpha_2}]$ by lemma 5.
- If $\mathcal{R}$ has a final *wo*-node n whose conclusion N has name α, let u be a sequentialization of $\mathcal{R} \setminus n$. We sequentialize $\mathcal{R}$ as $[\alpha]^N\mu\delta^N.u$ where δ does not occur in u.
- If $\mathcal{R}$ has a final *ci*-node n whose conclusion $?N^\perp$ has name x, let u be a sequentialization of $\mathcal{R} \setminus n$ with names x_1 and x_2 for the premises $?N^\perp$ of n. We sequentialize $\mathcal{R}$ as $u[{}^{x}/_{x_1}, {}^{x}/_{x_2}]$ by lemma 5.
- If $\mathcal{R}$ has no final negative node, we consider a maximal *cut*-node which is splitting by lemma 7. Let $\mathcal{R}^-$ (resp. $\mathcal{R}^+$) be the proof-structure above its negative (resp. positive) premise. By lemma 7, $\mathcal{R}^+$ is a $\otimes$-tree. We have two cases depending on the type of the cut formula:
  - if it is an input formula $?N^\perp$, we sequentialize $\mathcal{R}^-$ (or $\mathcal{R}^- \setminus n'$ if $?N^\perp$ is introduced by a *wi*-node n') with name x for the conclusion $?N^\perp$ as u. Since the root of $\mathcal{R}^+$ is $!N$, $\mathcal{R}^+$ must be flat, i.e. $\mathcal{R}^+$ is a box $\mathcal{B}$ associated to a !-node. Let v be the complete sequentialization of $\mathcal{B}$ (whose conclusion N is distinguished and whose other conclusions are named as in $\mathcal{R}$). We pick an output conclusion of $\mathcal{R}^-$ (it must have one by the fourth remark in section 1.2) with name α. We sequentialize $\mathcal{R}$ as $[\alpha]^M(\lambda x^N.\mu\alpha^M.u)v^N$;
  - if it is an output formula N, let u be the sequentialization of $\mathcal{R}^-$ with the new name α associated to its conclusion N and $v_1, \ldots, v_n$ be the complete sequentializations of the boxes associated to the !-leaves of $\mathcal{R}^+$. If β is the name of the main conclusion[4] of the $\otimes$-tree $\mathcal{R}^+$, we sequentialize $\mathcal{R}$ as $[\beta]^N(\mu\alpha^N.u)v_1...v_n$.
- If $\mathcal{R}$ has no final negative node and no cut at depth 0, then let n be its main initial node which is a $?d$-node by the remark above. Then the conclusion of n is a conclusion of $\mathcal{R}$ by lemma 6. Furthermore its premise is the root of a $\otimes$-tree $\mathcal{R}^+$ and the conclusions of $\mathcal{R}^+$ are conclusions of $\mathcal{R}$ by lemma 6, i.e. $\mathcal{R}^+ = \mathcal{R} \setminus n$. If the conclusion $?M^\perp$ of n has name x and the main conclusion N of $\mathcal{R}^+$ has name α, we sequentialize $\mathcal{R}$ as $[\alpha]^N(x^M)v_1...v_n$ where $v_1, \ldots, v_n$ are the complete sequentializations of the boxes associated to the !-leaves of $\mathcal{R}^+$.

One may check that in each case the translation $()^\bullet$ applied to the constructed term yields the original proof-net (up to associativity of contractions, commutations of auxiliary doors with contractions and neutrality of weakening w.r.t. contractions, as explained above). □

[4]see section 1.3

Remarks:

- Maximal *cut*-nodes correspond to leftmost redexes in $\lambda\mu$-calculus. The translation of the leftmost redex of a $\lambda\mu$-term is a maximal *cut*-node and conversely if c is a maximal *cut*-node, there exists a $\lambda\mu$-term such that its leftmost redex is translated as c.
- The dereliction at depth 0 corresponds to the head variable of the term. If the conclusion of this node is a conclusion of the proof-net it means that the head variable of the $\lambda\mu$-term is free and linear.

This sequentialization procedure yields $\lambda\mu$-terms with a particular shape: $\mu\alpha[\beta]$ constructions surround each λ-abstraction, each application, ... The *σ-equivalence* will show that this kind of $\lambda\mu$-terms is not so peculiar because there exists at least one such term in each σ-equivalence class.

3 The σ-equivalence

We characterize now the identification between $\lambda\mu$-terms induced by the translation. The answer has already been given for the λ-calculus in [12, 13] as the σ-equivalence. We give here a "conservative" extension of this equivalence on $\lambda\mu$-terms. However this extension has a very different behavior on $\lambda\mu$-terms. The σ-equivalence of λ-calculus is defined in terms of commutations of redexes, in particular a normal term is only equivalent to itself, but here the (`pop/pop`) equation provides identifications between normal terms. We will also see a normal term equivalent to a non normal one.

