

# Proof Nets and Explicit Substitutions

Roberto Di Cosmo<sup>1</sup>

Delia Kesner<sup>2</sup>

Emmanuel Polonovski<sup>1</sup>

<sup>1</sup> PPS (CNRS UMR 7126) - Université Paris 7 - 175, rue du Chevaleret - Paris, France.

Email:{dicosmo,polonovs}@pps.jussieu.fr

<sup>2</sup> LRI (CNRS UMR 8623) - Bât 490, Université de Paris-Sud - 91405 Orsay Cedex, France.

Email:kesner@lri.fr

Received 26 March 2002

We refine the simulation technique introduced in (Di Cosmo and Kesner 1997) to show strong normalization of  $\lambda$ -calculi with explicit substitutions via termination of cut elimination in proof nets (Girard 1987). We first propose a notion of equivalence relation for proof nets that extends the one in (Di Cosmo and Guerrini 1999), and we show that cut elimination modulo this equivalence relation is terminating. We then show strong normalization of the typed version of the  $\lambda_{ws}$ -calculus with de Bruijn indices (a calculus with full composition defined in (David and Guillaume 1999)) using a translation from typed  $\lambda_{ws}$  to proof nets. Finally, we propose a version of typed  $\lambda_{ws}$  with named variables which helps to better understand the complex mechanism of the explicit weakening notation introduced in the  $\lambda_{ws}$ -calculus with de Bruijn indices (David and Guillaume 1999).

## 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 - 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 provides a more refined computational model than the one given by the  $\lambda$ -calculus, which is the computational side of intuitionistic logic<sup>†</sup>.

The pioneer calculus with explicit substitutions,  $\lambda_{\sigma}$ , was introduced in (Abadi *et al* 1991, ) as a bridge between the classical  $\lambda$ -calculus and concrete implementations of functional programming languages. An important property of calculi with explicit substitutions is

<sup>†</sup> 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 (Girard 1989; Abramsky and Jagadeesan 1992; Danos 1990), leading to the works on optimal reduction (Gonthier, Abadi Lévy 1992; Lamping 1990).

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. 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 (Mellies 1995).

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 (Danos and Regnier 1995).

Finally, in (Di Cosmo and Kesner 1997), 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  $\lambda x$  (Rose 1992; Bloo and Rose 1995), 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 (Di Cosmo and Kesner 1997) 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  $\lambda_{ws}$ , a complex calculus of explicit substitution with de Bruijn indices and full composition (David and Guillaume 1999; David and Guillaume 2001). This translation allows us to prove that typed  $\lambda_{ws}$  is strongly normalizing, which is a new result confirming a conjecture in (David and Guillaume 1999; David and Guillaume 2001). Also, the fact that in the translation all information about variable order is lost suggests a version of typed  $\lambda_{ws}$  with named variables which 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, we believe that the typed named version of  $\lambda_{ws}$  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  $\lambda_{ws}$  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 of 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  $\lambda_{ws}$  calculus with its type system, present the translation into proof nets, and show strong normalization of typed  $\lambda_{ws}$ . Finally, we introduce a version of typed  $\lambda_{ws}$  with named variables (Section 5), enjoying the same good properties, and we conclude with some remarks and directions for future work (Section 7).

## 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* equipped with an involutive <sup>‡</sup> function  $\perp : \mathcal{A} \rightarrow \mathcal{A}$ , called *linear negation*.

The set of formulae of the multiplicative exponential fragment of linear logic (called MEL-L) is defined by the following grammar, where  $a \in \mathcal{A}$ :

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

We extend the notion of *linear negation* to formulae as follows:

$$\begin{aligned} (?A)^\perp &= !(A^\perp) & (A \otimes B)^\perp &= A^\perp \wp B^\perp \\ (!A)^\perp &= ?(A^\perp) & (A \wp B)^\perp &= A^\perp \otimes B^\perp \end{aligned}$$

The name MELL comes from the connectors  $\otimes$  and  $\wp$  which are called “multiplicatives”, while  $!$  and  $?$  are called “exponentials”. While we refer the interested reader to (Girard 1987) for more details on linear logic, we give here a one-sided presentation of the sequent calculus for MELL:

$$\begin{array}{c} \frac{}{\vdash A, A^\perp} \textit{Axiom} \quad \frac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta} \textit{Cut} \quad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \textit{Dereliction} \quad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \textit{Contraction} \\ \\ \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B} \textit{Par} \quad \frac{\vdash \Gamma, A \quad \vdash B, \Gamma'}{\vdash \Gamma, A \otimes B, \Gamma'} \textit{Times} \quad \frac{\vdash \Gamma}{\vdash \Gamma, ?A} \textit{Weakening} \quad \frac{\vdash A, ?\Gamma}{\vdash !A, ?\Gamma} \textit{Box} \end{array}$$

*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  $\vdash A_1 \wp A_2, \dots, A_{n-1} \wp A_n$  from  $\vdash A_1, \dots, A_n$ , 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; they are given in Figure 2. 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  $\Gamma, \Delta, \dots$  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  $A \quad A^\perp$ .

*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

<sup>‡</sup> A function  $f$  is involutive iff  $f(f(p)) = p$

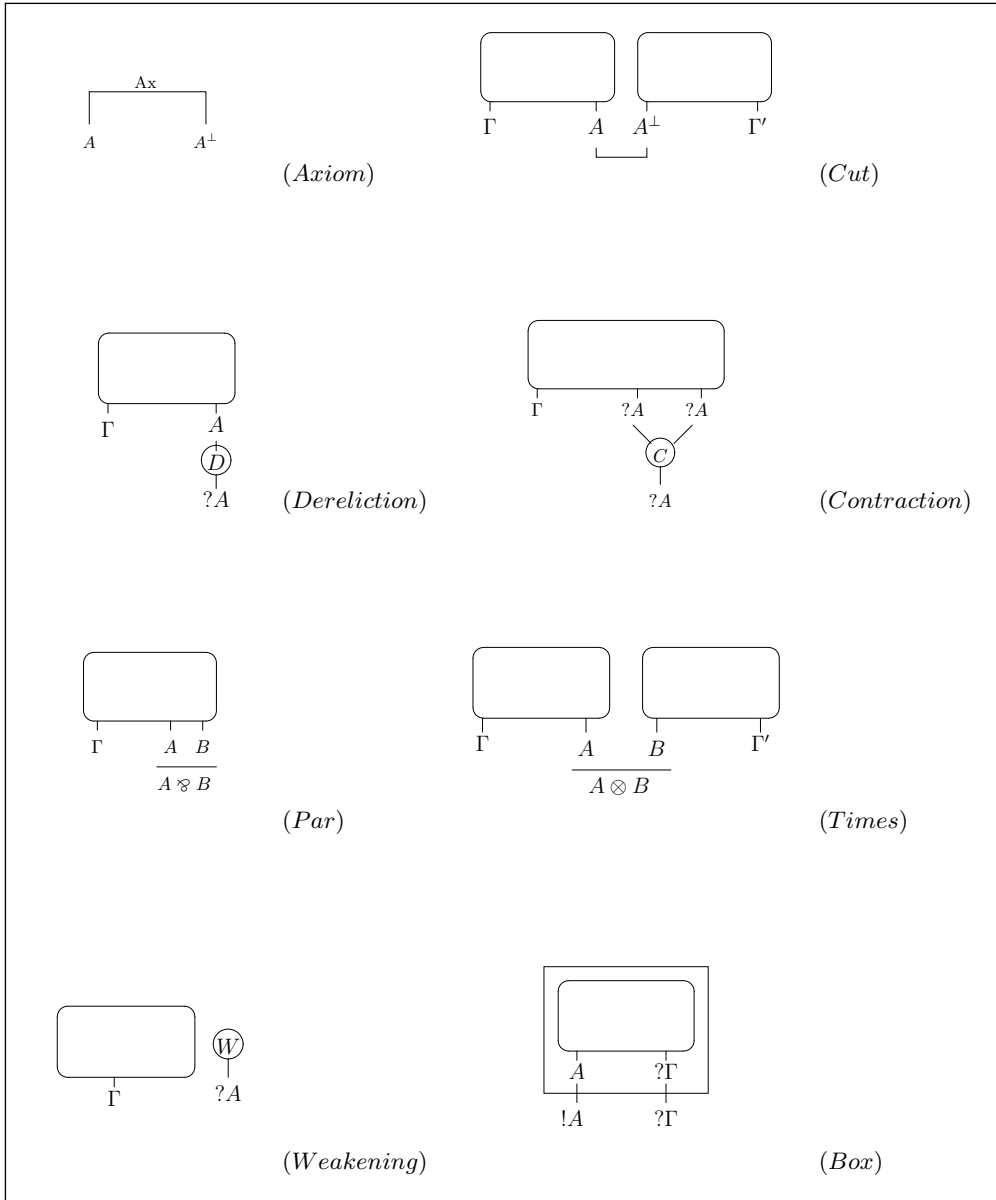
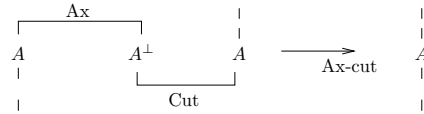


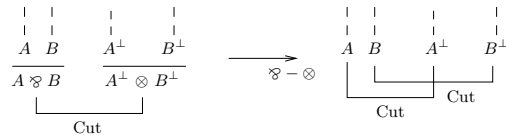
Fig. 1. MELL Proof Nets

procedure on sequent derivations. The traditional reduction system for MELL is defined as follows:

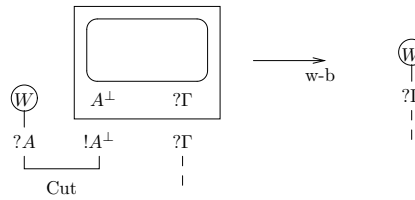
Reduction acting on a cut  $Ax - cut$ , removing an axiom :



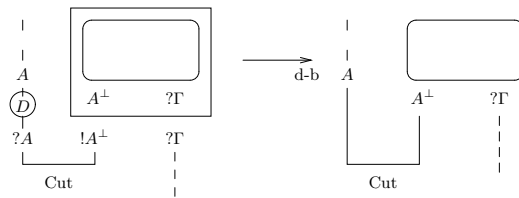
Reduction acting on a cut  $\wp - \otimes$  :



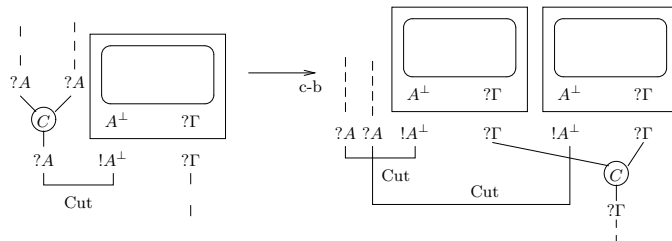
Reduction acting on a cut  $w - b$ , erasing a box :



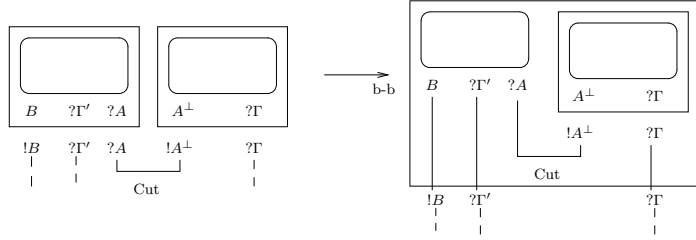
Reduction acting on a cut  $d - b$ , opening a box :



