# Proof Nets and Explicit Substitutions

Roberto Di Cosmo[1], Delia Kesner[2], and Emmanuel Polonovski[1]

[1] Departement d'Informatique
Ecole Normale Supérieure
45, Rue d'Ulm
75230 Paris France
{dicosmo,polonovs}@ens.fr
[2] LRI (CNRS URA 410)
Bât 490, Université de Paris-Sud
91405 Orsay Cedex, France
kesner@lri.fr

**Abstract.** In this paper we refine the simulation technique introduced in [10] to show strong normalization of $\lambda$-calculi with explicit substitutions via termination of cut elimination in proof nets [13]. We first propose a notion of equivalence relation for proof nets that extends the one in [9], and we show that cut elimination modulo this equivalence relation is terminating. We then show strong normalization of the typed version of the $\lambda_l$-calculus with de Bruijn indices (a calculus with full composition defined in [8]) using a translation from typed $\lambda_l$ to proof nets. Finally, we propose a version of typed $\lambda_l$ with named variables which helps to better understand the complex mechanism of explicit weakening notation introduced in the $\lambda_l$-calculus with de Bruijn indices [8].

## 1 Introduction

This paper uses linear logic's proof nets, equipped with an extended notion of reduction, to provide several new results in the field of explicit substitutions. It is also an important step forward in clarifying the connection between explicit substitutions and proof nets, two well established formalisms that have been used to gain a better understanding of the $\lambda$-calculus over the past decade. On one side, explicit substitutions provide an intermediate formalism that - by decomposing the $\beta$ rule into more atomic steps, more similar to what happens in environment machines - allows a better understanding of the execution models. On the other side, linear logic decomposes the intuitionistic logical connectives, like the arrow, into more atomic, resource-aware connectives, like the linear arrow and the explicit erasure and duplication operators given by the exponentials: this decomposition is reflected in proof nets, which are the computational side of linear logic, and provide a more refined computational model that the one given by the $\lambda$-calculus, which is the computational side of intuitionistic logic[1].

The pioneer calculus with explicit substitutions, $\lambda_\sigma$, was introduced in [1] as a bridge between the classical $\lambda$-calculus and concrete implementations of functional programming languages. An important property of calculi with explicit substitutions is nowadays known as PSN, which stands for "Preservation of Strong Normalization": a calculus with explicit substitutions has PSN when all $\lambda$-terms that are strongly normalizing using the traditional $\beta$-reduction rule are also strongly normalizing w.r.t. the more refined reduction system defined using explicit substitutions.

[1] Using various translations of the $\lambda$-calculus into proof nets, new abstract machines have been proposed, exploiting the Geometry of Interaction and the Dynamic Algebras [14, 2, 5], leading to the works on optimal reduction [15, 17].

But $\lambda_\sigma$ does *not* preserve $\beta$-strong normalization as shown by Mellies, who exhibited a well-typed term which, due to the substitution composition rules in $\lambda_\sigma$, is not $\lambda_\sigma$-strongly normalizing [18].
Since then, a quest was started to find an "optimal" calculus having all of a wide range of desired properties: it should preserve strong normalization, but also be confluent (in a very large sense that implies the ability to compose substitutions), and its typed version should be strongly normalizing.
Meanwhile, in the linear logic community, many studies focused of the connection between $\lambda$-calculus (without explicit substitutions) and proof nets, trying to find the proper variant or extension of proof nets that could be used to cleanly simulate $\beta$-reduction, like in [7].
Finally, in [10], the first two authors of this work showed for the first time that explicit substitutions could be tightly related to linear logic's proof nets, by providing a translation into a variant of proof nets from $\lambda_x$ [19, 4], a simple calculus with explicit substitutions and named variables, but no composition.
This connection was promising because proof nets seem to have many of the properties which are required of a "good" calculus of explicit substitutions, and especially the strong normalization in the presence of a reduction rule which is reminiscent of the composition rule at the heart of Mellies' counterexample. But [10] only dealt with a calculus without composition, and the translation was complex and obscure enough to make the task of extending it to the case of a calculus with composition quite a daunting one.
In this paper, we can finally present a notion of reduction for Girard's proof nets which is flexible enough to allow a natural and simple translation from David and Guillaume's $\lambda_l$, a complex calculus of explicit substitution with de Bruijn indices and full composition [8]. This translation allows us to prove that typed $\lambda_l$ is strongly normalizing, which is a new result confirming a conjecture in [8]. Also, the fact that in the translation all information about variable order is lost suggests a version of typed $\lambda_l$ with named variables whose typed version is immediately proved to be strongly normalizing. This is due to the fact that only the type information is used in the translation of both calculi. Also, the typed named version of $\lambda_l$ gives a better understanding of the mechanisms of labels existing in the calculus. In particular, names allow to understand the fine manipulation of explicit weakenings in $\lambda_l$ without entering into the complicate details of renaming used in a de Bruijn setting.

The paper is organized as follows: we first recall the basic definitions on linear logic and proof nets and we introduce our refined reduction system for proof nets (Section 2), then prove that it is strongly normalizing (Section 3). In Section 4 we recall the definition of the $\lambda_l$ calculus with its type system, present the translation into proof nets, and show strong normalization of typed $\lambda_l$. Finally, we introduce a version of typed $\lambda_l$ with named variables (Section 5), enjoying the same good properties, and we conclude with some remarks and directions for future work (Section 6).

## 2 Linear logic, proof nets and extended reduction

We recall here some classical notions from linear logic, namely the linear sequent calculus and proof nets, and some basic results concerning confluence and normalization.

*MELL: Multiplicative Exponential linear logic* Let $\mathcal{A}$ be a set of *atomic formulae*. We suppose that $\mathcal{A}$ is partitioned in two disjoint subsets representing *positive* and *negative* atoms respectively. For every $p \in \mathcal{A}$, we assume that there is $p' \in \mathcal{A}$, called the *linear negation of the atom* $p$.
The set of formulae of the Multiplicative Exponential fragment of linear logic (called MELL) is defined by the following grammar, where $a \in \mathcal{A}$:

$$\mathcal{F} ::= a \mid \mathcal{F} \otimes \mathcal{F} \text{ (tensor)} \mid \mathcal{F} \parr \mathcal{F} \text{ (par)} \mid !\mathcal{F} \text{ (of course)} \mid ?\mathcal{F} \text{ (why not)}$$

Linear negation is *defined* as follows

$$p^\perp = p' \qquad p'^\perp = p \qquad A^{\perp\perp} = A \qquad (?A)^\perp = !(A^\perp) \qquad (A \otimes B)^\perp = A^\perp \parr B^\perp$$

From now on we will write $p^\perp$ instead of $p'$.