3.1 Definition

Definition 10 (Atomic contexts)
An *atomic context* C_0 is obtained by applying one of the constructions of the $\lambda\mu$-calculus to a hole instead of a term:

$$C_0 \quad ::= \quad \lambda y.[\,] \quad | \quad \mu\gamma[\beta][\,] \quad | \quad ([\,])u$$

An *atomic named context* N_0 is constructed in the same way but with named terms:

$$N_0 \quad ::= \quad [\beta]\lambda y.\mu\gamma.[\,] \quad | \quad [\beta](\mu\gamma.[\,])u$$

We say that α (resp. x) is *bound* in C_0 if $\alpha \notin C_0[\mu\delta[\alpha]x]$ (resp. $x \notin C_0[x]$). We say that C_0 is *α-free* (resp. *x-free*) if α (resp. x) $\notin C_0$ and if α (resp. x) is not bound in C_0. And we use the same terminology for N_0.

Definition 11 (σ-equivalence)
The *σ-equivalence* is the smallest compatible (i.e. preserved by abstractions and application) equivalence relation on $\lambda\mu$-terms containing:

• generalization of σ-equivalence on λ-calculus for commutation of λ-redexes with all the constructions of $\lambda\mu$-calculus,

$$\begin{array}{llllr} ((\lambda x.u)v)w & \sim_\sigma & (\lambda x.(u)w)v & x \notin w & (\sigma_1) \\ (\lambda x.\lambda y.u)v & \sim_\sigma & \lambda y.(\lambda x.u)v & \begin{cases} x \neq y \\ y \notin v \end{cases} & (\sigma_2) \\ (\lambda x.\mu\alpha[\beta]u)v & \sim_\sigma & \mu\alpha[\beta](\lambda x.u)v & \alpha \notin v & (\sigma_3) \end{array}$$

- particular $\lambda\mu$-features giving new commutations,

$$
\begin{array}{llll}
[\alpha'](\mu\alpha[\beta'](\mu\beta.u)v)w & \sim_\sigma & [\beta'](\mu\beta[\alpha'](\mu\alpha.u)w)v & \begin{cases}\alpha\notin v\\ \beta\notin w\end{cases} \quad (\texttt{push/push})\\
[\alpha']\lambda x.\mu\alpha[\beta']\lambda y.\mu\beta.u & \sim_\sigma & [\beta']\lambda y.\mu\beta[\alpha']\lambda x.\mu\alpha.u & x\neq y \quad (\texttt{pop/pop})\\
[\alpha'](\mu\alpha[\beta']\lambda x.\mu\beta.u)v & \sim_\sigma & [\beta']\lambda x.\mu\beta[\alpha'](\mu\alpha.u)v & \begin{cases}x\notin v\\ \beta\notin v\end{cases} \quad (\texttt{push/pop})
\end{array}
$$

with $\alpha \neq \beta$, $\alpha \neq \beta'$ and $\alpha' \neq \beta$ in these three equations.

- usual ρ- and θ-reductions of $\lambda\mu$-calculus.

$$
\begin{array}{llll}
[\beta]\mu\alpha.u & \sim_\sigma & u[{}^{\beta}/_{\alpha}] & & (\rho)\\
\mu\alpha[\alpha]u & \sim_\sigma & u & \alpha\notin u & (\theta)
\end{array}
$$

The ρ-reduction (resp. θ-reduction) is obtained by orienting (ρ) (resp. (θ)) from the left hand side to the right hand side.

These equations (except (ρ) and (θ)) can also be factorized through the notion of atomic context:

$$C_0[(\lambda x.u)v] \quad \sim_\sigma \quad (\lambda x.C_0[u])v \qquad (\sigma_i)$$

where C_0 is x-free and no free variable of v is bound in C_0,

$$N_0[N'_0[u]] \quad \sim_\sigma \quad N'_0[N_0[u]] \qquad (\texttt{p/p})$$

where if α (or x) is bound in N_0 (resp. N'_0) then N'_0 (resp. N_0) is α-free (or x-free).

We may justify these equations operationally by examining their behavior within Krivine's abstract machine [6]. Approximatively a state of this machine is a triple (t, e, s) where t is a term, e is an *environment* containing associations (λ-variable, term) and (μ-variable, stack) and s is a *stack*, that is a sequence of terms. The transitions are:

(jump) If t is a variable x, proceed with its value as defined in e.

(push) If t is an application $(u)v$, the argument v is pushed onto the stack and the machine proceeds with u.

(pop) If t is a λ-abstraction $\lambda x.u$, the first element of the stack is popped and stored with name x in the environment and the machine proceeds with u.

(store) If t is a μ-abstraction $\mu\alpha.u$, the stack is stored under the name α in the environment and execution continues with u and the empty stack ε.

(restore) If t is a named term $[\alpha]u$, the stack is replaced by a copy of the one associated to α in the environment and execution proceeds with u and the new stack.