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

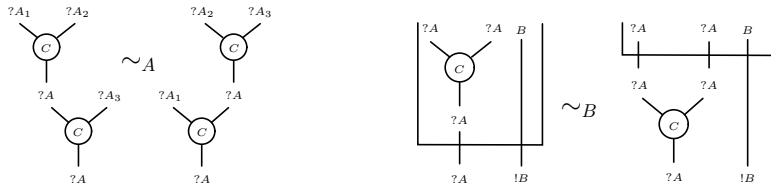


Reduction acting on a cut  $b - b$ , absorbing a box into another :



*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 (Di Cosmo and Guerrini 1999), where the following two equivalences are introduced:



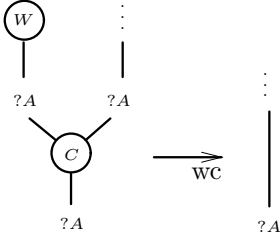
Equivalence  $A$  turns contraction into an associative operator, and corresponds to forgetting the order in which the contraction rule is used to build, for example, the derivation:

$$\frac{\frac{\frac{\vdash ?A, ?A, ?A}{\vdash ?A, ?A} \text{ Contraction}}{\vdash ?A} \text{ Contraction}}$$

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.

$$\frac{\frac{\frac{\vdash ?A, ?A, B}{\vdash ?A, B} \text{ Contraction}}{\vdash ?A, !B} \text{ Box}}{\vdash ?A, !B} \text{ Contraction} \quad \frac{\frac{\frac{\vdash ?A, ?A, B}{\vdash ?A, ?A, !B} \text{ Box}}{\vdash ?A, !B} \text{ Contraction}}$$

Finally, besides the equivalence relation defined in (Di Cosmo and Guerrini 1999), we will also need an extra reduction rule allowing to remove unneeded weakening links when simulating explicit substitutions:



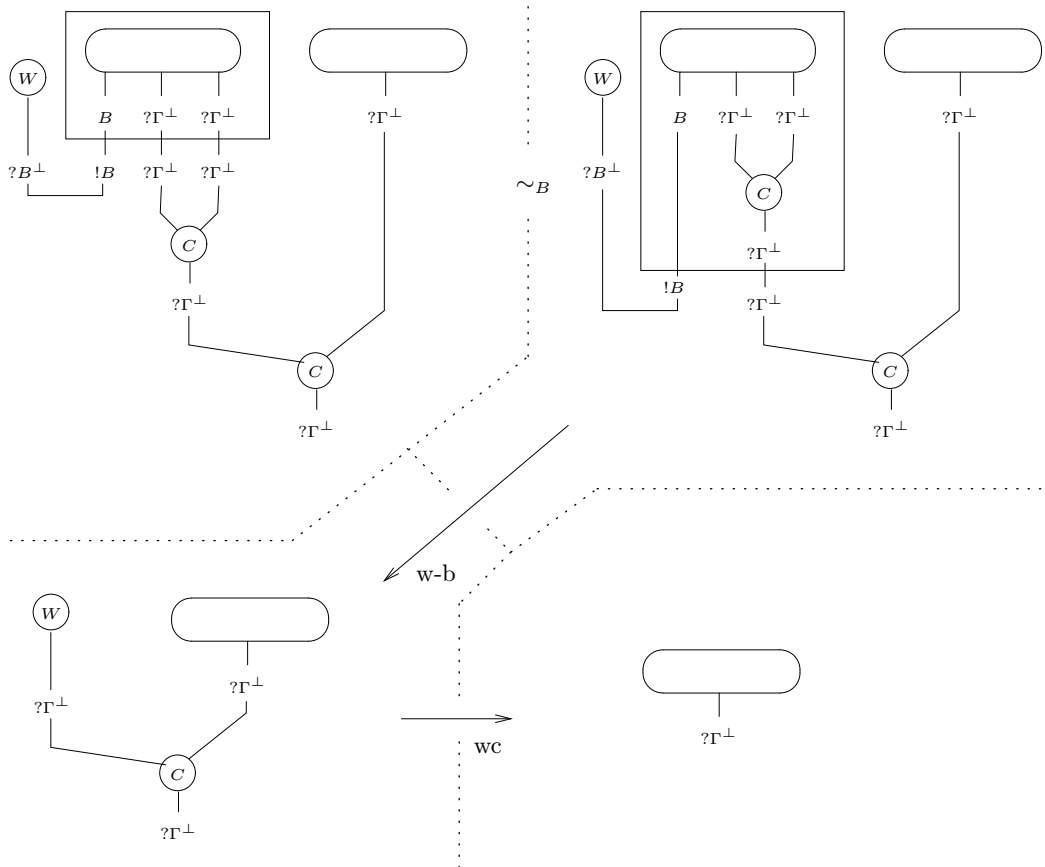
This rule allows to simplify the proof below on the left into the proof on the right

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

**Notation:** We will call in the following  $R$  the system made of rules  $Ax - cut$ ,  $\wp - \otimes$ ,  $w - b$ ,  $d - b, c - b, b - b$  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,  $R_E^{-wc}$  will stand for system  $R_E$  without rule  $wc$ .

Systems  $R_E$  and  $R_E^{-wc}$ , that contain  $E$ , are actually defining a notion of *reduction modulo an equivalence relation*, so we write for example  $r \rightarrow_{R_E} s$  if and only if there exist  $r'$  and  $s'$  such that  $r =_E r' \rightarrow_R s' =_E s$ , where the equality  $=_E$  is the reflexive, symmetric and transitive closure of the relation defined by  $A$  and  $B$ .

An example of reduction in  $R_E$  is given here:



The reduction  $R_E$  is flexible enough to allow an elegant simulation of  $\beta$  reduction and of explicit substitutions, but for that, we first need to establish that  $R_E$  is strongly normalizing. Let us see this property in the next section.

### 3. Termination of $R_E$

We know from (Di Cosmo and Guerrini 1999) that  $R_E^{-wc}$  is terminating, and we can show easily that  $wc$  is terminating too. In this section we show that the  $wc$ -rule can be postponed with respect to all the other rules of  $R_E^{-wc}$ , so that termination of  $R_E$  will follow from a well-known abstract lemma.

Let us first remind the following result from (Di Cosmo and Guerrini 1999):

**Lemma 3.1 (Termination of  $R_E^{-wc}$ )** The relation  $\longrightarrow_{R_E^{-wc}}$  is terminating on  $PN$ .

Then, we establish the termination of  $wc$ .

**Lemma 3.2 (Termination of  $wc$ )** The 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.  $\square$

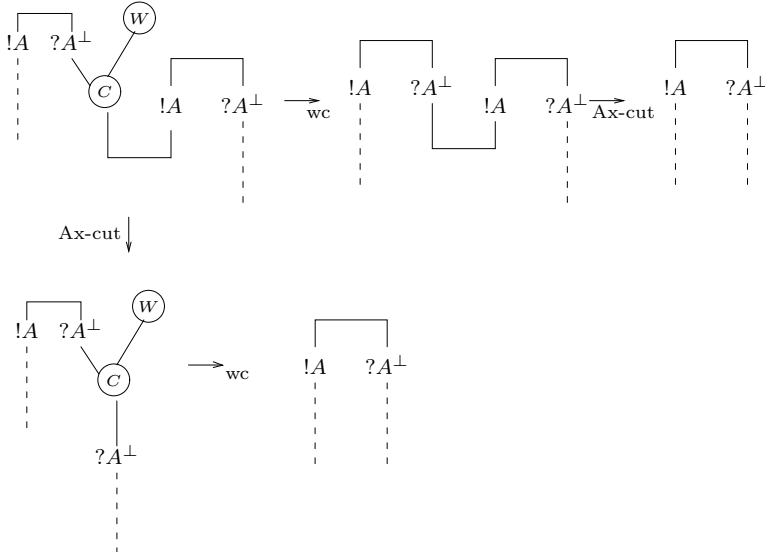
Finally, we show that given any proof net, the  $wc$ -rule can be postponed with respect to any rule of  $R_E^{-wc}$ .

**Lemma 3.3 (Postponement of  $wc$  w.r.t  $R_E^{-wc}$ )** Let  $t$  be a proof net. If  $t \longrightarrow_{wc} \longrightarrow_{R_E^{-wc}} t'$ , then, there is a sequence  $t \longrightarrow_{R_E^{-wc}}^+ \longrightarrow_{wc}^* t'$ .

*Proof.* Let  $t \longrightarrow_{wc} \longrightarrow_{R_E^{-wc}} t'$  be a reduction sequence starting at  $t$  with a  $wc$ -reduction step. Let us show that we can build an equivalent reduction  $t \longrightarrow_{R_E^{-wc}}^+ \longrightarrow_{wc}^* t'$  by analyzing all the possible cases.

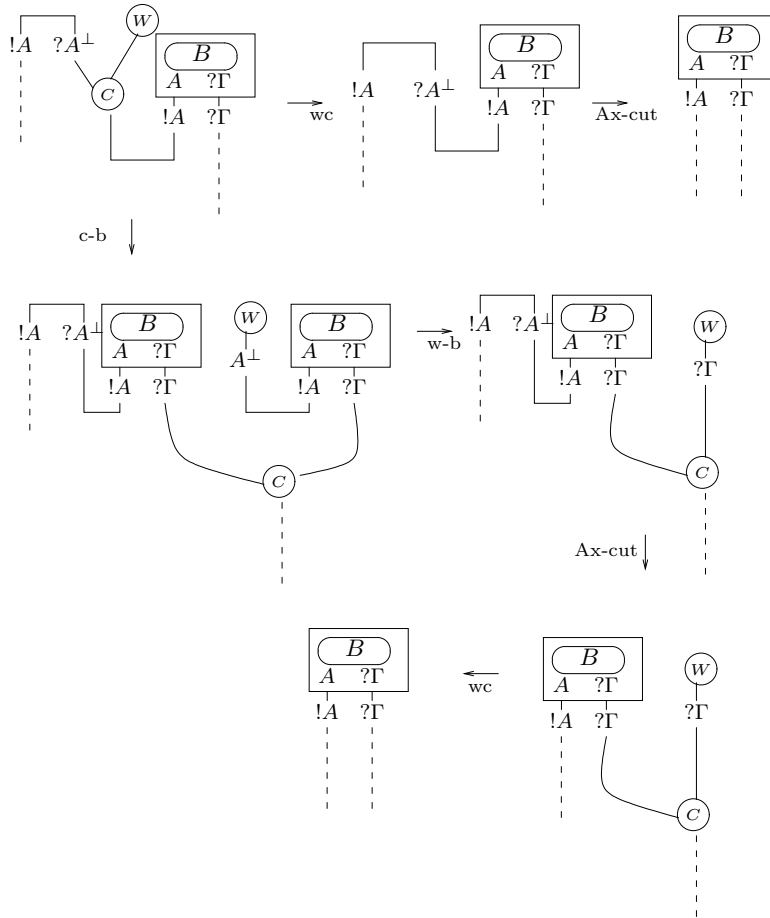
We do not detail here the cases of disjoint redexes: if we apply the  $wc$ -rule followed by a rule  $R1$  in  $R_E^{-wc}$  and if the redexes occur at disjoint positions, then it is evident that  $R1$  can be applied first, followed by  $wc$ , and getting the same result. We study now all the remaining cases:

1 The rule  $Ax - cut$ , first possibility :

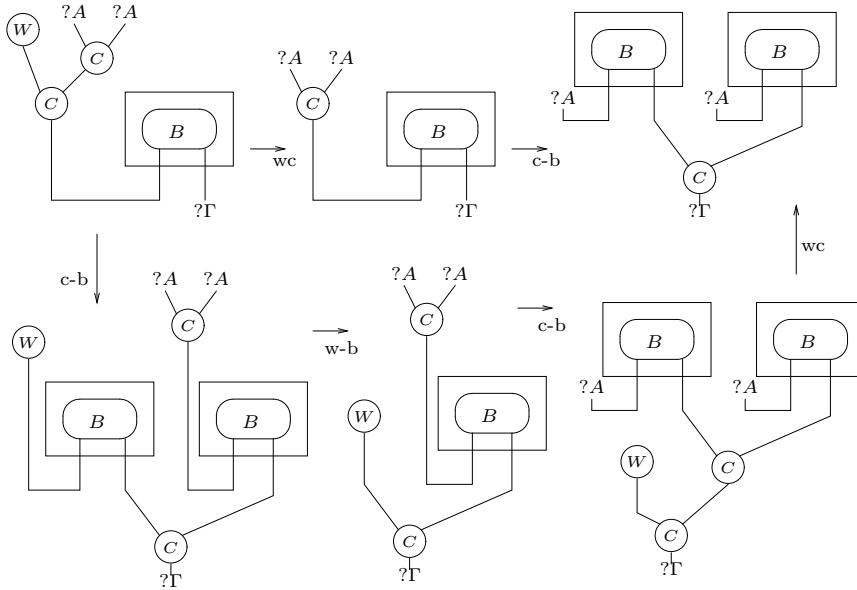




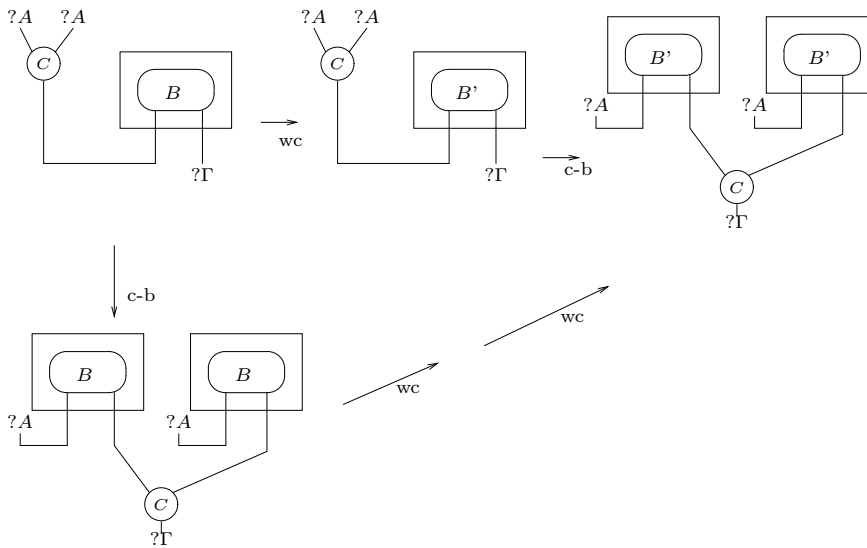
2 The rule  $Ax - cut$ , second possibility :



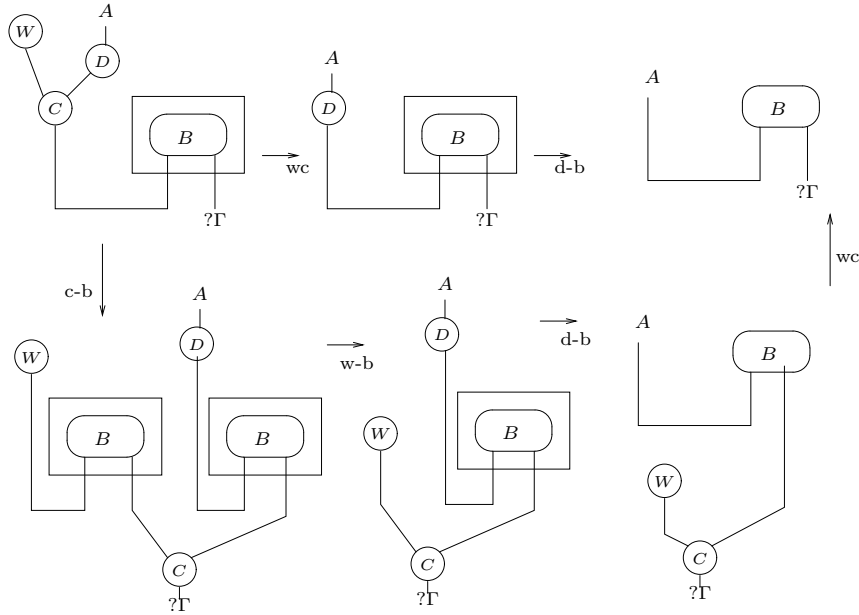
3 The rule  $c - b$ , first possibility :



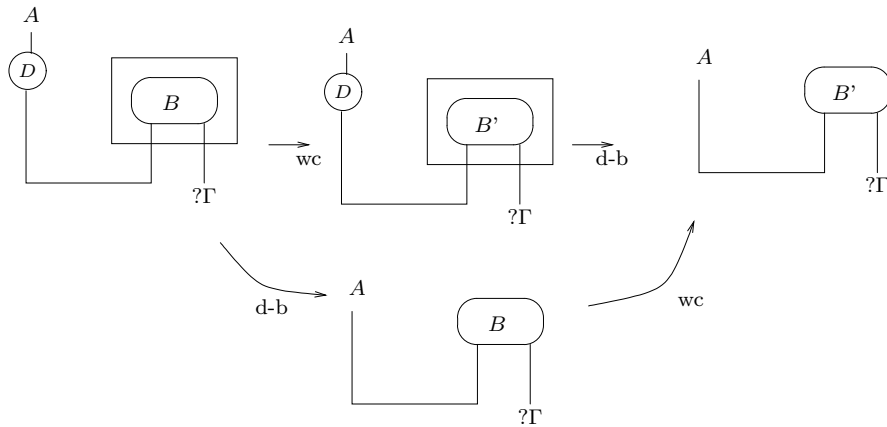
4 The rule  $c - b$ , second possibility :



5 The rule  $d - b$ , first possibility :

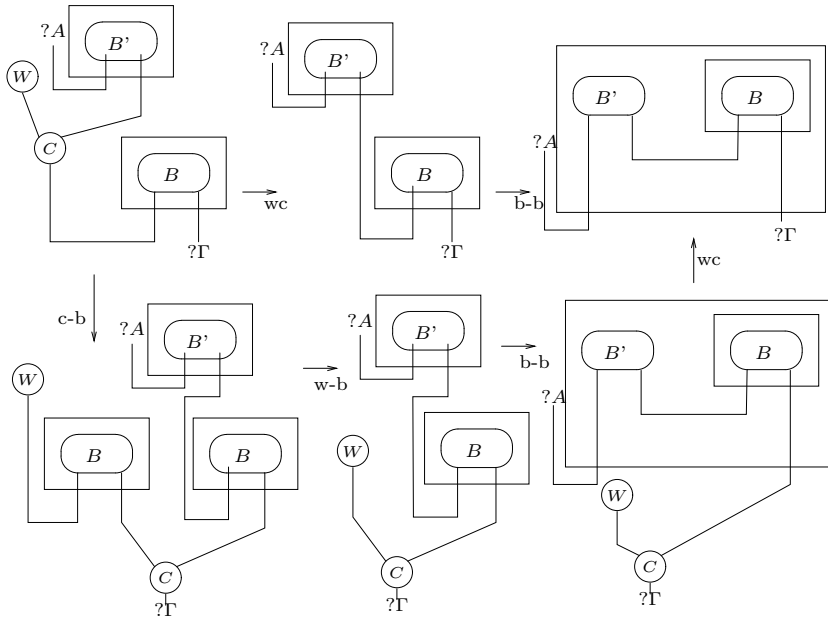


6 The rule  $d - b$ , second possibility :



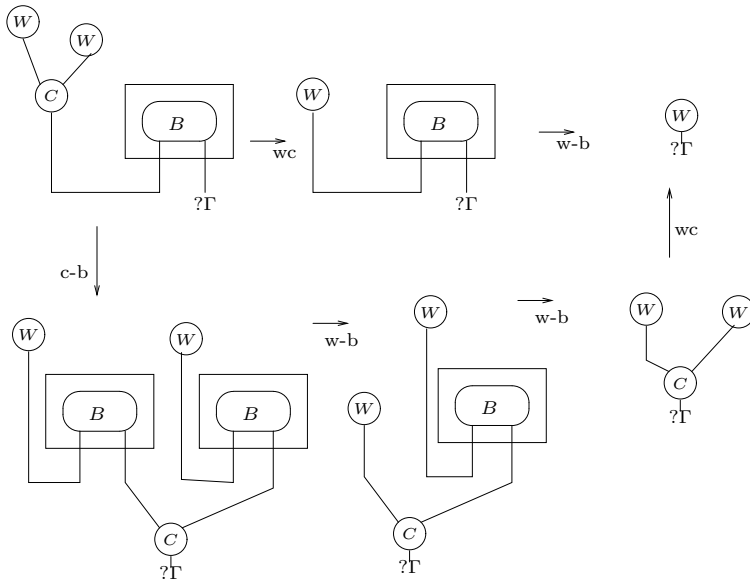
We notice that everything is happening as if the redexes were disjoint. This is due to the fact that the  $d - b$  rule is non-duplicating and non-erasing w.r.t boxes. As a consequence, the  $wc$ -redex is still preserved after the application of the  $d - b$  rule.

7 The rule  $b - b$ , first possibility :

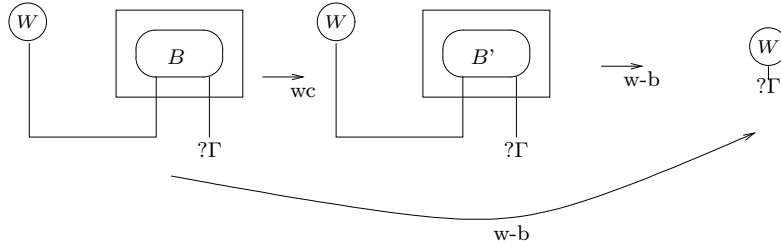


8 The rule  $b - b$ , second possibility : For the same reason as for  $d - b$ , the redexes are considered as disjoint.

9 The rule  $w - b$ , first possibility :



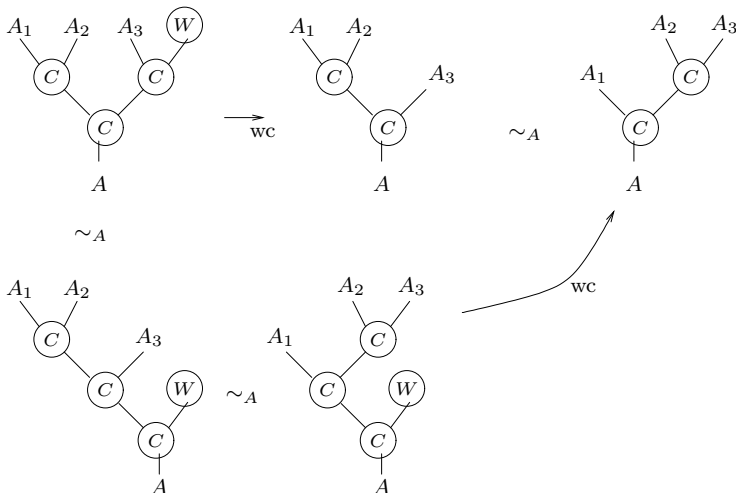
10 The rule  $w - b$ , second possibility :



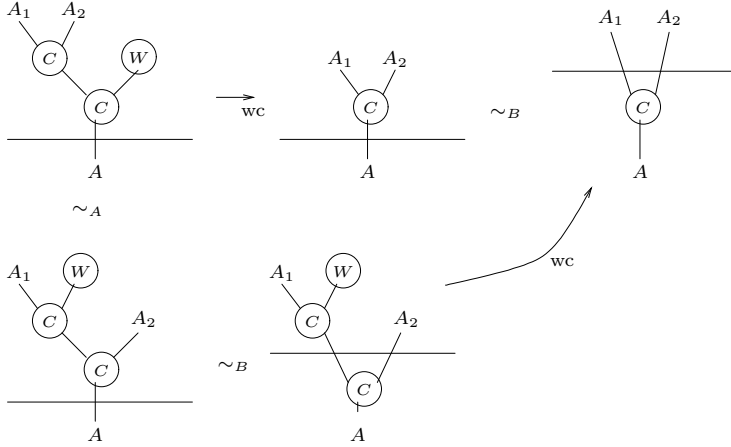
11 The rules  $\wp \otimes$  cut : we just notice that in this case the redexes are disjoint.