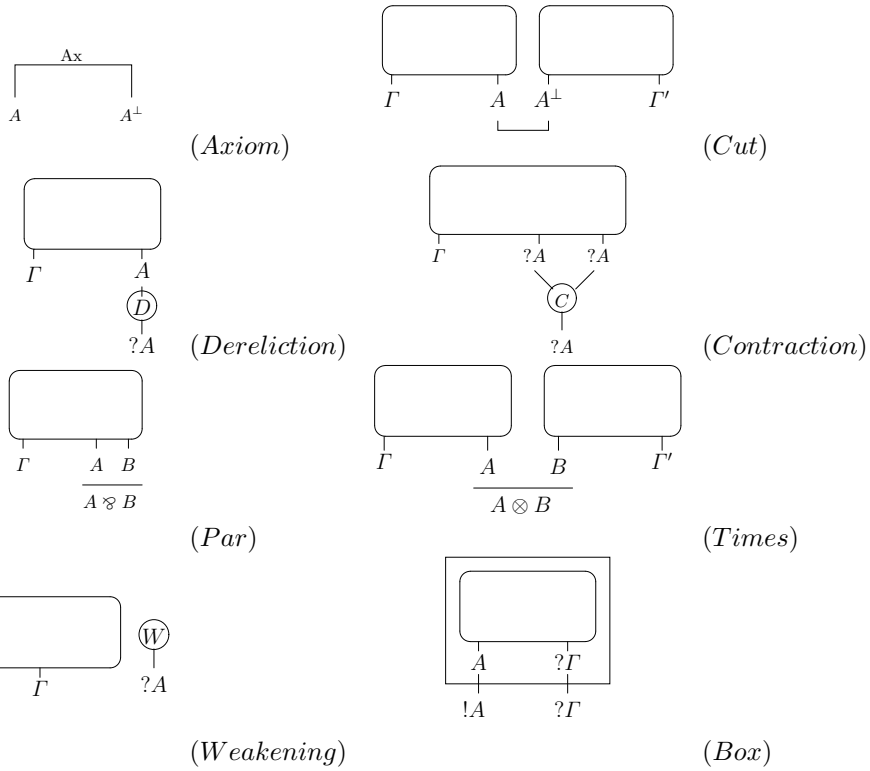
The name of MELL comes from the connectors : $\otimes$ and $\parr$ are called "multiplicatives", while ! and ? are called "exponentials". We will also say that a formula is exponential if it starts with an exponential connector. While we refer the interested reader to [13] for more details on linear logic in general, we give here a one-sided presentation of the sequent calculus for MELL:

$$\frac{}{\vdash A, A^\perp}\ Axiom \qquad \frac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}\ Cut \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ Dereliction \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ Contraction$$

$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ Par \qquad \frac{\vdash \Gamma, A \quad \vdash B, \Gamma'}{\vdash \Gamma, A \otimes B, \Gamma'}\ Times \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ Weakening \qquad \frac{\vdash A, ?\Gamma}{\vdash !A, ?\Gamma}\ Box$$

*MELL proof nets* To all sequent derivations in MELL it is possible to associate an object called a "proof net", which allows to abstract from many inessential details in a derivation, like the order of application of independent logical rules: for example, there are many inessentially different ways to obtain $\vdash A_1 \parr A_2, \ldots, A_{n-1} \parr A_n$ from $\vdash A_1, \ldots A_n$, while there is only one proof net representing all these derivations.

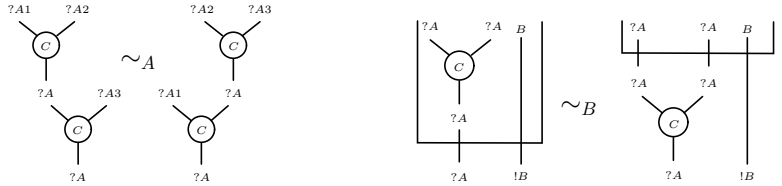
Proof nets are defined inductively by rules that follow closely the ones of the one-sided sequent calculus, and the set of proof nets is denoted $PN$. To simplify the drawing of a proof net, we use the following notation: a conclusion with a capital greek letter $\Gamma, \Delta, \ldots$ really stands for a set of conclusions, each one with its own wire.




Each box has exactly one conclusion preceded by a !, which is named "principal" port (or formula), while the other conclusions are named "auxiliary" ports (or formulae). In what follows, we will sometimes write an axiom link as $\overline{A \qquad A^\perp}$.

*Reduction of proof nets* Proof nets are the "computational object" behind linear logic, because there is a notion of reduction on them (called also "cut elimination") that corresponds to the cut-elimination procedure on sequent derivations. The traditional reduction system for MELL is recalled in Appendix A.

*Extended reduction modulo an equivalence relation* Unfortunately, the original notion of reduction on $PN$ is not well adapted to simulate neither the $\beta$ rule of $\lambda$-calculus, nor the rules dealing with propagation of substitution in explicit substitution calculi: too many inessential details on the order of application of the rules are still present, and to make abstraction from them, one is naturally led to define an equivalence relation on $PN$, as is done in [9], where the following two equivalences are introduced:



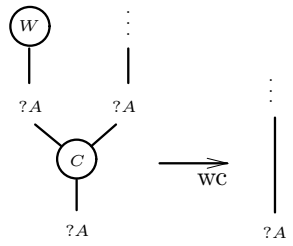

Equivalence $A$ turns contraction into an associative operator, and corresponds to forgetting the order in which the contraction rule is used to build, for example, the derivation:

$$\dfrac{\dfrac{\vdash ?A, ?A, ?A}{\vdash ?A, ?A}\ Contraction}{\vdash ?A}\ Contraction$$

Equivalence $B$ abstracts away the relative order of application of the rules of box-formation and contraction on the premises of a box, like in the following example.

$$\dfrac{\dfrac{\vdash ?A, ?A, B}{\vdash ?A, B}\ Contraction}{\vdash ?A, !B}\ Box \qquad\qquad \dfrac{\dfrac{\vdash ?A, ?A, B}{\vdash ?A, ?A, !B}\ Box}{\vdash ?A, !B}\ Contraction$$

Finally, besides the equivalence relation defined in [9], for the sake of simulating explicit substitutions, we will also need an extra reduction rule allowing to remove unneeded weakening links:




This rule allows to simplify the proof below on the left into the proof on the right

$$\dfrac{\dfrac{\dfrac{\pi}{\vdash ?A}}{\vdash ?A, ?A}\ Weakening}{\vdash ?A}\ Contraction \qquad\qquad \dfrac{\pi}{\vdash ?A}$$