We can, for example, describe the case of (push/pop) (which should enlighten its name). If we start with the state $([\alpha'](\mu\alpha[\beta']\lambda x.\mu\beta.u)v, e, \varepsilon)$ where e contains the associations: (α', s_1) and $(\beta', w :: s_2)$, the machine goes through the following steps:

- restore the stack s_1;
- push v on s_1;
- store the new stack $v :: s_1$ with name α;

- restore the stack $w :: s_2$;

- pop w and store it with name x;

- save the popped stack s_2 with name β.

The machine reaches the state (u, e', ε) where e' is e augmented with the associations $(\alpha, v :: s_1)$, (x, w) and (β, s_2). One easily checks that starting with the state $([\beta']\lambda x.\mu\beta[\alpha'](\mu\alpha.u)v, e, \varepsilon)$, the machine goes through the same steps.

3.2 Properties of the σ-equivalence

Proposition 2
Let u and v be two $\lambda\mu$-terms, if $u \sim_\sigma v$ then $u \sim_{\beta\eta\mu\rho\theta} v$.

PROOF: We prove that each equation of the σ-equivalence is realized by the $\beta\eta\mu\rho\theta$-equivalence:

- (σ_1), (σ_2) and (σ_3) are realized by the β-equivalence;
- (`push/push`) and (`push/pop`) are realized by the $\mu\rho$-equivalence;
- for the (`pop/pop`) equation we first show the following equivalence which corresponds to the (S_3) rule in [10]:

$$\begin{aligned} \mu\alpha' \ldots [\alpha']\lambda x.\mu\alpha.t \ldots &\sim \lambda x.(\mu\alpha' \ldots [\alpha']\lambda x.\mu\alpha.t \ldots)x && (\eta) \\ &\sim \lambda x.\mu\alpha' \ldots [\alpha'](\lambda x.\mu\alpha.t)x \ldots && (\mu) \\ &\sim \lambda x.\mu\alpha' \ldots [\alpha']\mu\alpha.t \ldots && (\beta) \\ &\sim \lambda x.\mu\alpha' \ldots t[^{\alpha'}/_{\alpha}] \ldots && (\rho) \end{aligned}$$

  where the μ-reduction also substitutes the other sub-terms $[\alpha']u$ by $[\alpha'](u)x$. If we apply two ρ-expansions in each member of the (`pop/pop`) equation we obtain

$$[\alpha']\mu\alpha'[\beta']\mu\beta'[\alpha']\lambda x.\mu\alpha[\beta']\lambda y.\mu\beta.u \quad \text{and} \quad [\alpha']\mu\alpha'[\beta']\mu\beta'[\beta']\lambda y.\mu\beta[\alpha']\lambda x.\mu\alpha.u$$

  and by applying the above equation twice to each term we get the same one

$$[\alpha']\lambda x.\mu\alpha'[\beta']\lambda y.\mu\beta' u[^{\alpha'}/_{\alpha}, ^{\beta'}/_{\beta}]$$

  This shows that (`pop/pop`) is realized by the $\beta\eta\mu\rho$-equivalence.
- (ρ) and (θ) are realized by the $\rho\theta$-equivalence. □

Proposition 3 (Preservation properties)
Let u and v be two $\lambda\mu$-terms such that $u \sim_\sigma v$.

- *If u is normalizable then v is normalizable and their normal forms are σ-equivalent.*
- *If u has a head normal form[5] then v has a head normal form.*
- *If u is strongly normalizable then v is strongly normalizable.*
- *If u is typable of type A then v is typable of type A.*

PROOF: All these properties are in fact corollaries of theorem 4. □

[5] It is the natural generalization of the notion of head normal forms for the λ-calculus, see [4] for a precise definition.

Let us look at the kind of transformations of typing derivations realized by the σ-equivalence, for example with the (push/pop) equation. If $\Gamma, x : A \vdash u \mid \alpha : C \to D, \beta : B, \Delta$ and $\Gamma' \vdash v : C \mid \Delta'$, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : A \vdash u \mid \alpha : C \to D, \beta : B, \Delta}{\Gamma, x : A \vdash \mu\beta.u : B \mid \alpha : C \to D, \Delta}\ \mu\text{-abs}}{\Gamma \vdash \lambda x.\mu\beta.u : A \to B \mid \alpha : C \to D, \Delta}\ \text{abs}}{\Gamma \vdash \mu\alpha[\beta']\lambda x.\mu\beta.u : C \to D \mid \beta' : A \to B, \Delta}\ \mu \qquad \Gamma' \vdash v : C \mid \Delta'}{\dfrac{\Gamma, \Gamma' \vdash (\mu\alpha[\beta']\lambda x.\mu\beta.u)v : D \mid \beta' : A \to B, \Delta, \Delta'}{\Gamma, \Gamma' \vdash [\alpha'](\mu\alpha[\beta']\lambda x.\mu\beta.u)v \mid \alpha' : D, \beta' : A \to B, \Delta, \Delta'}\ \mu\text{-name}}\ \text{app}$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : A \vdash u \mid \alpha : C \to D, \beta : B, \Delta}{\Gamma, x : A \vdash \mu\alpha.u : C \to D \mid \beta : B, \Delta}\ \mu\text{-abs} \qquad \Gamma' \vdash v : C \mid \Delta'}{\Gamma, \Gamma', x : A \vdash (\mu\alpha.u)v : D \mid \beta : B, \Delta, \Delta'}\ \text{app}}{\Gamma, \Gamma', x : A \vdash \mu\beta[\alpha'](\mu\alpha.u)v : B \mid \alpha' : D, \Delta, \Delta'}\ \mu}{\Gamma, \Gamma' \vdash \lambda x.\mu\beta[\alpha'](\mu\alpha.u)v : A \to B \mid \alpha' : D, \Delta, \Delta'}\ \text{abs}}{\Gamma, \Gamma' \vdash [\beta']\lambda x.\mu\beta[\alpha'](\mu\alpha.u)v \mid \alpha' : D, \beta' : A \to B, \Delta, \Delta'}\ \mu\text{-name}$$