Until now we have only worked with reduction rules of  $R_E$ , but to complete our statement we also need to show that the  $wc$ -rule can be delayed w.r.t one equivalence step. We proceed as we did for the reduction rules. We do not study the cases where redexes are disjoint because they are evident. The remaining cases are the following:

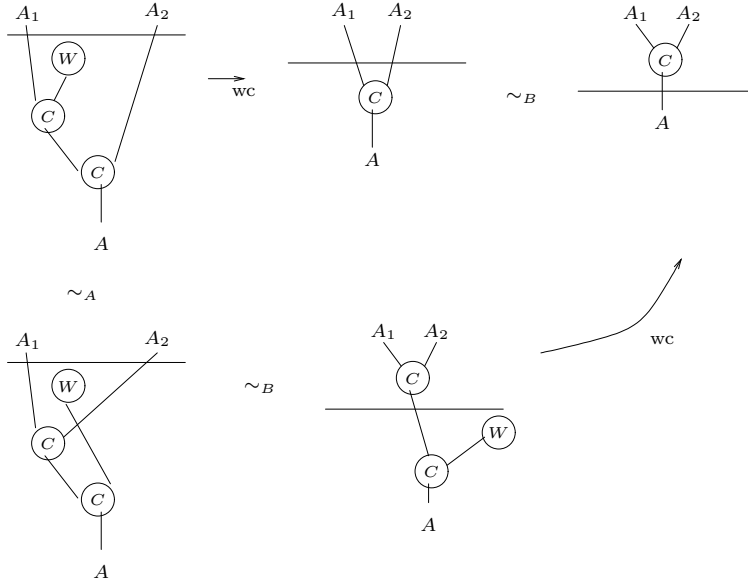
1 Associativity :



2 Box passing, first case :



3 Box passing second case :



□

**Lemma 3.4 (Extraction of  $R_E^{-wc}$ )** Let  $S$  be an infinite sequence of  $R_E$ -reductions starting at a proof net  $t$ . Then, there is a sequence of  $R_E$ -reductions from the same proof net  $t$  which starts by  $t \rightarrow_{R_E^{-wc}} t'$ , where  $t'$  is also a proof net, and which continues with an infinite sequence  $S'$ . We write this sequence as  $(t \rightarrow_{R_E^{-wc}} t') \cdot S'$ .

*Proof.* Let  $S$  be an infinite sequence of  $R_E$ -reductions starting at  $t$ :

$$t \rightarrow_{R_E} \dots \rightarrow_{R_E} \dots \rightarrow_{R_E} \dots$$

We know, by Lemmas 3.2 and 3.1, that the systems  $wc$  and  $R_E^{-wc}$  are both terminating, so it is not possible to have an infinite sequence only made of  $wc$  or  $R_E^{-wc}$ . As a consequence,

the infinite sequence of  $R_E$ -reductions must be an infinite alternation of non-empty finite sequences of  $wc$  and  $R_E^{-wc}$ .

Now, there are two cases: either the alternation of sequences starts with a  $R_E^{-wc}$ -reduction step, and then the result holds by taking the sequence  $S$  without its first reduction step as  $S'$ ; or the alternation starts with a  $wc$ -step:

$$t \longrightarrow^+_{wc} \longrightarrow^+_{R_E^{-wc}} \longrightarrow^+_{wc} \longrightarrow^+_{R_E^{-wc}} \dots$$

that is, written in other way

$$t \longrightarrow^+_{wc} \longrightarrow_{R_E^{-wc}} t'' \longrightarrow^*_{R_E^{-wc}} \longrightarrow^+_{wc} \longrightarrow^+_{R_E^{-wc}} \dots$$

In this case, we consider the sub-sequence  $P = t \longrightarrow^+_{wc} \longrightarrow_{R_E^{-wc}} t''$  of the sequence  $S$  starting at  $t$ . This sub-sequence is composed by  $k$  reduction steps of  $wc$  and one reduction of  $R_E^{-wc}$ . Let call  $R$  the remaining sub-sequence of  $S$ .

By applying Lemma 3.3  $k$  times on  $P$ , we can move the rule of  $R_E^{-wc}$  at the head of the sequence. We thus obtain a finite sequence  $P'$  which begins with a reduction  $t \longrightarrow_{R_E^{-wc}} t'$ , and ends on  $t''$ . As a consequence,  $P' \cdot R$  is the infinite sequence starting by a reduction  $R_E^{-wc}$  we were looking for.  $\square$

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

**Theorem 3.5 (Termination of  $R_E$  on proof nets)** The reduction relation  $R_E$  is terminating on the set of proof nets.

*Proof.* We show it by contradiction. Let us suppose that  $R_E$  is not terminating. Then, there exist a proof net  $t$  and an infinite sequence  $S$  of  $R_E$  starting at  $t$ . By applying Lemma 3.4 to this sequence  $S$ , we obtain a sequence  $(t \longrightarrow_{R_E^{-wc}} t') \cdot S'$  such that  $S'$  is infinite again. If we iterate this procedure an arbitrary number times, we obtain a sequence of  $R_E^{-wc}$ -reduction steps arbitrary long. This contradicts the fact that  $R_E^{-wc}$  is terminating.  $\square$

#### 4. From $\lambda_{ws}$ with de Bruijn indices to $PN$

We now study the translation from typed terms of the  $\lambda_{ws}$ -calculus (David and Guillaume 1999; David and Guillaume 2001) into proof nets. We start by introducing the calculus, then we give the translation of types of  $\lambda_{ws}$  into formulae of linear logic, and the translation of terms of  $\lambda_{ws}$  into linear logic proof nets  $PN$ . We verify that we can correctly simulate every reduction step of  $\lambda_{ws}$  via the notion of reduction  $R_E$ . Finally, we use this simulation result to show strong normalization of the  $\lambda_{ws}$ -calculus.

##### 4.1. The $\lambda_{ws}$ -calculus

The  $\lambda_{ws}$ -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  $\lambda_{ws}$  are given by the following grammar:

$M ::=$	$\underline{n}$	<i>variable</i>
	$\lambda M$	<i>abstraction</i>
	$(MM)$	<i>application</i>
	$\langle k \rangle M$	<i>label</i>
	$[i/M, j]M$	<i>substitution</i>

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 the  $j - 1$  following indices are not free in  $M$ . Those indices are used to split the typing environment of  $[i/N, j]M$  in three parts: the first (resp. second) one for free variables of  $M$  (resp.  $N$ ), the third one for the free variables in  $M$  and  $N$ .

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  $\lambda_{ws}$  are given in Figure 2 and the typing rules of  $\lambda_{ws}$  are given in Figure 3, where we suppose that  $|\Gamma| = i$  and  $|\Delta| = j$ .

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

Fig. 2. Reduction rules of  $\lambda_{ws}$  with de Bruijn indices

$$\begin{array}{ll}
\frac{}{\Gamma, A, \Delta \vdash \underline{i} : A} \text{Ax} & \frac{\Delta \vdash M : B}{\Gamma, \Delta \vdash \langle i \rangle M : B} \text{Weak} \\
\frac{\Gamma \vdash M : B \rightarrow A \quad \Gamma \vdash N : B}{\Gamma \vdash (MN) : A} \text{App} & \frac{B, \Gamma \vdash M : C}{\Gamma \vdash \lambda M : B \rightarrow C} \text{Lamb} \\
\frac{\Delta, \Pi \vdash N : A \quad \Gamma, A, \Pi \vdash M : B}{\Gamma, \Delta, \Pi \vdash [i/N, j]M : B} \text{Sub}
\end{array}$$

Fig. 3. Typing rules for  $\lambda_{ws}$  with de Bruijn indices

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

As expected the  $\lambda_{ws}$ -calculus enjoys the subject reduction property (Guillaume 1999).

**Theorem 4.1 (Subject Reduction)** If  $\Psi \vdash M : C$  and  $M \longrightarrow M'$ , then  $\Psi \vdash M' : C$ .

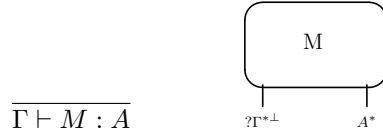
#### 4.2. Translation of types and terms of $\lambda_{ws}$

We use the translation of types introduced in (Danos, Joinet and Schellinx 1995) given by :



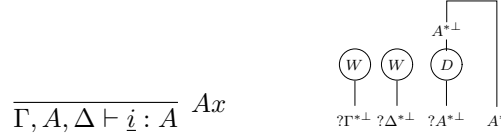
$$\begin{aligned} A^* &= A && \text{if } A \text{ is an atomic type} \\ (A \rightarrow B)^* &= ?((A^*)^\perp) \wp !B^* && \text{(that is, } !A^* \multimap !B^* \text{) otherwise} \end{aligned}$$

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  $\lambda_{ws}$ , whose typing judgement ends with the conclusion written below on the left, a proof net having the shape sketched below on the right:



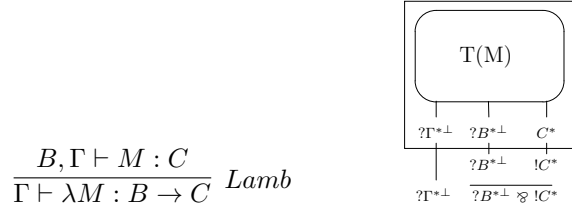
Here is the formal definition of the translation  $T$  from  $\lambda_{ws}$ -terms into proof nets.