*Notation* We will call in the following $R$ the system made of rules $Ax - cut$, $\parr - \otimes$, $w - b$, $d - b$,$c - b$, $b - b$ and $wc$; we will name $E$ the relation induced on $PN$ by the contextual closure of axioms $A$ and $B$; we will write $R_E$ for the system made of the rules in $R$ and the

equivalences in $E$; finally, $R_E^{\neg wc}$ will stand for system $R_E$ *without* rule $wc$.
Systems $R_E$ and $R_E^{\neg wc}$, that contain $E$, are actually defining a notion of *reduction modulo an equivalence relation*, so we write for example $t \longrightarrow_{R_E} s$ if and only if there exist $r'$ and $s'$ such that $r =_E r' \longrightarrow_R s' =_E s$, where the equality $=_E$ is the reflexive, symmetric and transitive closure of the relation defined by $A$ and $B$.

The reduction $R_E$ is flexible enough to allow a clean simulation of $\beta$ reduction and of explicit substitutions, but we first need to establish that $R_E$ is strongly normalizing. We will do so in the next section by building on the strong normalization result for $R_E^{\neg wc}$ shown in [9].

## 3 Termination of $R_E$

We know from [9] that $R_E^{\neg wc}$ is terminating, and we can show easily that $wc$ is terminating too, so if we could show that the $wc$-rule can be postponed with respect to all the other rules of $R_E^{\neg wc}$, we would be easily done using a well-known abstract lemma. Unfortunately, there is precisely one case in which we cannot postpone the $wc$-rule: when a $wc$ reduction creates an axiom-cut redex, which in turn can only happen if the axiom link in question introduces an exponential formula. So we are forced to proceed in two steps: first, we prove by postponement that $R_E$ is terminating on the set of proof nets *without exponential axioms* (Theorem 1). Then, we show that termination of $R_E$ on all proof nets of $PN$ is a consequence of termination of $R_E$ on proof nets without exponential axioms (Theorem 2). To obtain this last result, we show how to translate a proof net $R$ with exponential axioms into a proof net $R'$ without exponential axioms in such a way that a reduction out of $R$ can be simulated by a longer or equal reduction out of $R'$.

### 3.1 Termination of $R_E$ on proof nets without exponential axioms

We show in this section that all the $R_E$-reduction sequences from a proof net without exponential axioms terminate. We first remind the following result from [9]:

**Lemma 1 (Termination of $R_E^{\neg wc}$).** *The reduction relation $\longrightarrow_{R_E^{\neg wc}}$ is terminating on $PN$.*

Then, we establish the termination of $wc$.

**Lemma 2 (Termination of $wc$).** *The reduction relation $\longrightarrow_{wc}$ is terminating on $PN$.*

*Proof.* The $wc$-rule strictly decreases the number of nodes in a proof net so no infinite $wc$-reduction sequence is possible.

Finally, we show that given any proof net without exponential axioms, the $wc$-rule can be postponed with respect to any rule of $R_E^{\neg wc}$.

**Lemma 3 (Postponement of $wc$ w.r.t $R_E^{\neg wc}$).** *Let $t$ be a proof net without exponential axioms. If $t \longrightarrow_{wc} \longrightarrow_{R_E^{\neg wc}} t'$, then, there is a sequence $t \longrightarrow^+_{R_E^{\neg wc}} \longrightarrow^*_{wc} t'$.*

*Proof.* By analyzing all the possible cases. See [11] for details.

We can now put together the previous results to prove termination of $R_E$ on the set of proof nets without exponential axioms.

**Lemma 4 (Extraction of $R_E^{\neg wc}$).** *Let $S$ be an infinite sequence of $R_E$-reductions starting at a proof net $t$ without exponential axioms. Then, there is a sequence of $R_E$-reductions from the same proof net $t$ which starts by $t \longrightarrow_{R_E^{\neg wc}} t'$, where $t'$ is also a proof net without exponential axioms, and which continues with an infinite sequence $S'$. We write this sequence as $(t \longrightarrow_{R_E^{\neg wc}} t') \cdot S'$.*

Now it is easy to establish the fundamental theorem of this section:

**Theorem 1 (Termination of $R_E$ on proof nets without exponential axioms).** *The reduction relation $R_E$ is terminating on the set of proof nets without exponential axioms.*

*Proof.* We show it by contradiction. Let us suppose that $R_E$ is not terminating on those nets. Then, there exist a proof net without exponential axioms $t$ and an infinite sequence $S$ of $R_E$ starting at $t$. By applying Lemma 4 to this sequence $S$, we obtain a sequence $(t \longrightarrow_{R_E^{\neg wc}} t') \cdot S'$ such that $S'$ is infinite again. If we iterate this procedure an arbitrary number times, we obtain a sequence of $R_E^{\neg wc}$-reduction steps arbitrary long. This contradicts the fact that $R_E^{\neg wc}$ is terminating.

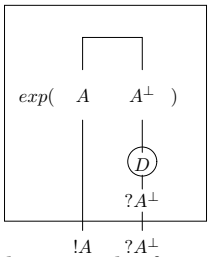
### 3.2 Termination of $R_E$ on proof nets with exponential axioms

We know now that $R_E$ is terminating on every proof net without exponential axioms, but we want now to show even more: termination of $R_E$ on *all* the proof nets. To achieve this result, we show in this section how to associate to a proof net $t$, which can eventually contain some exponential axioms, another proof net $E(t)$ without exponential axioms, and such that every reduction from $t$ of length $n$ can be "simulated" on $E(t)$ by another reduction of length at least $n$. This property will be enough to reduce termination of $R_E$ on proof nets with exponential axioms to termination of $R_E$ on proof nets without exponential axioms.

We define in what follows a notion of complete expansion on axiom links that is able to replace all exponential axiom by a correct net with the same conclusions, but containing no exponential axiom, and then extend it to a full proof net in the natural way (replace each exponential axiom by its complete expansion).

**Definition 1 (Complete expansion of an axiom link).** *For each axiom link $\overline{A \qquad A^\perp}$ we can associate a net $exp(\overline{A \qquad A^\perp})$ with same conclusions, defined by induction on the complexity of the formula $A$ as follows:*

- $exp(\overline{A \qquad A^\perp}) = \overline{A \qquad A^\perp}$, if $A$ is not an exponential formula
- $exp(\overline{!A \qquad ?A^\perp}) =$




  which is well defined, because the formula $A$ is smaller than $!A$.