The σ-equivalence realizes complex identifications in particular between normal and non normal terms even if we consider normal terms modulo ρ- and θ-reductions: let t be a closed term and $u = \lambda x.\mu\beta[\beta](\mu\alpha[\beta]\lambda y.\mu\delta[\alpha]x)t$, $v = \lambda x.\mu\beta[\beta]\lambda y.\mu\delta[\beta](x)t$, we have $u \sim_\sigma v$ (this is in fact a variant of (push/pop) and a particular case of lemma 9) but also $u \to^+ v$:

$$u \quad \to_\mu \quad \lambda x.\mu\beta[\beta]\mu\alpha[\beta]\lambda y.\mu\delta[\alpha](x)t \quad \to_\rho \quad v$$

Thus there is no hope that σ-equivalence preserves length of reduction as it does in λ-calculus. The reason is that $\lambda\mu$-calculus contains *linear* μ-redexes which have no real operational meaning. More precisely, we are now going to show that the σ-equivalence identifies terms which differ only by linear μ-redexes.

Definition 12 (Contexts)
A *context* C is a term, with a hole in place of a sub-term, defined in the following way:

$$C \quad ::= \quad [\,] \quad | \quad \lambda y.C \quad | \quad \mu\gamma[\beta]C \quad | \quad (C)u$$

A *named context* N is a named term, with a hole in place of a named sub-term:

$$N \quad ::= \quad [\,] \quad | \quad [\beta]C[\mu\gamma.[\,]]$$

The notions of variable *bound* in a context and of *α-free* and *x-free* contexts are the same as for atomic contexts.

Definition 13 (Linear μ-redex)
A μ-redex occurring in a $\lambda\mu$-term is *linear* if it has the shape $(\mu\alpha.N[[\alpha]u])v$ where N is an α-free named context and $\alpha \notin u$. The *size* of the μ-redex is the size of the term $\mu\alpha.N[[\alpha]u]$.

Lemma 8 (pop out)
If N is x- and α'-free, α not bound in N and $\alpha' \notin u$:

$$N[[\alpha]\lambda x.u] \sim_\sigma [\alpha]\lambda x.\mu\alpha'.N[[\alpha']u]$$

PROOF: By induction on N:

- If $N = [\,]$:

$$[\alpha]\lambda x.u \sim_\sigma [\alpha]\lambda x.\mu\alpha'[\alpha']u \qquad (\theta)$$

- If $N = [\beta]\mu\gamma.[\,]$:

$$\begin{aligned}
[\beta]\mu\gamma[\alpha]\lambda x.u &\sim_\sigma [\beta]\mu\gamma[\alpha]\lambda x.\mu\alpha'[\alpha']u && (\theta)\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\alpha']u[{}^\beta/_\gamma] && (\rho)\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\beta]\mu\gamma[\alpha']u && (\rho)
\end{aligned}$$

- If $N = [\beta]\lambda y.C[\mu\gamma.[\,]]$:

$$\begin{aligned}
[\beta]\lambda y.C[\mu\gamma.[\alpha]\lambda x.u] &\sim_\sigma [\beta]\lambda y.\mu\beta'[\beta']C[\mu\gamma.[\alpha]\lambda x.u] && (\theta)\\
&\sim_\sigma [\beta]\lambda y.\mu\beta'[\alpha]\lambda x.\mu\alpha'[\beta']C[\mu\gamma[\alpha']u] && \text{by induction}\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\beta]\lambda y.\mu\beta'[\beta']C[\mu\gamma[\alpha']u] && (\texttt{pop/pop})\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\beta]\lambda y.C[\mu\gamma[\alpha']u] && (\theta)
\end{aligned}$$