- If the term is a variable and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

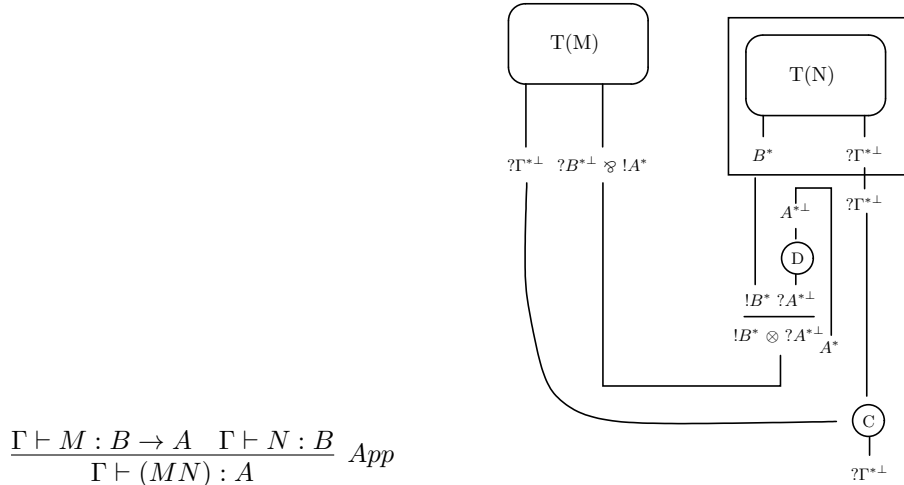


where  $i$  is the position of  $A$  in the typing environment,

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

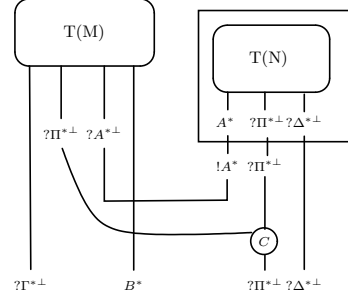


- If the term is an application and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right



- If the term is a substitution and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

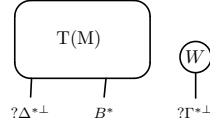
$$\frac{\Delta, \Pi \vdash N : A \quad \Gamma, A, \Pi \vdash M : B}{\Gamma, \Delta, \Pi \vdash [i/N, j]M : B} \textit{Sub}$$



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 typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



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

### 4.3. Simulating $\lambda_{ws}$ -reduction

We now verify that our notion of reduction  $R_E$  on  $PN$  simulates the  $\lambda_{ws}$ -reduction on typed  $\lambda_{ws}$ -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  $A \rightarrow B$  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.

**Notation:** In the proof of the following lemma, we will draw several complex proof nets, where the translations  $?\Gamma^{*\perp}, ?\Delta^{*\perp}, ?\Pi^{*\perp}$ , etc. of the environments  $\Gamma, \Delta, \Pi$ , etc. appear repeated many times. In order to make these pictures more readable, we will make a slight abuse of notation, only in the following proof, by simply writing  $\Gamma$  in place of its correct translation  $?\Gamma^{*\perp}$ .

**Lemma 4.2 (Simulation of  $\lambda_{ws}$ )** The relation  $R_E$  simulates the  $\lambda_{ws}$ -reduction on typed terms: if  $t \rightarrow_{\lambda_{ws}} t'$ , then  $T(t) \rightarrow_{R_E}^+ T(t')$ , excepted for the rules  $e_2$  and  $d$  for which we have  $T(t) = T(t')$ .

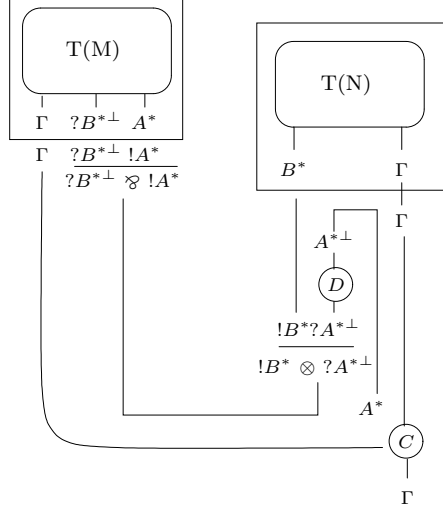
*Proof.* The proof proceeds by cases on the reduction rule applied in the step  $t \rightarrow_{\lambda_{ws}} t'$ . Since reductions  $\lambda_{ws}$  and  $R_E$  are closed under all contexts, we only need to study the cases where reduction takes place at the head position of  $t$ . In the proof, rule  $wc$  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  $f, a, c_1, c_2$ .

- **Rule  $b_1$  :**  $(\lambda MN) \rightarrow [0/N, 0]M$

The typing judgment of  $(\lambda MN)$  ends with

$$\frac{\frac{B, \Gamma \vdash M : A}{\Gamma \vdash \lambda M : B \rightarrow A} \textit{Lamb} \quad \Gamma \vdash N : B}{\Gamma \vdash ((\lambda M)N) : A} \textit{App}$$

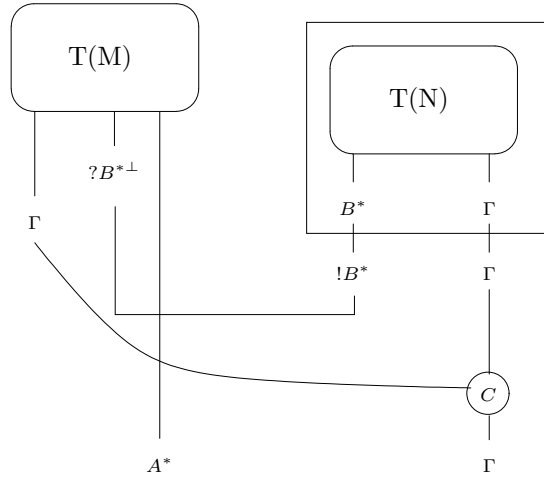
and its translation is the proof net



The typing judgment of  $[0/N, 0]M$  must end with:

$$\frac{B, \Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash [0/N, 0]M : A} \text{Sub}$$

and its translation is the proof net



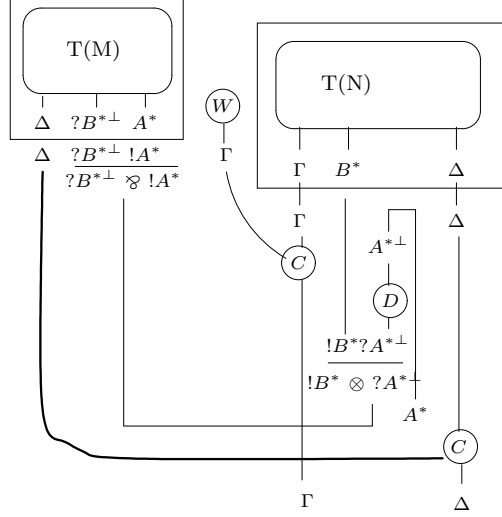
Starting from the first proof net, we eliminate the  $\wp$ - $\otimes$  cut, then the  $d$ - $b$  cut and finally the  $Ax$ -cut to obtain the final proof net.

— **Rule  $b_2$**  :  $((\langle k \rangle \lambda M)N) \longrightarrow [0/N, k]M$

The typing environment can be split in two parts  $\Gamma$  and  $\Delta$ , where  $k$  is the length of  $\Gamma$ . The typing judgment of  $((\langle k \rangle \lambda M)N)$  ends with

$$\frac{\frac{B, \Delta \vdash M : A}{\Delta \vdash \lambda M : B \rightarrow A}}{\Gamma, \Delta \vdash \langle k \rangle \lambda M : B \rightarrow A} \quad \Gamma, \Delta \vdash N : B}{\Gamma, \Delta \vdash ((\langle k \rangle \lambda M)N) : A}$$

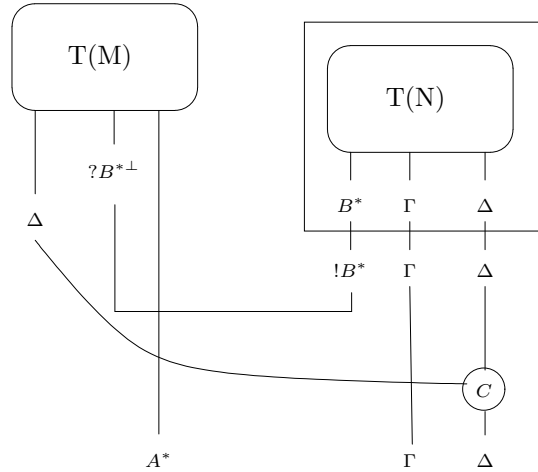
and its translation is the proof net



The typing judgment of  $[0/N, k]M$  must end with:

$$\frac{\Gamma, \Delta \vdash N : B \quad B, \Delta \vdash M : A}{\Gamma, \Delta \vdash [0/N, k]M : A} \textit{Sub}$$

and its translation is the proof net



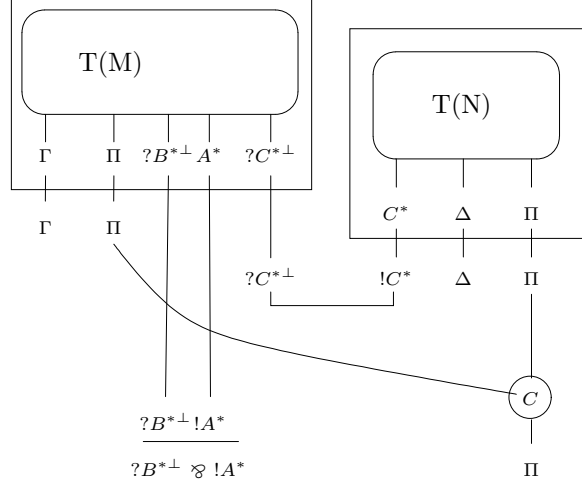
As for the  $b_1$  rule, we eliminate the  $\wp$ - $\otimes$  cut, then the  $d-b$  cut, and the  $Ax-cut$  cut. Finally, we apply the  $wc$  rule to achieve the desired result.

— **Rule  $f$**  :  $[i/N, j]\lambda M \longrightarrow \lambda[i+1/N, j]M$

The typing environment can be split in three parts  $\Gamma, \Delta, \Pi$ , where  $i$  is the length of  $\Gamma$  and  $j$  is the length of  $\Delta$ . The typing judgment of  $[i/N, j]\lambda M$  ends with

$$\frac{\Delta, \Pi \vdash N : C \quad \frac{B, \Gamma, C, \Pi \vdash M : A}{\Gamma, C, \Pi \vdash \lambda M : B \rightarrow A} \textit{Lamb}}{\Gamma, \Delta, \Pi \vdash [i/N, j]\lambda M : B \rightarrow A} \textit{Sub}$$

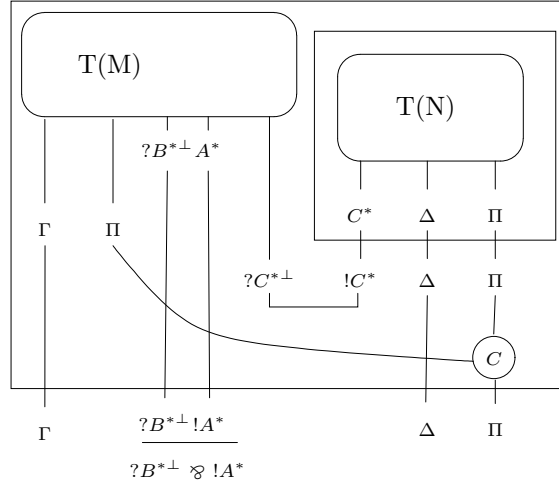
and its translation is the proof net



The typing judgment of  $\lambda[i + 1/N, j]M$  must end with:

$$\frac{\frac{\Delta, \Pi \vdash N : C \quad B, \Gamma, C, \Pi \vdash M : A}{B, \Gamma, \Delta, \Pi \vdash [i + 1/N, j]M : A} \text{Sub}}{\Gamma, \Delta, \Pi \vdash \lambda[i + 1/N, j]M : B \rightarrow A} \text{Lamb}$$

and its translation is the proof net



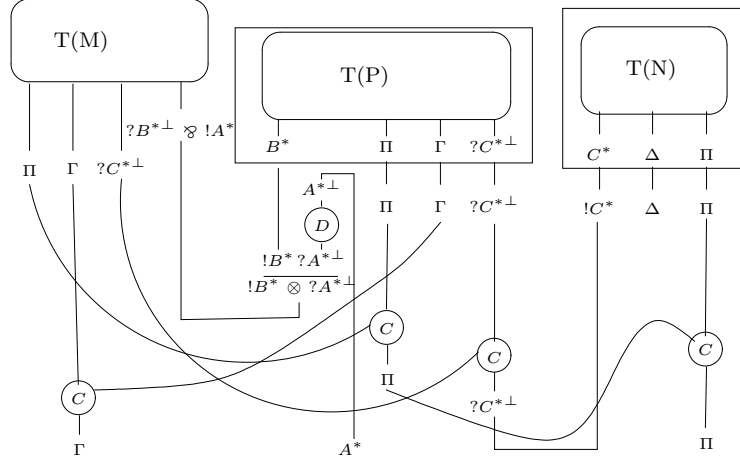
To reduce the first proof net into the second one, we must eliminate the  $b - b$  cut, then use the equivalence relation  $B$  (we will exactly show how to use the equivalence relations in the case of the rule  $a$ ).