We can associate a complexity measure $rk$ to a complete expansion.

**Definition 2 (Measure of a complete expansion).** *We define the measure $rk$ of a complete expansion of an axiom by cases:*

- $rk(exp(\overline{A \qquad A^\perp})) = 0$, if $A$ is not an exponential formula
- $rk(exp(\overline{?A^\perp \qquad !A})) = 1 + rk(exp(\overline{A \qquad A^\perp}))$

We can now define the notion of expanded net $E(t)$ for every net $t$:

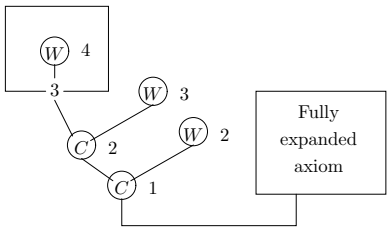
**Definition 3 (Expanded net).** *Let $t$ be a MELL net. We call* expanded net of $t$*, written $E(t)$, the proof net obtained from $t$ by replacing each occurrence of an exponential axiom $a$ by $exp(a)$.*

*Remark 1.* The only difference between a proof net $t$ and its expanded net $E(t)$ is on the set of their axioms. So, for every reduction $t \longrightarrow_{R_E} t'$ which does not affect the axioms of $t$, there is a reduction $E(t) \longrightarrow_{R_E} E(t')$.

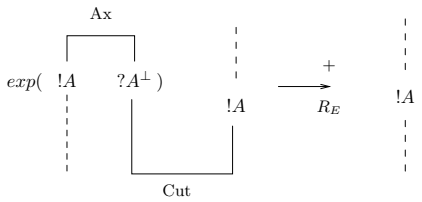
We have now to show that there is no problem for the axioms either, and to do so we need the following measure:

**Definition 4 (Maximal distance of a cut).** *Given a proof net $t$ and a cut link on a completely expanded axiom $a$ in $t$, the measure $d(a,t)$ is the maximal distance, in the proof net $t$, between this cut and the first weakening or dereliction node encountered in the way which leaves the cut, by the opposite extremity from the expanded axiom $a$, and go throw the nodes from down to up (here up and down are used formally for the orientation of the nodes presented in the introduction). More precisely, each node encountered and each box passed on the way values* 1*, including the final dereliction or weakening node. This measure is always finite on a finite proof net because there are no arbitrary long ascendant ways.*

*Example 1.* In the following net, the maximal distance of the cut is 4.




**Lemma 5 (Cut elimination on an expanded net).** *Let $t$ be an expanded net. A cut in $t$ with a completely expanded axiom $exp(a)$ reduces in $t$ like in an ordinary axiom cut. In other words,*




*Proof.* We prove the property by induction on the lexicographic order $(rk(exp(a)), d(exp(a),t))$ where $exp(a)$ is the completely expanded axiom in the proof net $t$.

All the cases such that $rk(exp(a)) = 0$ (including the base case) correspond to a proof net in which $exp(a)$ is just an axiom link, so that we can apply the same reduction rule and the property then trivially holds. For the cases with $rk(exp(a)) > 0$, we refer the interested reader to [11].

This allows us to establish the final result of this section :

**Theorem 2 (Termination of $R_E$).** *The reduction $R_E$ is terminating on every proof net $t$.*

*Proof.* We establish this result by proving that each reduction step $t \longrightarrow_{R_E} t'$ can be simulated by at least one reduction step $E(t) \longrightarrow^{+}_{R_E} E(t')$.

If the reduction step $t \longrightarrow_{R_E} t'$ does not reduce any exponential axiom with a cut, then we obtain the result immediately because the only difference between $t$ and $E(t)$ is on their axioms. Indeed, we can reproduce the same reduction on $E(t)$ in order to obtain $E(t')$ and this concludes this case.

Otherwise, if $t \longrightarrow_{R_E} t'$ reduces an exponential axiom $a$ with a cut then by Lemma 5 there exist a non-empty sequence of reductions starting at $E(t)$ which eliminates the complete expansion of the axiom $a$, and gives the proof net $E(t')$.

Now, to conclude the proof, suppose that there is a proof net $t$ such that the reduction $R_E$ is not terminating on $t$, that is, there is an infinite $R_E$-reduction sequence starting at $t$. By the previous remark we can simulate this infinite reduction sequence by another $R_E$-reduction sequence on expanded proof nets not containing exponential axioms. This leads to a contradiction with Theorem 1 so that we can conclude that $R_E$ is terminating on the set of all proof nets.

## 4 From $\lambda_l$ with de Bruijn indices to $PN$

In this section we study the translation of the typed terms in the $\lambda_l$-calculus [8] into proof nets in $PN$. We start by introducing the calculus, then we give the translation of types of $\lambda_l$ into formulae of linear logic, and the translation of terms of $\lambda_l$ into linear logic proof nets $PN$. We verify that we can correctly simulate every reduction step of $\lambda_l$ via the notion of reduction $R_E$. Finally, we use this simulation result to show strong normalization of the $\lambda_l$-calculus.

### 4.1 The $\lambda_l$-calculus

The $\lambda_l$-calculus is a calculus with explicit substitutions where substitutions are unary (and not multiple). The version studied in this section has variables encoded with de Bruijn indices. The terms of $\lambda_l$ are given by the following grammar:

$$M ::= \underline{\mathrm{n}} \mid \lambda M \mid (MM) \mid \langle k \rangle M \mid [i/M, j]M$$

The term $\underline{\mathrm{n}}$ is called a *variable,* $\lambda M$ *an abstraction,* $(MM)$ *an application,* $\langle k \rangle M$ *a labeled term and* $[i/M, j]M$ *a substitution.*

Intuitively, the term $\langle k \rangle M$ means that the $k-1$ first indices in $M$ are not "free" (in the sense of free variables of calculus with indices). The term $[i/N, j]M$ means that the $i-1$ first indices are not free in $N$ and that the $j-1$ following indices are not free in $M$. Those indices are used to split the typing environment in three parts : the first one for free variables of $M$, the second one for free variables of $N$ and the third one for free variables which are in both terms.

The de Bruijn indices we use start with $\underline{0}$ instead of $\underline{1}$. For example, the identity function is written as $I = \lambda\underline{0}$.

The reduction rules of $\lambda_l$ are given in Figure 1:

The typing rules of $\lambda_l$ are given in Figure 2, where we suppose that $|\Gamma| = i$ and $|\Delta| = j$.

We notice that for each well-typed term of the $\lambda_l$-calculus, there is only one possible typing judgment. This will simplify the proof of simulation of $\lambda_l$ by easily considering the unique typing judgment of terms.