- If $N = [\beta]\mu\beta'[\gamma']C[\mu\gamma.[\,]]$:

$$\begin{aligned}
[\beta]\mu\beta'[\gamma']C[\mu\gamma[\alpha]\lambda x.u] &\sim_\sigma [\beta]\mu\beta'[\alpha]\lambda x.\mu\alpha'[\gamma']C[\mu\gamma[\alpha']u] && \text{by induction}\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\gamma']C[\mu\gamma[\alpha']u][{}^\beta/_{\beta'}] && (\rho)\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\beta]\mu\beta'[\gamma']C[\mu\gamma[\alpha']u] && (\rho)
\end{aligned}$$

- If $N = [\beta](C[\mu\gamma.[\,]])v$:

$$\begin{aligned}
[\beta](C[\mu\gamma[\alpha]\lambda x.u])v &\sim_\sigma [\beta](\mu\beta'[\beta']C[\mu\gamma[\alpha]\lambda x.u])v && (\theta)\\
&\sim_\sigma [\beta](\mu\beta'[\alpha]\lambda x.\mu\alpha'[\beta']C[\mu\gamma[\alpha']u])v && \text{by induction}\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\beta](\mu\beta'[\beta']C[\mu\gamma[\alpha']u])v && (\texttt{push/pop})\\
&\sim_\sigma [\alpha]\lambda x.\mu\alpha'[\beta](C[\mu\gamma[\alpha']u])v && (\theta)
\end{aligned}$$

□

Lemma 9 (push out)
If N is α'-free, if α is not bound in N, if $\alpha' \notin u$ and if none of the free variables of v is bound in N:

$$N[[\alpha](u)v] \sim_\sigma [\alpha](\mu\alpha'.N[[\alpha']u])v$$

In particular linear μ-reduction is included in σ-equivalence.

Proposition 4 (Elimination of linear μ-redexes)
Let u be a $\lambda\mu$-term, there exists u' such that $u' \sim_\sigma u$ and u' has no linear μ-redex.

In particular linear μ-reduction terminates.

PROOF: We prove that each linear μ-redex can be replaced by a smaller μ-redex or eliminated:

$$\begin{aligned}
(\mu\alpha.N[[\alpha]u])v &\sim_\sigma \mu\beta[\beta](\mu\alpha.N[[\alpha]u])v && (\theta)\\
&\sim_\sigma \mu\beta.N[[\beta](u)v] && \text{by lemma 9}
\end{aligned}$$

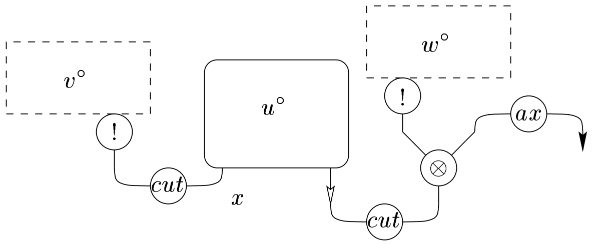
□

3.3 Completeness of the σ-equivalence

Theorem 4
Let t and t' be two $\lambda\mu$-terms, $t^\bullet = t'^\bullet \iff t \sim_\sigma t'$.

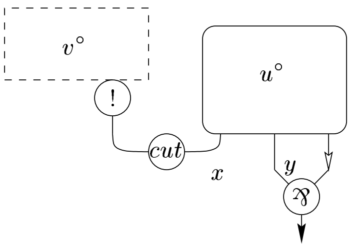
PROOF: Let t and t' be two σ-equivalent terms, we have to show that they have the same translations. To do this we just look at the proof-nets corresponding to the eight equations of σ-equivalence:

(σ_1) By multiplicative reductions, $(((\lambda x.u)v)w)^\circ$ and $((\lambda x.(u)w)v)^\circ$ yield the same proof-net:

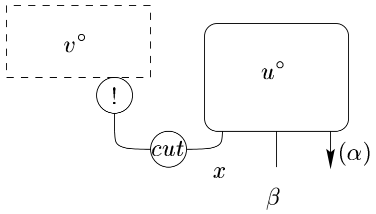


which eventually gives $t^\bullet = t'^\bullet$ by ending the multiplicative reduction.

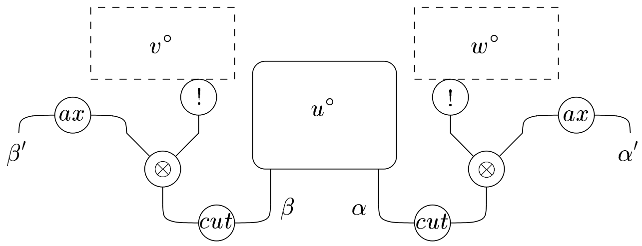
(σ_2) In the same way, by multiplicative reductions from $((\lambda x.\lambda y.u)v)^\circ$ and $(\lambda y.(\lambda x.u)v)^\circ$, we obtain the proof-net:



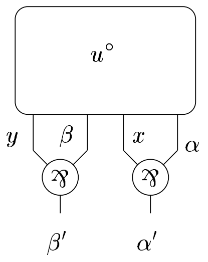
(σ_3) Idem with $((\lambda x.\mu\alpha[\beta]u)v)^\circ$ and $(\mu\alpha[\beta](\lambda x.u)v)^\circ$.