— **Rule  $a$**  :  $[i/N, j](MP) \longrightarrow (([i/N, j]M)([i/N, j]P))$

The typing environment can be split in three parts  $\Gamma, \Delta, \Pi$ , where  $i$  is the length of  $\Gamma$  and  $j$  is the length of  $\Delta$ . The typing judgment of  $[i/N, j](MP)$  ends with

$$\frac{\frac{\Gamma, C, \Pi \vdash M : B \rightarrow A \quad \Gamma, C, \Pi \vdash P : B}{\Gamma, C, \Pi \vdash MP : A} \text{App}}{\Gamma, \Delta, \Pi \vdash [i/N, j](MP) : A} \text{Sub}$$

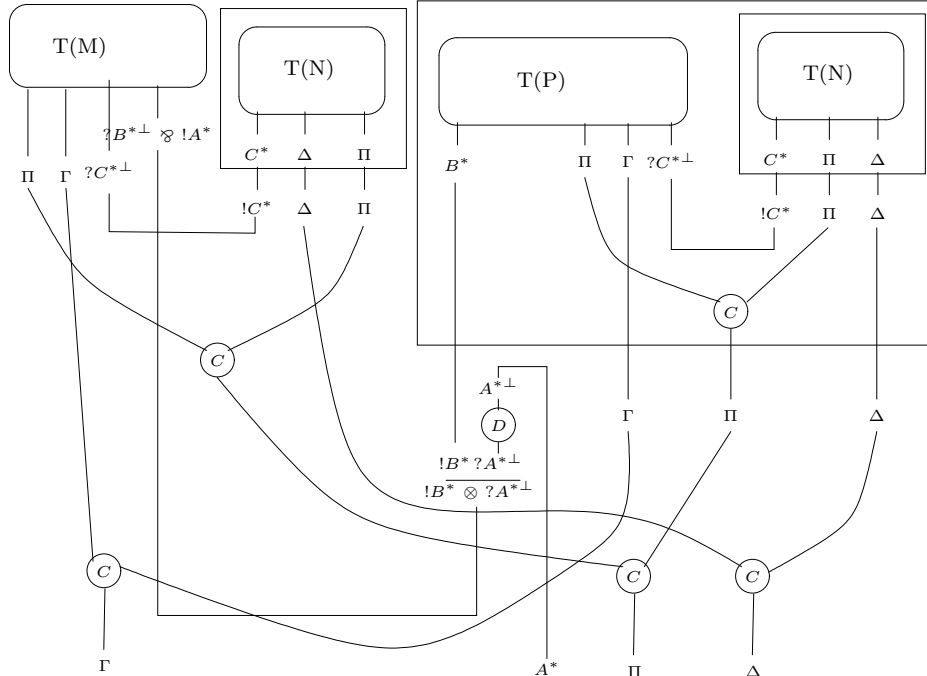
and its translation is the proof net



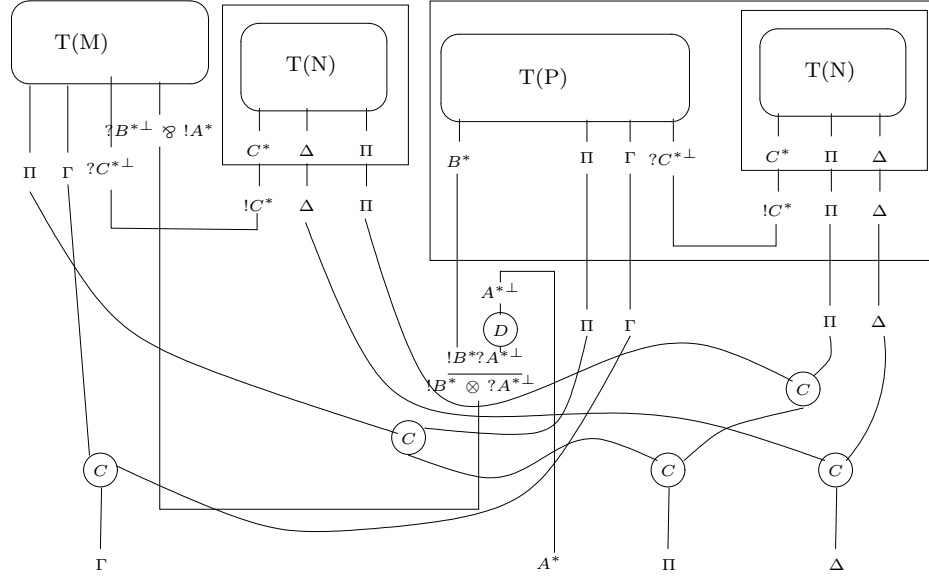
The typing judgment of  $(([i/N, j]M)([i/N, j]P))$  must end with:

$$\frac{\frac{\Delta, \Pi \vdash N : C \quad \Gamma, C, \Pi \vdash M : B \rightarrow A}{\Gamma, \Delta, \Pi \vdash ([i/N, j]M) : B \rightarrow A} \text{Sub} \quad \frac{\Delta, \Pi \vdash N : C \quad \Gamma, C, \Pi \vdash P : B}{\Gamma, \Delta, \Pi \vdash ([i/N, j]P) : B} \text{Sub}}{\Gamma, \Delta, \Pi \vdash (([i/N, j]M)([i/N, j]P)) : A} \text{App}$$

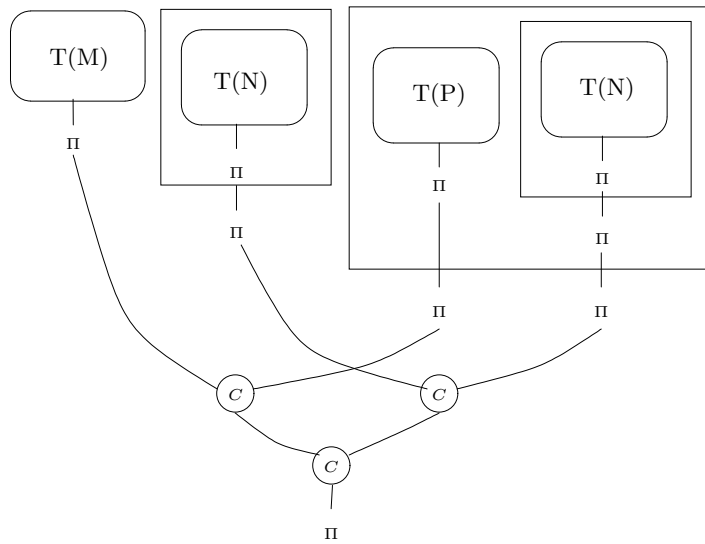
and its translation is the proof net



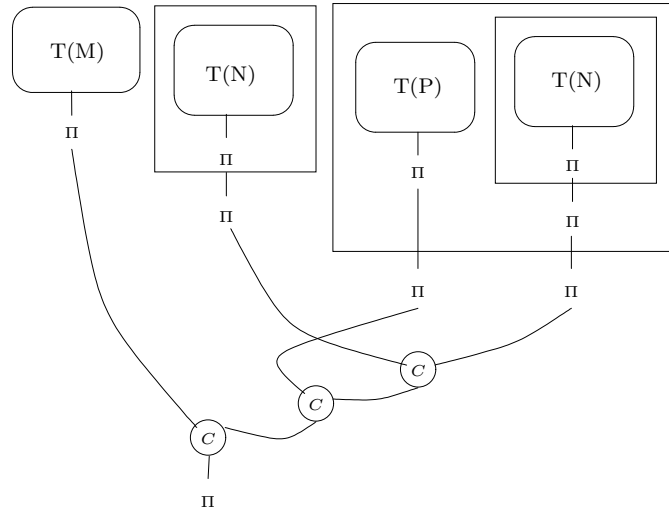
We eliminate the  $c - b$  cut, then the  $b - b$  cut, and thus we get the following proof net:



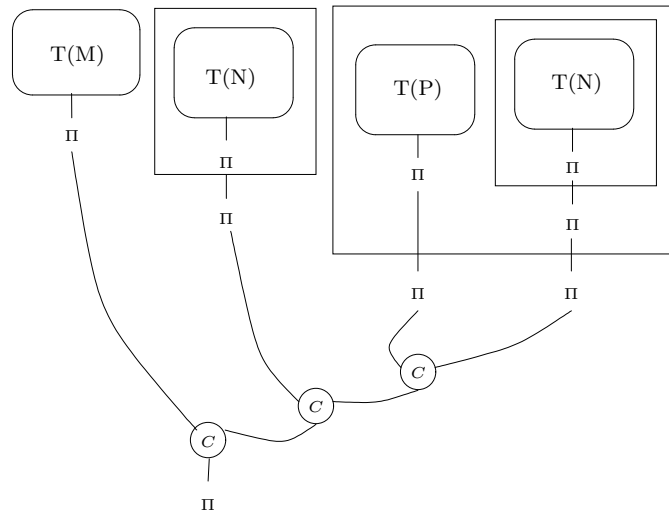
To get to the desired proof net we need to use the equivalence relations  $A$  and  $B$  which were introduced in Section 2. To better understand how to use them, we focus on the crucial informations, i.e. the contraction nodes and their connections with the nets  $T(M)$ ,  $T(N)$  and  $T(P)$ . Here is the net corresponding to the above net :



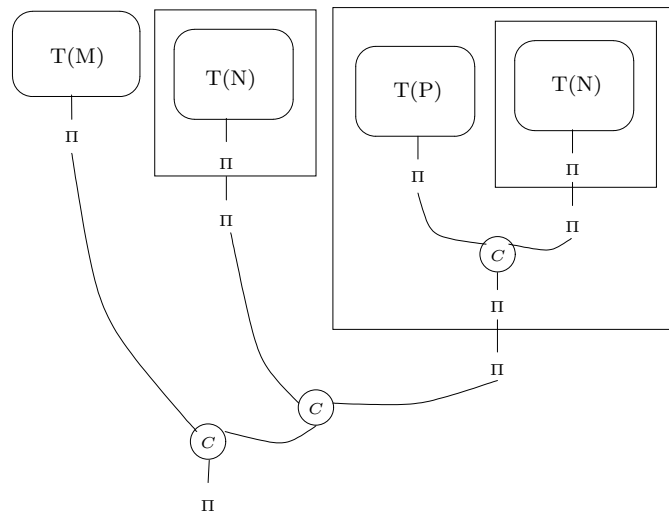
We use the associativity axiom  $A$  to obtain :