As expected the $\lambda_l$-calculus enjoys the subject reduction property (see [16] for a detailed proof).

**Theorem 3 (Subject Reduction).** *If* $\Psi \vdash M : C$ *and* $M \longrightarrow M'$*, then* $\Psi \vdash M' : C$.

$$
\begin{array}{lrcll}
(b_1) & (\lambda MN) & \longrightarrow & [0/N,0]M & \\
(b_2) & (\langle k\rangle(\lambda M)N) & \longrightarrow & [0/N,k]M & \\
(f) & [i/N,j](\lambda M) & \longrightarrow & \lambda[i+1/N,j]M & \\
(a) & [i/N,j](MP) & \longrightarrow & ([i/N,j]M)([i/N,j]P) & \\
(e_1) & [i/N,j]\langle k\rangle M & \longrightarrow & \langle j+k-1\rangle M & if\ i<k \\
(e_2) & [i/N,j]\langle k\rangle M & \longrightarrow & \langle k\rangle[i-k/N,j]M & if\ i\geq k \\
(n_1) & [i/N,j]\underline{\mathrm{k}} & \longrightarrow & \underline{\mathrm{k}} & if\ i>k \\
(n_2) & [i/N,j]\underline{\mathrm{i}} & \longrightarrow & \langle i\rangle N & \\
(n_3) & [i/N,j]\underline{\mathrm{k}} & \longrightarrow & \underline{\mathrm{j+k\text{-}1}} & if\ i<k \\
(c_1) & [i/N,j][k/P,l]M & \longrightarrow & [k/[i-k/N,j]P,j+l-1]M & if\ k\leq i<k+l \\
(c_2) & [i/N,j][k/P,l]M & \longrightarrow & [k/[i-k/N,j]P,l][i-l+1/N,j]M & if\ i\geq k+l \\
(d) & \langle i\rangle\langle j\rangle M & \longrightarrow & \langle i+j\rangle M &
\end{array}
$$

**Fig. 1.** Reduction rules of $\lambda_l$ with de Bruijn indices

$$
\frac{}{\Gamma, A, \Delta \vdash \underline{\mathrm{i}} : A}\ Axiom
\qquad
\frac{\Delta \vdash M : B}{\Gamma, \Delta \vdash \langle i\rangle M : B}\ Weak
$$

$$
\frac{\Gamma \vdash M : B \to A \quad \Gamma \vdash N : B}{\Gamma \vdash (MN) : A}\ App
\qquad
\frac{B, \Gamma \vdash M : C}{\Gamma \vdash \lambda M : B \to C}\ Lambda
$$

$$
\frac{\Delta, \Pi \vdash N : A \quad \Gamma, A, \Pi \vdash M : B}{\Gamma, \Delta, \Pi \vdash [i/N,j]M : B}\ Subst
$$

**Fig. 2.** Typing rules for $\lambda_l$ with de Bruijn indices

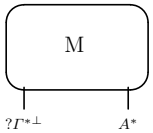
### 4.2 Translation of types and terms of $\lambda_l$

We use the translation of types introduced in [6] given by :

$$
\begin{array}{ll}
A^* = A & \text{if } A \text{ is an atomic type} \\
(A \to B)^* = ?((A^*)^\perp) \parr\, !B^* & (\text{that is, } !A^* \multimap !B^*) \text{ otherwise}
\end{array}
$$

Since wires are commutative in proof nets, we feel free to exchange them when we define the translation of a term. The translation associates to every typed term $M$ of $\lambda_l$, whose type judgment ends with the conclusion written below on the left, a proof net having the shape sketched below on the right:

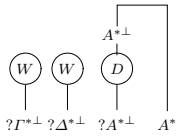
$$\frac{}{\Gamma \vdash M : A}$$




Here is the formal definition of the translation $T$ from $\lambda_l$-terms into proof nets.

– If the term is a variable and its type judgment ends with the rule written below on the left, then its translation is the proof net on the right

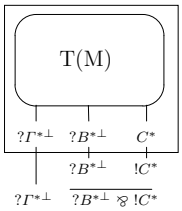
$$\frac{}{\Gamma, A, \Delta \vdash \underline{\mathrm{i}} : A}\ Axiome$$



where $i$ is the position of $A$ in the typing environment,

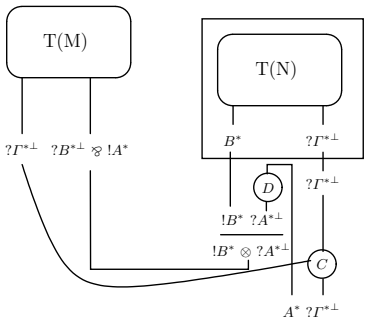
– If the term is a $\lambda$-abstraction and its type judgment ends with the rule written below on the left, then its translation is the proof net on the right

$$\frac{B, \Gamma \vdash M : C}{\Gamma \vdash \lambda M : B \to C}\ Lambda$$



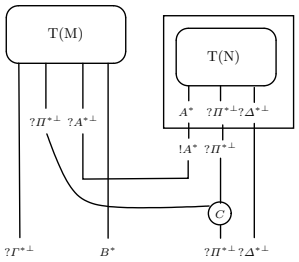

– If the term is an application and its type judgment ends with the rule written below on the left, then its translation is the proof net on the right

$$\frac{\Gamma \vdash M : B \to A \quad \Gamma \vdash N : B}{\Gamma \vdash (MN) : A}\ App$$




– If the term is a substitution and its type judgment ends with the rule written below on the left, then its translation is the proof net on the right

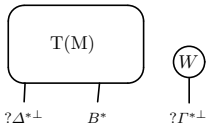
$$\frac{\Delta, \Pi \vdash N : A \quad \Gamma, A, \Pi \vdash M : B}{\Gamma, \Delta, \Pi \vdash [i/N, j]M : B}\ Subst$$




where $i$ is the length of the list $\Gamma$ and $j$ is the length of the list $\Delta$, then its translation is the proof net

– Finally, if the term is a label and its type judgment ends withthe rule written below on the left, then its translation is the proof net on the right

$$\frac{\Delta \vdash M : B}{\Gamma, \Delta \vdash \langle i \rangle M : B}\ Weak$$




where $i$ is the length of the list $\Gamma$, then its translation is the proof net

### 4.3 Simulating $\lambda_l$-reduction

We now verify that our notion of reduction $R_E$ on $PN$ simulates the $\lambda_l$-reduction on typed $\lambda_l$-terms. It is in this proof that we find the motivation for our choice of translation from $\lambda$-terms into proof nets: with the more traditional translation sending the intuitionistic type $A \to B$ into the linear $!A \multimap B$, the simulation of the rewrite rule $f$ would give rise to an equality, not to a reduction step like in this paper.