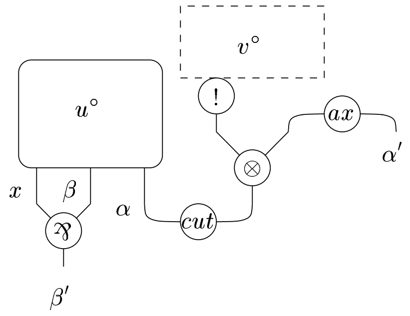
(push/push) The terms $([\alpha'](\mu\alpha[\beta'](\mu\beta.u)v)w)^\circ$ and $([\beta'](\mu\beta[\alpha'](\mu\alpha.u)w)v)^\circ$ immediately give the same proof-net:



(pop/pop) Idem with $([\alpha']\lambda x.\mu\alpha[\beta']\lambda y.\mu\beta.u)^\circ$ and $([\beta']\lambda y.\mu\beta[\alpha']\lambda x.\mu\alpha.u)^\circ$.



(push/pop) Idem with $([\alpha'](\mu\alpha[\beta']\lambda x.\mu\beta.u)v)^\circ$ and $([\beta']\lambda x.\mu\beta[\alpha'](\mu\alpha.u)v)^\circ$.



The last two cases are even simpler so we skip them.

For the converse we need some new lemmas.

Lemma 10
Let $[\beta]u$ be a named term containing a free μ-variable α such that the conclusion corresponding to α in $([\beta]u)^\bullet$ is neither an auxiliary door of a box nor the conclusion of a co-node. There exist an α-free context N and a $\lambda\mu$-term u_1 such that $[\beta]u = N[[\alpha]u_1]$ with $\alpha \notin u_1$.

PROOF: If $\beta = \alpha$ then $\alpha \notin u$ (otherwise we would have a contraction above the conclusion corresponding to α in $([\alpha]u)^\bullet$) and the result is proved with $N = [\,]$. If $\beta \neq \alpha$ we prove the lemma by induction on u:

- If $u = x$ then $\beta = \alpha$.
- If $u = \lambda x.u'$ then $[\beta]u'$ verifies the same hypothesis as $[\beta]u$ thus, by induction hypothesis, $[\beta]u' = N[[\alpha]u_1]$ with $N \neq [\,]$ since $\alpha \neq \beta$. We have $N = [\beta]C[\mu\gamma.[\,]]$ and $[\beta]u = [\beta]\lambda x.C[\mu\gamma[\alpha]u_1]$.
- If $u = \mu\gamma[\gamma']u'$ then $\gamma \neq \alpha$ (otherwise $\alpha \notin [\beta]u$). If $\gamma' = \alpha$ we have $N = [\beta]\mu\gamma.[\,]$. If $\gamma' \neq \alpha$, $[\beta]u'$ verifies the same hypothesis as $[\beta]u$ and, by induction, $[\beta]u' = N[[\alpha]u_1]$ with $N \neq [\,]$ thus $N = [\beta]C[\mu\beta'.[\,]]$ and $[\beta]u = [\beta]\mu\gamma[\gamma']C[\mu\beta'[\alpha]u_1]$.
- If $u = (u')v$ then α is not free in both u' and v otherwise the corresponding conclusion of $([\alpha]u)^\bullet$ would be conclusion of a *co*-node and $\alpha \notin v$ otherwise it would be an auxiliary door of a box. $[\beta]u'$ verifies the same hypothesis as $[\beta]u$ and, by induction, $[\beta]u' = N[[\alpha]u_1]$ with $N \neq [\,]$ thus $N = [\beta]C[\mu\gamma.[\,]]$ and $[\beta]u = [\beta](C[\mu\gamma[\alpha]u_1])v$. □

Lemma 11
If $u^\bullet$ has a final $⅋$-node n:

- *if n is above the distinguished conclusion then there exists a term u' such that $u \sim_\sigma \lambda x.u'$;*
- *if n is above another conclusion then there exists a term u' such that $u \sim_\sigma \mu\alpha[\beta]\lambda x.u'$.*

PROOF: We begin with the second point which is an easy consequence of the first one. Let β be the name of (the conclusion of) n and consider the term $v = \mu\beta[\alpha]u$ ($\alpha \notin u$). By definition of the translation, $v^\bullet$ is the same proof-net as $u^\bullet$ but has n as distinguished conclusion. By the first case, we obtain u' such that $v \sim_\sigma \lambda x.u'$. Thus $\mu\alpha[\beta]\mu\beta[\alpha]u \sim_\sigma \mu\alpha[\beta]\lambda x.u'$; furthermore $\mu\alpha[\beta]\mu\beta[\alpha]u \sim_\sigma u$ by (ρ) and (θ) because $\alpha \notin u$ so that $u \sim_\sigma \mu\alpha[\beta]\lambda x.u'$

We now turn to the first point which is proved by induction on the term u:

1. If $u = x$ then the distinguished conclusion must be the conclusion of an axiom node, a contradiction.
2. If $u = \lambda x.u_0$, the result is proved.
3. If $u = \mu\alpha[\beta]u_0$ then $\alpha \in [\beta]u_0$ since the distinguished conclusion of $u^\bullet$ is not below a *wo*-node. By lemma 10, we have $u = \mu\alpha.N[[\alpha]u_1]$ where N is an α-free context and $\alpha \notin u_1$. By induction hypothesis, $u_1 \sim_\sigma \lambda x.u'$ and by lemma 8,
$$\begin{aligned} u &\sim_\sigma \mu\alpha.N[[\alpha]\lambda x.u'] \\ &\sim_\sigma \mu\alpha[\alpha]\lambda x.\mu\alpha'.N[[\alpha']u'] \\ &\sim_\sigma \lambda x.\mu\alpha' N[[\alpha']u'] \end{aligned}$$
4. If $u = (u_0)v_1...v_n$ where $n > 0$ and u_0 is not an application, we look at the different cases for u_0:
   (a) $u_0 = x$, impossible as for 1.
   (b) $u_0 = \lambda x.u_1$ then $((u_1)v_2...v_n)^\bullet$ is a sub-proof-net of $u^\bullet$ which has the same final ⅋-node. By induction hypothesis: $(u_1)v_2...v_n \sim_\sigma \lambda y.u'$ (with y chosen not free in v_1) so
$$\begin{aligned} u &= (\lambda x.u_1)v_1...v_n \\ &\sim_\sigma (\lambda x.(u_1)v_2...v_n)v_1 && (\sigma_1) \\ &\sim_\sigma (\lambda x.\lambda y.u')v_1 \\ &\sim_\sigma \lambda y.(\lambda x.u')v_1 && (\sigma_2) \end{aligned}$$
   (c) $u_0 = \mu\alpha[\beta]u_1$ then $u^\bullet$ is obtained by the multiplicative reduction of a cut between $u_0^\bullet$ and the non flat $\otimes$-tree containing the $v_i^\bullet$'s. If the node above the distinguished conclusion of $u_0^\bullet$ is not a ⅋-node or an *ax*-node, this cut cannot be multiplicatively reduced and the distinguished conclusion of $u^\bullet$ is the main conclusion of the $\otimes$-tree which contradicts our hypothesis that it is conclusion of the ⅋-node n. By lemma 10, $u_0 = \mu\alpha.N[[\alpha]u_2]$ where N is an α-free context and $\alpha \notin u_2$. We use lemma 9:
$$\begin{aligned} u &= (\mu\alpha.N[[\alpha]u_2])v_1...v_n \\ &\sim_\sigma \mu\gamma[\gamma](\mu\alpha.N[[\alpha]u_2])v_1...v_n \\ &\sim_\sigma \mu\gamma.N[[\gamma](u_2)v_1...v_n] \end{aligned}$$
   then by induction hypothesis $(u_2)v_1...v_n \sim_\sigma \lambda x.u'$ and by lemma 8:
$$\begin{aligned} u &\sim_\sigma \mu\gamma.N[[\gamma]\lambda x.u'] \\ &\sim_\sigma \mu\gamma[\gamma]\lambda x.\mu\alpha.N[[\alpha]u'] \\ &\sim_\sigma \lambda x.\mu\alpha.N[[\alpha]u'] \end{aligned}$$

□

Lemma 12
If $u^\bullet$ has no final negative node and has a maximal cut-node with a non flat $\otimes$-tree above it, there exist u', v_1, … , v_n such that:

- *if the main conclusion of the $\otimes$-tree is distinguished then we have $u \sim_\sigma (\mu\alpha[\beta]u')v_1...v_n$;*
- *if the main conclusion of the $\otimes$-tree is not distinguished then $u \sim_\sigma \mu\alpha'[\beta'](\mu\alpha[\beta]u')v_1...v_n$.*

Lemma 13
If $u^\bullet$ has no final negative node and has a maximal cut-node with a flat $\otimes$-tree above it then there exist u' and v such that $u \sim_\sigma (\lambda x.u')v$.

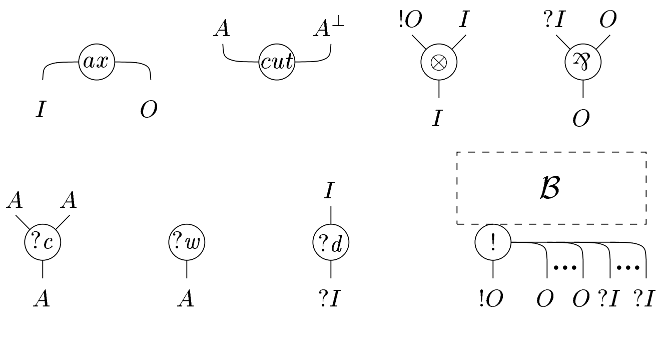
We are now able to finish the proof of theorem 4.