Again by associativity we get

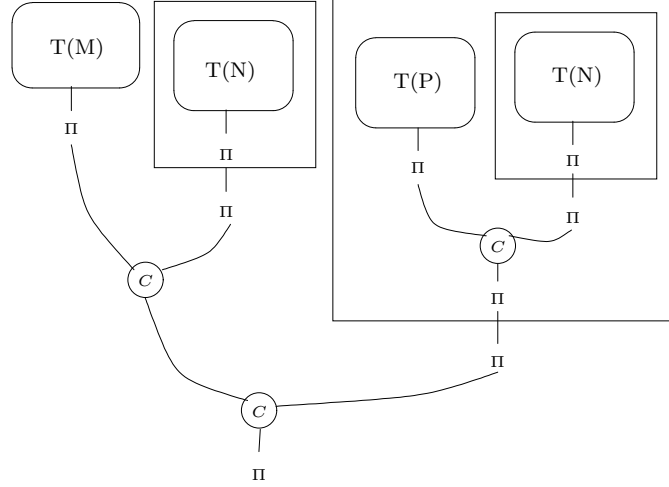


Using the  $B$  axiom we can put the contraction inside the box :





And finally we use the  $A$  axiom again to obtain the desired proof net :

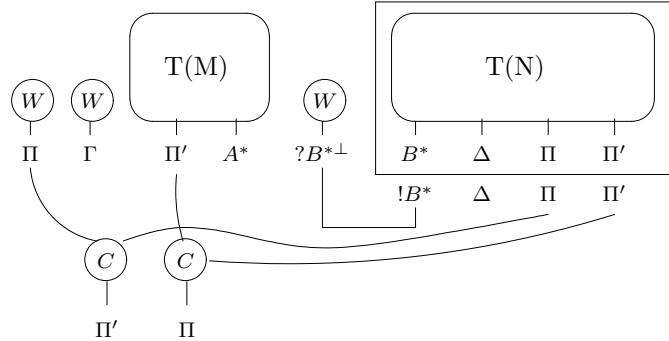


— **Rule  $e_1$**  :  $[i/N, j]\langle k \rangle M \longrightarrow \langle j + k - 1 \rangle M$  if  $i < k$

The typing environment can be split in four parts  $\Gamma$ ,  $\Delta$ ,  $\Pi$ , and  $\Pi'$ , where  $i$  is the length of  $\Gamma$ ,  $j$  is the length of  $\Delta$ , and  $k$  ( $k > i$ ) is the length of  $\Gamma$  plus the length of  $\Pi$  plus 1. The typing judgment of  $[i/N, j]\langle k \rangle M$  ends with

$$\frac{\frac{\Delta, \Pi, \Pi' \vdash N : B \quad \frac{\Pi' \vdash M : A}{\Gamma, B, \Pi, \Pi' \vdash \langle k \rangle M : A} \text{Weak}}{\Gamma, \Delta, \Pi, \Pi' \vdash [i/N, j]\langle k \rangle M : A} \text{Sub}}$$

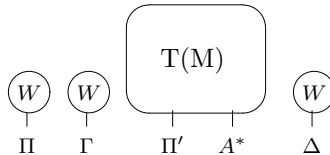
and its translation is the proof net



The typing judgment of  $\langle j + k - 1 \rangle M$  must end with:

$$\frac{\Pi' \vdash M : A}{\Gamma, \Delta, \Pi, \Pi' \vdash \langle j + k - 1 \rangle M : A} \text{Weak}$$

and its translation is the net



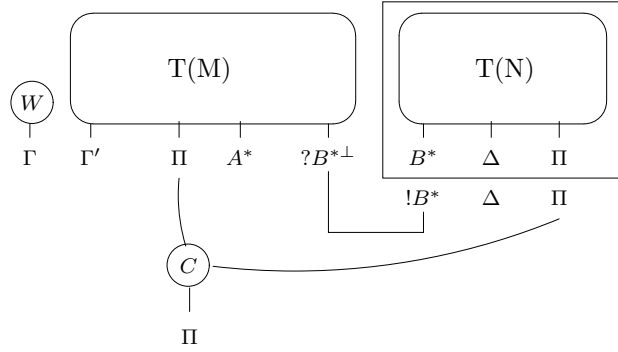
Starting from the first proof net, we eliminate the  $w - b$  cut, then we apply the  $wc$  rule and we finally obtain the desired proof net.

— **Rule**  $e_2 : [i/N, j]\langle k \rangle M \longrightarrow \langle k \rangle [i - k/N, j]M$  if  $i \geq k$

The typing environment can be split in four parts  $\Gamma, \Gamma', \Delta, \Pi$ , where  $i$  is the length of  $\Gamma$  plus the length of  $\Gamma'$ ,  $j$  is the length of  $\Delta$  and  $k$  ( $k \leq i$ ) is the length of  $\Gamma$ . The typing judgment of  $[i/N, j]\langle k \rangle M$  ends with

$$\frac{\frac{\Delta, \Pi \vdash N : B \quad \frac{\Gamma', B, \Pi \vdash M : A}{\Gamma, \Gamma', B, \Pi \vdash \langle k \rangle M : A} \text{Weak}}{\Gamma, \Gamma', \Delta, \Pi \vdash [i/N, j]\langle k \rangle M : A} \text{Sub}}$$

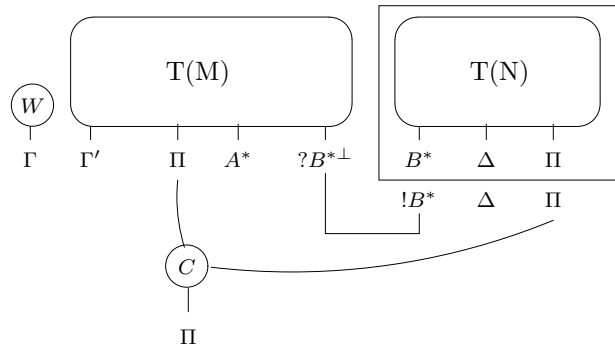
and its translation is the proof net



The typing judgment of  $\langle k \rangle [i - k/N, j]M$  must end with:

$$\frac{\frac{\Delta, \Pi \vdash N : B \quad \Gamma', B, \Pi \vdash M : A}{\Gamma', \Delta, \Pi \vdash [i - k/N, j]M : A} \text{Sub}}{\Gamma, \Gamma', \Delta, \Pi \vdash \langle k \rangle [i - k/N, j]M : A} \text{Weak}$$

and its translation is the proof net



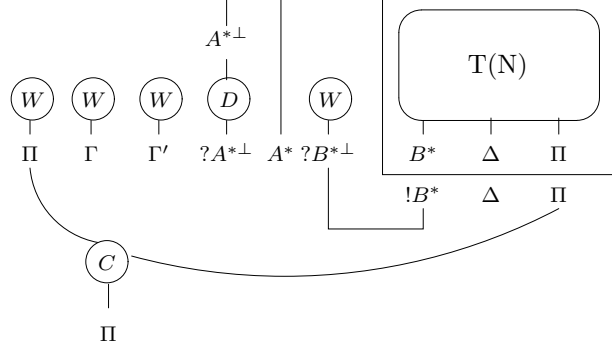
We notice that the two nets are already the same. This is the first of the exception cases of the lemma.

— **Rule**  $n_1 : [i/N, j]\underline{k} \longrightarrow \underline{k}$  if  $i > k$

The typing environment can be split in five parts  $\Gamma, A, \Gamma', \Delta, \Pi$ , where  $i$  is the length of  $\Gamma$  plus the length of  $\Gamma'$  plus 1,  $j$  is the length of  $\Delta$  and  $k$  ( $k < i$ ) is the length of  $\Gamma$ . The typing judgment of  $[i/N, j]\underline{k}$  ends with

$$\frac{\Delta, \Pi \vdash N : B \quad \overline{\Gamma, A, \Gamma', B, \Pi \vdash \underline{k} : A} \text{Ax}}{\Gamma, A, \Gamma', \Delta, \Pi \vdash [i/N, j]\underline{k} : A} \text{Sub}$$

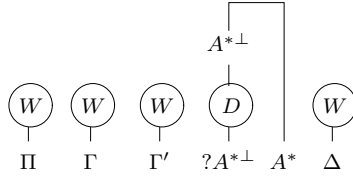
and its translation is the proof net



The typing judgment of  $\underline{k}$  must end with:

$$\frac{}{\Gamma, A, \Gamma', \Delta, \Pi \vdash \underline{k} : A}$$

and its translation is the proof net



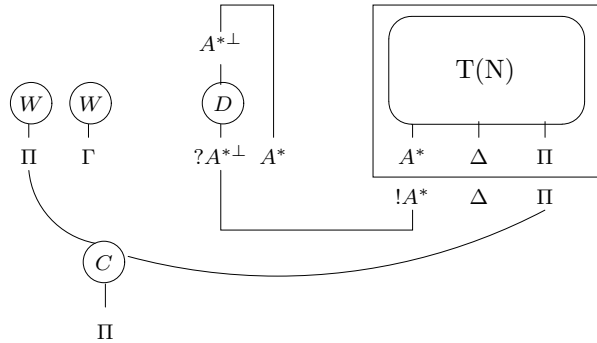
To reduce the first proof net into the second one it is enough to eliminate the  $w - b$  cut and to apply the  $wc$  rule.

— **Rule  $n_2$**  :  $[i/N, j]i \rightarrow \langle i \rangle N$

The typing environment can be split in three parts  $\Gamma, \Delta, \Pi$ , where  $i$  is the length of  $\Gamma$  and  $j$  is the length of  $\Delta$ . The typing judgment of  $[i/N, j]i$  ends with

$$\frac{\frac{\Delta, \Pi \vdash N : A \quad \overline{\Gamma, A, \Pi \vdash i : A}}{Sub} Ax}{\Gamma, \Delta, \Pi \vdash [i/N, j]i : A}$$

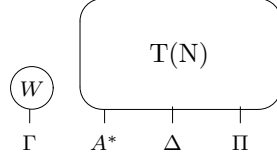
and its translation is the proof net



The typing judgment of  $\langle i \rangle N$  must end with:

$$\frac{\Delta, \Pi \vdash N : A}{\Gamma, \Delta, \Pi \vdash \langle i \rangle N : A} Weak$$

and its translation is the proof net



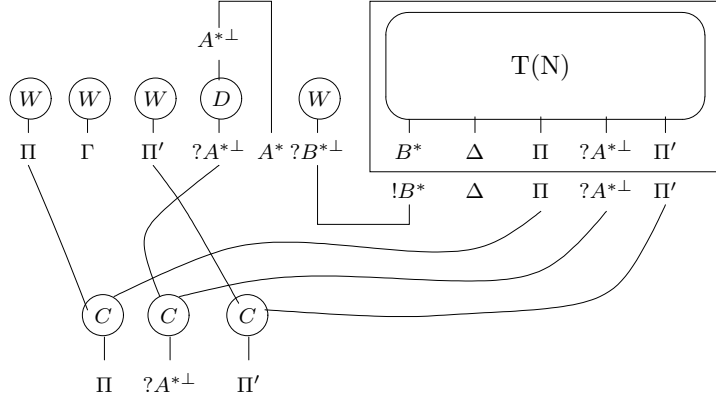
Starting from the first proof net, we eliminate the  $d - b$  cut, then the  $Ax - cut$  cut, and we apply the  $wc$  rule to obtain the desired proof net.