**Lemma 6 (Simulation of $\lambda_l$).** *The relation $R_E$ simulates the $\lambda_l$-reduction on typed terms: if $t \longrightarrow_{\lambda_l} t'$, then $T(t) \longrightarrow^+_{R_E} T(t')$, excepted for the rules $e_2$ and $d$ for which we have $T(t) = T(t')$.*

*Proof.* The proof procedes by cases on the reduction rule applied in the step $t \longrightarrow_{\lambda_l} t'$. Since reductions $\lambda_l$ and $R_E$ are closed under all contexts, we only need to study the cases where reduction takes place at the head position of $t$. In the proof, rule $wc$ is used to simulate $b_2, e_1, n_1, n_2, n_3$, equivalence $A$ is used to simulate $a, c_1, c_2$, and equivalence $B$ is used to simulate $f, a, c_1, c_2$.

Due to space limitations, we cannot give the proof here, that the interested reader can find fully developed in [11], but we show anyway the case of rule $c_1$, one of the composition rules:

$$[i/N, j][k/P, l]M \longrightarrow [k/[i-k/N, j]P, j+l-1]M \text{ if } k \leq i < k+l$$

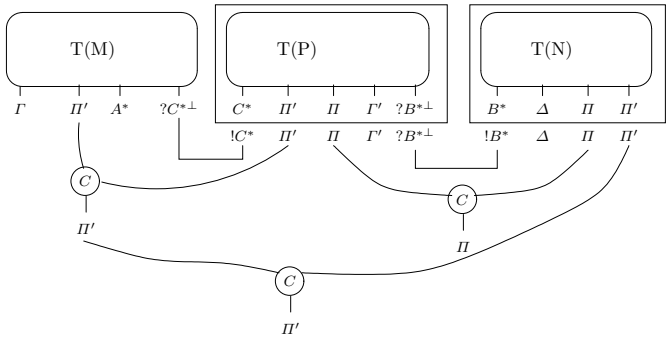
Here, the typing judgment of $[i/N, j][k/P, l]M$ must end with

$$\dfrac{\Delta, \Pi, \Pi' \vdash N : B \qquad \dfrac{\Gamma', B, \Pi, \Pi' \vdash P : C \quad \Gamma, C, \Pi' \vdash M : A}{\Gamma, \Gamma', B, \Pi, \Pi' \vdash [k/P, l]M : A}\ Subst}{\Gamma, \Gamma', \Delta, \Pi, \Pi' \vdash [i/N, j][k/P, l]M : A}\ Subst$$

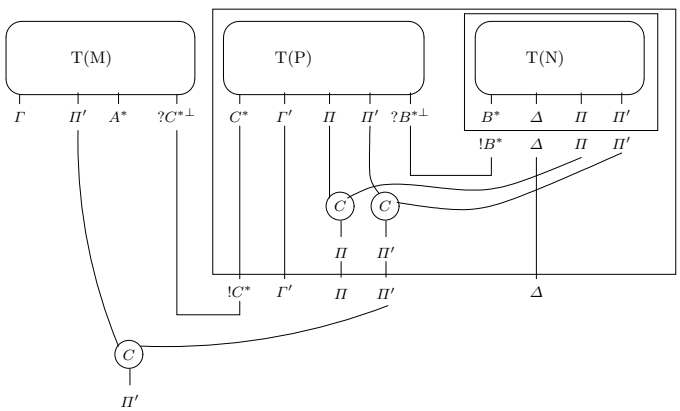
while the typing judgment of $[k/[i-k/N, j]P, j+l-1]M$ must end with

$$\dfrac{\dfrac{\Delta, \Pi, \Pi' \vdash N : B \quad \Gamma', B, \Pi, \Pi' \vdash P : C}{\Gamma', \Delta, \Pi, \Pi' \vdash [i-k/N, j]P : C}\ Subst \qquad \Gamma, C, \Pi' \vdash M : A}{\Gamma, \Gamma', \Delta, \Pi, \Pi' \vdash [k/[i-k/N, j]P, j+l-1]M : A}\ Subst$$

So, the translation of the type derivation of the first term is




while the translation of the second derivation is



To reduce the first proof net into the second one, we must eliminate the $b-b$ cut, then apply the equivalence relations $A$ and $B$.

We are now able to show strong normalization of $\lambda_l$. To achieve this result, we use the following abstract theorem (see for example [12]) :

**Theorem 4.** *Let $R = \langle \mathcal{O}, R_1 \cup R_2 \rangle$ be an abstract reduction system such that $R_2$ is strongly normalizing and there exist a reduction system $S = \langle \mathcal{O}', R' \rangle$, with a translation $T$ of $\mathcal{O}$ into $\mathcal{O}'$ such that $a \longrightarrow_{R_1} b$ implies $T(a) \longrightarrow^+_{R'} T(b)$; $a \longrightarrow_{R_2} b$ implies $T(a) = T(b)$. Then if $R'$ is strongly normalizing, $R_1 \cup R_2$ is also strongly normalizing.*

If we take $\mathcal{O}$ as the set of typed $\lambda_l$-terms, $R_1$ as $\lambda_l - \{e_2, d\}$, $R_2$ as $\{e_2, d\}$, $\mathcal{O}'$ as the set of proof nets and $R'$ as the reduction $R_E$, then, by the Theorem 4 and the fact that the system including the rules $\{e_2, d\}$ is strongly normalizing [8], we can conclude :

**Theorem 5 (Strong normalization of $\lambda_l$).** *The typed $\lambda_l$-calculus is strongly normalizing.*

## 5 The $\lambda_l$-calculus with names

In this section we present a version of typed $\lambda_l$ with named variables. We first introduce the grammar of terms, then the typing and reduction rules and finally we will briefly discuss the translation of this syntax to $PN$ in order to show strong normalization of the associated reduction.

The terms of this calculus are given by the following grammar:

$$M ::= x \mid \lambda x.M \mid (MM) \mid \Delta M \mid M[x, M, \Gamma, \Delta]$$

The term $x$ is called a *variable*, $\lambda x.M$ an *abstraction*, $(MM)$ an *application*, $\Delta M$ a *labeled term* and $M[x, M, \Gamma, \Delta]$ a substitution. Intuitively, the term $\Delta M$ means that the variables in $\Delta$ are not in $M$, and the term $M[x, N, \Gamma, \Delta]$ means that the variables in $\Gamma$ do not appear in $N$ (they only belong to the type environment of $M$) and the variables $\Delta$ do not appear in $M$ (they only belong to the type environment of $N$).