PROOF: (Theorem 4 – continued) By induction on $t^\bullet$, following the proof of theorem 3:

- If $t^\bullet$ has a final $⅋$-node n above the distinguished conclusion, by lemma 11, $t \sim_\sigma \lambda x.t_0$ and $t' \sim_\sigma \lambda x.t'_0$. By definition of the translation $t_0^\bullet = t^\bullet \setminus n = t'^\bullet \setminus n = t_0'^\bullet$, thus $t_0^\bullet = t_0'^\bullet$, by induction hypothesis $t_0 \sim_\sigma t'_0$ and $t \sim_\sigma t'$.
- If $t^\bullet$ has a final $⅋$-node n above another conclusion, by lemma 11, $t \sim_\sigma \mu\alpha[\beta]\lambda x.t_0$ and $t' \sim_\sigma \mu\alpha[\beta]\lambda x.t'_0$ so that $t_0^\bullet = t^\bullet \setminus n = t'^\bullet \setminus n = t_0'^\bullet$ thus by induction hypothesis $t_0 \sim_\sigma t'_0$ and $t \sim_\sigma t'$.
- If $t^\bullet$ has a final *co*-node or a final *wo*-node it is very similar.
- If $t^\bullet$ has a final *ci*-node n with name x, let t_0 (resp. t'_0) be a term with two free variables x_1 and x_2 such that $t_0{}^\bullet = t^\bullet \setminus n$ (resp. $t_0'^\bullet = t'^\bullet \setminus n$) where x_1 and x_2 are the names of the premises of n and such that $t_0[^x/_{x_1}, ^x/_{x_2}] = t$ (resp. $t'_0[^x/_{x_1}, ^x/_{x_2}] = t'$). By induction $t_0 \sim_\sigma t'_0$ thus we have $t \sim_\sigma t'$.
- $t^\bullet$ cannot have a final *wi*-node.
- If $t^\bullet$ has no final negative node but some *cut*-nodes, let c be a maximal one. If the $\otimes$-tree above the positive premise of c is not flat we apply lemma 12. If it is flat we apply lemma 13.
- If $t^\bullet$ has no final negative node and no cut then $t \sim_\sigma (x)u_1...u_n$ and $t' \sim_\sigma (x)v_1...v_n$ with $u_i \sim_\sigma v_i$ by induction hypothesis inside boxes thus $t \sim_\sigma t'$. □

4 Pure case

Considering pure proof-nets corresponds to applying the recursive equation $N = !N \multimap N$ on types. This identifies all output formulas and gives exactly four types: O (output formulas) and its dual I (anti output formulas); $?I$ (input formulas) and its dual $!O$ (anti input formulas).



where A is either O or $?I$.

The translations studied before can be seen as translations of pure $\lambda\mu$-calculus into pure polarized proof-nets extending those for λ-calculus in [1, 12].

All the results are still valid except, of course, strong normalization (corollary 1.2). In particular:

Proposition 5 (Confluence)
The reduction of pure polarized proof-nets is confluent.

Proposition 6 (σ-equivalence)
Let t and t' be two pure $\lambda\mu$-terms, then $t^\bullet = t'^\bullet \iff t \sim_\sigma t'$.

Acknowledgements

I would like to thank V. Danos for the key question of section 3 and L. Regnier for personal and mathematical help and support. Thanks also to the referee for the important comments and suggestions he made.

References

[1] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul).* Thèse de doctorat, Université Paris VII, 1990.

[2] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic: linear logic. *Journal of Symbolic Logic*, 62(3):755–807, September 1997.

[3] Vincent Danos and Laurent Regnier. Proof-nets and the Hilbert space. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 307–328. Cambridge University Press, 1995.

[4] René David and Walter Py. $\lambda\mu$-calculus and Böhm's theorem. To appear in *Journal of Symbolic Logic*, 2001.

[5] Jean-Yves Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[6] Jean-Louis Krivine. Un interpréteur du lambda-calcul. Unpublished.

[7] Olivier Laurent. Logique linéaire polarisée et logiques classiques. Mémoire du magistère M.M.F.A.I., Ecole Normale Supérieure de Paris, September 1999.

[8] Olivier Laurent. Polarized proof-nets: proof-nets for LC (extended abstract). In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications '99*, volume 1581 of *Lecture Notes in Computer Science*, pages 213–227. Springer, April 1999.

[9] Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of Internationnal Conference on Logic Programming and Automated Deduction*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

[10] Michel Parigot. Classical proofs as programs. In *Proceedings of Kurt Gödel Colloquium*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 1993.

[11] Myriam Quatrini and Lorenzo Tortora de Falco. Polarisation des preuves classiques et renversement. *Compte-Rendu de l'Académie des Sciences de Paris*, 323:113–116, 1996.

[12] Laurent Regnier. *Lambda-Calcul et Réseaux.* Thèse de doctorat, Université Paris VII, 1992.

[13] Laurent Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126:281–292, 1994.

[14] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. To appear in *Mathematical Structures in Computer Science*, 2001.