Variables are bound by the abstraction and substitution operators, so that for example $x$ is bound in $\lambda x.x$ and in $x[x, N, \Gamma, \Delta]$.

Terms are identified modulo $\alpha$-conversion so that bound variables can be systematically renamed. Indeed, we have $\lambda y.y[x, z, \emptyset, \emptyset] =_\alpha \lambda y'.y'[x, z, \emptyset, \emptyset]$ and $\lambda y.y[x, z, \emptyset, \emptyset] =_\alpha \lambda y.y[x', z, \emptyset, \emptyset]$ and $\lambda l.y[x, z, \{l\}, \emptyset] =_\alpha \lambda l'.y[x, z, \{l'\}, \emptyset]$. We remark that the conditions on indices used in the typing rules given in Section 4.1 are now conditions on sets of variables. The typing rules are given in Figure 3.

$$\frac{}{\Gamma, x : A \vdash x : A}\ Axiom \qquad \frac{\Gamma \vdash M : A \quad \Gamma \cap \Delta = \emptyset}{\Gamma, \Delta \vdash \Delta M : A}\ Weak$$

$$\frac{\Gamma \vdash M : B \to A \quad \Gamma \vdash N : B}{\Gamma \vdash (MN) : A}\ App \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : B \to A}\ Lambda$$

$$\frac{\Delta, \Pi \vdash N : A \quad \Gamma, x : A, \Pi \vdash M : B \quad (\Gamma, x : A) \cap \Delta = \emptyset}{\Delta, \Gamma, \Pi \vdash M[x, N, \Gamma, \Delta] : B}\ Subst$$

**Fig. 3.** Typing rules for the $\lambda_l$-calculus with named variables

We remark that whenever $\Gamma \vdash M[x, N, \Delta, \Pi]$ is derivable, then $\Gamma$ necessarily contains $\Delta$ and $\Pi$.

As expected the $\lambda_l$-calculus with names enjoys the subject reduction property (See [11] for a detailed proof).

**Theorem 6 (Subject Reduction).** *If $\Psi \vdash M : C$ and $M \longrightarrow M'$, then $\Psi \vdash M' : C$.*

We define the reduction rules only on typed terms, since we are focusing here on a named version of the *typed* $\lambda_l$ calculus with indices. These rules already give the flavor of what a general notion of reduction for non-typed terms with names should be, but a precise formalization of the untyped case is left for further work.

The reduction rules of the typed $\lambda_l$-calculus with names are given in Figure 4 (notice that rule $b_1$ is a particular case of rule $b_2$ with $\Delta = \emptyset$).

$$
\begin{array}{lrcll}
(b_1) & (\lambda x : A.M)N & \longrightarrow & M[x, N, \emptyset, \emptyset] & \\
(b_2) & (\Delta(\lambda x : A.M))N & \longrightarrow & M[x, N, \emptyset, \Delta] & \\
(f) & (\lambda y : A.M)[x, N, \Gamma, \Delta] & \longrightarrow & \lambda y : A.M[x, N, \Gamma + y, \Delta] & \text{if } y \notin FV(N) \\
(a) & (MP)[x, N, \Gamma, \Delta] & \longrightarrow & (M[x, P, \Gamma, \Delta]P[x, N, \Gamma, \Delta]) & \\
(e_1) & \Lambda M[x, N, \Gamma, \Delta] & \longrightarrow & (\Delta \cup (\Lambda \setminus x))M & x \in \Lambda \\
(e_2) & \Lambda M[x, N, \Gamma, \Delta] & \longrightarrow & (\Gamma \cap \Lambda)M[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] & x \notin \Lambda \\
(n_1) & y[x, N, \Gamma, \Delta] & \longrightarrow & y & y \neq x \\
(n_2) & x[x, N, \Gamma, \Delta] & \longrightarrow & \Gamma N & \\
(c_1) & M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta] & \longrightarrow & M[y, P[x, N, \Gamma \setminus \Lambda, \Delta], \Lambda, \Delta \cup (\Phi \setminus x)] & x \in \Phi \setminus \Lambda \\
(c_2) & M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta] & \longrightarrow & M[x, N, (\Gamma \setminus \Phi) + y, \Delta] & \\
 & & & \quad [y, P[x, N, \Gamma \setminus \Lambda, \Delta], \Lambda, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
(d) & \Gamma \Delta M & \longrightarrow & (\Gamma \cup \Delta)M &
\end{array}
$$

**Fig. 4.** Reduction Rules of the $\lambda_l$-calculus with named variables

As customary in explicit substitutions calculi with names [3], we work modulo $\alpha$-conversion, so that we can suppose that in the rule $Weak$ the set $\Delta$ does not contain variables that are bound in $M$. Also, this allows us to restrict rule $f$, without loss of generality, to the case where no variable capture arise.

In order to translate a term of $\lambda_l$ into a proof net, we use exactly the same translation of types that we used in Section 4.2 and we then define the translation of a term $M$ using the type derivation of $M$.

Since in proof nets there is no trace left of the order which is implicit in the formalism of de Bruijn indices, it comes as no surprise that the translation of $\lambda_l$ with names into the nets is really the same than the one for $\lambda_l$ (see [11] for full details).

The simulation of the reduction rules of the $\lambda_l$-calculus with names by the reduction $R_E$ is identical to that given in Section 4.2 for the $\lambda_l$-calculus with indices. We just remark that rule $n_3$ has no sense in the formalism with names so that the proof has one less case. We just state the result without repeating a boring verification:

**Lemma 7 (Simulation of $\lambda_l$ with names).** *If $t$ $\lambda_l$-reduces to $t'$ in the formalism with names, then $T(t) \longrightarrow^+_{R_E} T(t')$, except for the rules $e_2$ and $d$ for which we have $T(t) = T(t')$.*

We can then conclude the following:

**Theorem 7 (Strong Normalization of $\lambda_l$ with names).** *The typed $\lambda_l$-calculus with names is strongly normalizing.*

## 6 Conclusion and future works

In this paper we enriched the standard notion of cut elimination in proof nets in order to obtain a system $R_E$ which is flexible enough to provide an interpretation of $\lambda$-calculi with explicit substitutions and which is much simpler than the one proposed in [10]. We have proved that this system is strongly normalizing.

We have then proposed a natural translation from $\lambda_l$ into proof nets that immediately provides strong normalization of the typed version of $\lambda_l$, a calculus featuring full composition of substitutions. The proof is extremely simple w.r.t the proof of PSN of $\lambda_l$ given in [8] and shows in some sense that $\lambda_l$, which was designed independently of proof nets, is really tightly related to reduction in proof nets.

Finally, the fact that the relative order of variables is lost in the proof-net representation of a term lead us to discover a version of typed $\lambda_l$ with named variables, instead of de Bruijn indices. This typed named version of $\lambda_l$ gives a better understanding of the mechanisms of the calculus. In particular, names allow to understand the manipulation of explicit weakenings in $\lambda_l$ without entering into the details of renaming of de Bruijn indices. However, the definition of a general notion of reduction for non-typed terms with names remains as further work.

This work suggests several interesting directions for future investigation: on the linear logic side, one should wonder whether $R_E$ is the definitive system able to interpret $\beta$ reduction, or whether we need some more equivalences to be added. Indeed, there are still a few cases in which the details of a sequent calculus derivation are inessential, even if we did not need to consider them for the purpose of our work, like for example

$$\dfrac{\dfrac{\vdash \Gamma, B}{\vdash ?A, \Gamma, B}\ Weakening}{\vdash ?A, \Gamma, !B}\ Box \qquad \dfrac{\dfrac{\vdash \Gamma, B}{\vdash \Gamma, !B}\ Box}{\vdash ?A, \Gamma, !B}\ Weakening$$

On the explicit substitutions side, we look forward to the discovery of a calculus with multiple substitutions with the same properties as $\lambda_l$, in the spirit of $\lambda_\sigma$.


## Acknowledgments

We would like to thank Bruno Guillaume and Pierre-Louis Curien for their interesting remarks.

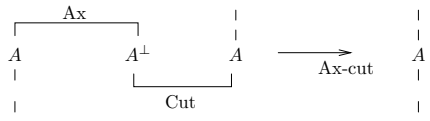

## References


1. M. Abadi, L. Cardelli, P. L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.
2. S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *LICS*, pages 211–222, Santa Cruz, California, 22–25 June 1992.
3. R. Bloo. *Preservation of Termination for Explicit Substitution.* PhD thesis, Eindhoven University of Technology, 1997.
4. R. Bloo and K. Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. In *Computing Science in the Netherlands*, pages 62–72. Netherlands Computer Science Research Foundation, 1995.
5. V. Danos. *La logique linéaire appliquée à l'étude de divers processus de normalisation (et principalement du $\lambda$-calcul).* PhD thesis, Université de Paris VII, 1990. Thèse de doctorat de mathématiques.
6. V. Danos, J.-B. Joinet, and H. Schellinx. Sequent calculi for second order logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic.* Cambridge University Press, 1995.

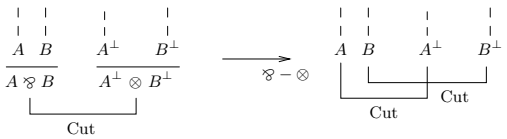
7. V. Danos and L. Regnier. Proof-nets and the Hilbert space. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 307–328. Cambridge University Press, London Mathematical Society Lecture Notes, 1995.
8. R. David and B. Guillaume. The $\lambda_l$-calculus. In *Proceedigs of the Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, pages 2–13, Trento, Italy, 1999.
9. R. Di Cosmo and S. Guerrini. Strong normalization of proof nets modulo structural congruences. In P. Narendran and M. Rusinowitch, editors, *10th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1631 of *Lecture Notes in Computer Science*, pages 75–89, Trento, Italy, 1999. Springer Verlag.
10. R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 35–46, Warsaw, Poland, 1997.
11. R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions, 1999. Available as `ftp://ftp.lri.fr/LRI/articles/kesner/es-pn.ps.gz`.
12. M. C. Ferreira, D. Kesner, and L. Puel. Lambda-calculi with explicit substitutions preserving strong normalization. *Applicable Algebra in Engineering Communication and Computing*, 9(4):333–371, 1999.
13. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.
14. J.-Y. Girard. Geometry of interaction I: interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic colloquium 1988*, pages 221–260. North Holland, 1989.
15. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *19thAnn. ACM Symp. on Principles of Programming Languages (POPL)*, pages 15–26, Albuquerque, New Mexico, 1992. ACM Press.
16. B. Guillaume. *Un calcul de substitution avec tiquettes*. PhD thesis, Universit de Savoie, 1999.
17. J. Lamping. An algorithm for optimal lambda calculus reduction. In *19thAnn. ACM Symp. on Principles of Programming Languages (POPL)*, pages 16–30, San Francisco, California, 1990. ACM Press.
18. P.-A. Mellies. Typed $\lambda$-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*, April 1995.
19. K. Rose. Explicit cyclic substitutions. In Rusinowitch and Rémy, editors, *Proc. of the Third International Workshop on Conditional Term Rewriting Systems (CTRS)*, number 656 in LNCS, pages 36–50, 1992.

# A Reduction of proof nets

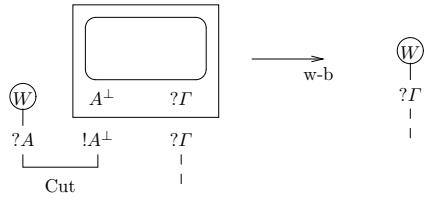
Reduction acting on a cut $Ax - cut$, removing an axiom :



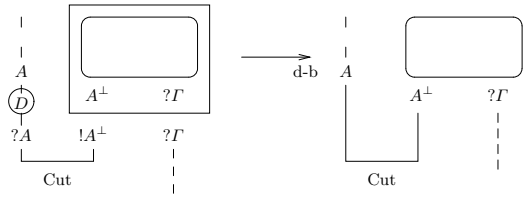

Reduction acting on a cut $\parr - \otimes$ :



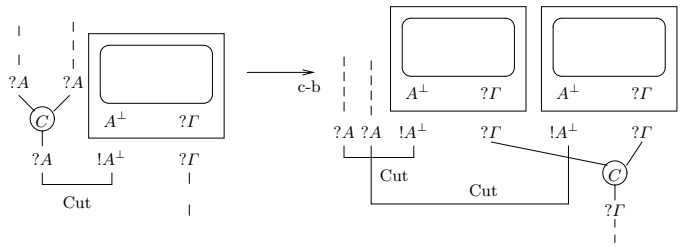
Reduction acting on a cut $w - b$, erasing a box :




Reduction acting on a cut $d - b$, opening a box :




Reduction acting on a cut $c - b$, duplicating a box :




Reduction acting on a cut $b - b$, absorbing a box into another :