- **Rule  $n_3$**  :  $[i/N, j]k \rightarrow j + k - 1$  if  $i < k$

The typing environment can be split in five parts  $\Gamma, \Delta, \Pi, A, \Pi'$ , where  $i$  is the length of  $\Gamma$ ,  $j$  is the length of  $\Delta$  and  $k$  ( $k > i$ ) is the length of  $\Gamma$  plus the length of  $\Pi$  plus 1. The typing judgment of  $[i/N, j]k$  ends with

$$\frac{\Delta, \Pi, A, \Pi' \vdash N : B \quad \overline{\Gamma, B, \Pi, A, \Pi' \vdash k : A} \quad Ax}{\Gamma, \Delta, \Pi, A, \Pi' \vdash [i/N, j]k : A} \quad Sub$$

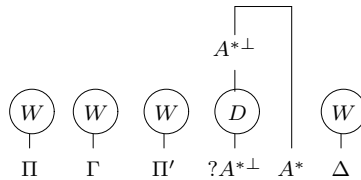
and its translation is the proof net



The typing judgment of  $j + k - 1$  must end with:

$$\overline{\Gamma, \Delta, \Pi, A, \Pi' \vdash j + k - 1 : A} \quad Ax$$

and its translation is the proof net



As for the  $n_1$  rule, we eliminate the  $w - b$  cut, then we apply three times the  $wc$  rule to achieve the desired result.

- **Rule  $c_1$**  :  $[i/N, j][k/P, l]M \rightarrow [k/[i - k/N, j]P, j + l - 1]M$  if  $k \leq i < k + l$

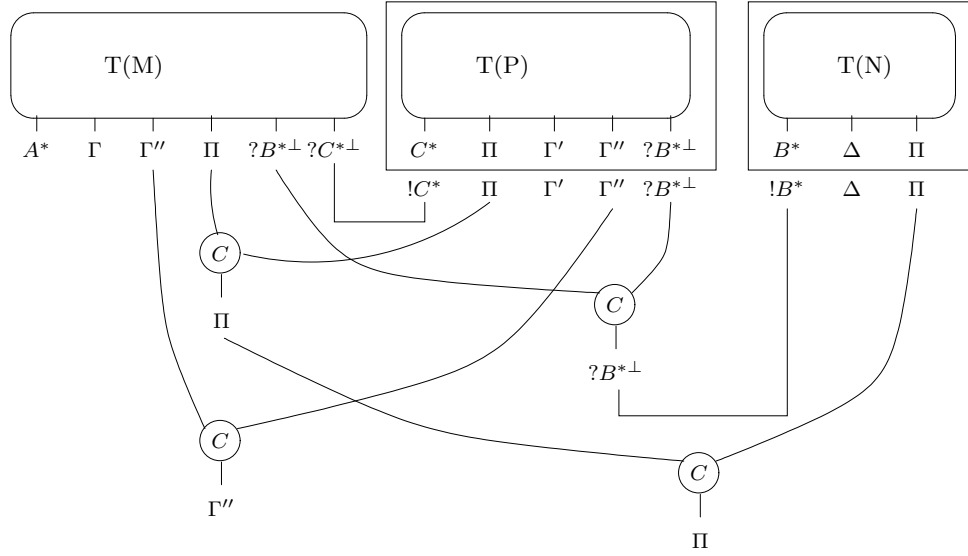
The typing environment can be split into five parts  $\Gamma, \Gamma', \Delta, \Pi, \Pi'$ , where  $i$  is the length of  $\Gamma$  plus the length of  $\Gamma'$ ,  $j$  is the length of  $\Delta$ ,  $k$  ( $k \leq i$ ) is the length of  $\Gamma$  and  $l$  ( $k + l > i$ ) is the length of  $\Gamma'$  plus the length of  $\Pi$  plus 1. The typing judgment of  $[i/N, j][k/P, l]M$  ends with

$$A \quad \frac{\mathbb{A}, \Pi, \Pi' \vdash N : B \quad \Gamma', B, \Pi, \Pi' \vdash P : C \quad \Gamma, C, \Pi' \vdash M : A}{\Gamma, \Pi, \Pi' \vdash N \circ P \circ M : A}$$

length of  $\Gamma$  and  $l$  is the length of  $\Gamma'$ . The typing judgment of  $[i/N, j][k/P, l]M$  ends with

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

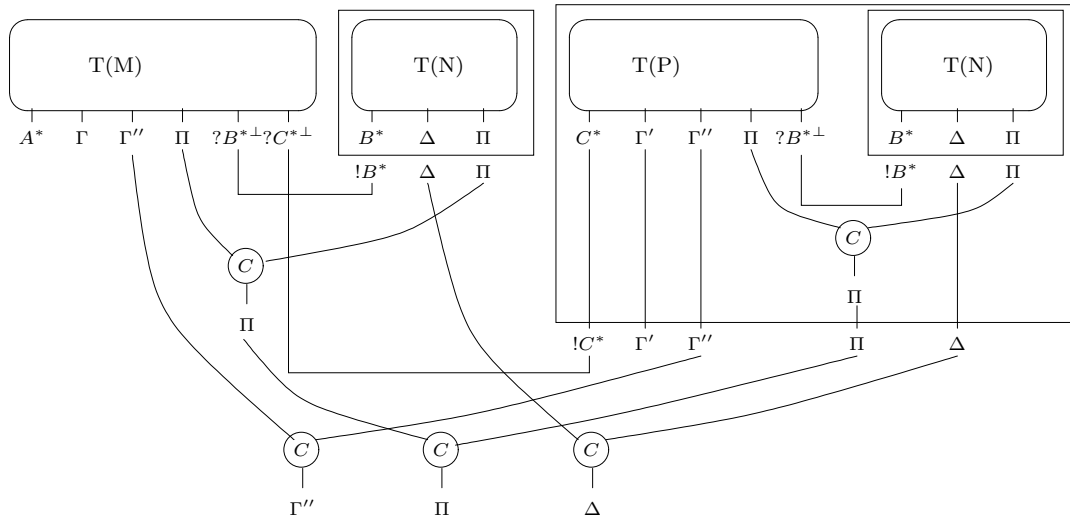
and its translation is the proof net



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

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

and its translation is the proof net



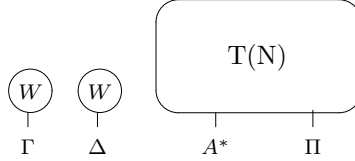
Starting from the first proof net, we eliminate the  $c - b$  cut, then the  $b - b$  cut, and we apply the equivalence rules  $A$  and  $B$  to obtain the desired proof net.

— **Rule  $d$**  :  $\langle i \rangle \langle j \rangle M \longrightarrow \langle i + j \rangle M$

The typing environment can be split in three parts  $\Gamma$ ,  $\Delta$ ,  $\Pi$ , where  $i$  is the length of  $\Gamma$  and  $j$  is the length of  $\Delta$ . The typing judgment of  $\langle i \rangle \langle j \rangle M$  ends with

$$\frac{\frac{\Pi \vdash M : A}{\Delta, \Pi \vdash \langle j \rangle M : A} \textit{Weak}}{\Gamma, \Delta, \Pi \vdash \langle i \rangle \langle j \rangle M : A} \textit{Weak}$$

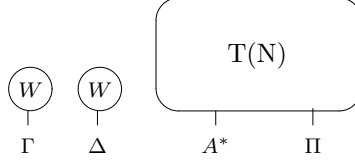
and its translation is the proof net



The typing judgment of  $\langle i + j \rangle M$  must end with:

$$\frac{\Pi \vdash M : A}{\Gamma, \Delta, \Pi \vdash \langle i + j \rangle M : A} \textit{Weak}$$

and its translation is the proof net



We notice that the two proof nets are already the same. This is the second of the exception cases of the lemma. □

#### 4.4. The proof of strong normalization of $\lambda_{ws}$

We are now able to show strong normalization of  $\lambda_{ws}$ . To achieve this result, we use the following abstract theorem (see for example (Ferreira, Kesner and Puel 1999)) :

**Theorem 4.3** 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  $\lambda_{ws}$ -terms,  $R_1$  as  $\lambda_{ws} - \{e_2, d\}$ ,  $R_2$  as  $\{e_2, d\}$ ,  $\mathcal{O}'$  as the set of proof nets,  $T$  the translation given in Section 4.2 and  $R'$  as the reduction  $R_E$ , then, by the Theorem 4.3 and the fact that the system including the rules  $\{e_2, d\}$  is strongly normalizing (David and Guillaume 1999; David and Guillaume 2001), we can conclude :

**Theorem 4.4 (Strong normalization of  $\lambda_{ws}$ )** The typed  $\lambda_{ws}$ -calculus is strongly normalizing.

### 5. A named version of the $\lambda_{ws}$ -calculus

In this section we present a version of typed  $\lambda_{ws}$  with named variables *à la Church*<sup>§</sup>. 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*.

The terms of this calculus are given by the following grammar, where  $A$  denotes a type and  $\Gamma$  and  $\Delta$  denote sets of variables:

$M ::=$	$x$	<i>variable</i>
	$  \lambda x : A.M$	<i>abstraction</i>
	$  (MM)$	<i>application</i>
	$  \Delta M$	<i>label</i>
	$  M[x, M, \Gamma, \Delta]$	<i>substitution</i>

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  $N$  ( $\Gamma$  is a subset of the type environment of  $M$ , not containing  $x$ ) and the variables  $\Delta$  do not appear in  $M$  ( $\Delta$  is a subset of the type environment of  $N$ ).

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

Terms are identified modulo  $\alpha$ -conversion so that bound variables can be systematically renamed. Indeed, we have  $\lambda y : A.y[x, z, \emptyset, \emptyset] =_{\alpha} \lambda y' : A.y'[x, z, \emptyset, \emptyset]$  and  $\lambda y : A.y[x, z, \emptyset, \emptyset] =_{\alpha} \lambda y : A.y[x', z, \emptyset, \emptyset]$  and  $\lambda l : A.y[x, z, \{l\}, \emptyset] =_{\alpha} \lambda l' : A.y[x, z, \{l'\}, \emptyset]$ .

The reduction rules of the calculus with names are given in Figure 4 (notice that rule  $b_1$  is a particular case of rule  $b_2$  with  $\Delta = \emptyset$ ). Remark that these rules may be applied to any term generated by the grammar, and they do not make use of any type information, which is only present in the terms due to their presentation *à la Church*.

The rule  $f$  should not be seen as a conditional reduction rule: as we work modulo  $\alpha$ -conversion, we can always find a term  $\alpha$ -equivalent to an abstraction  $\lambda y : A.M$  such that the condition imposed to the rule is true and thus no variable capture arises.

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 5. Remark that typing environments are managed here as sets (the relative order of variables in the environments does not matter). To make the proofs more readable, we will make a slight abuse of notation by not distinguishing explicitly between type environments (the capital greek letters on the left hand sides of the entailment relation) and sets of variables without type annotations (appearing in the labels of terms and in the explicit substitution constructors).

As we work modulo  $\alpha$ -conversion, we can suppose that in the rule *Weak* the set  $\Delta$  does not contain variables that are bound in  $M$ . We remark that whenever  $\Gamma \vdash M[x, N, \Delta, \Pi]$  is derivable, then  $\Gamma$  necessarily contains  $\Delta$  and  $\Pi$ , which are two *disjoint* sets of variables.

<sup>§</sup> It is of course possible to give a presentation *à la Curry* without type annotations on the abstracted variables.



(b <sub>1</sub> )	$(\lambda x : A.M)N \longrightarrow M[x, N, \emptyset, \emptyset]$	
(b <sub>2</sub> )	$(\Delta(\lambda x : A.M))N \longrightarrow M[x, N, \emptyset, \Delta]$	
(f)	$(\lambda y : A.M)[x, N, \Gamma, \Delta] \longrightarrow \lambda y : A.M[x, N, \Gamma \cup y, \Delta]$	if $y \notin FV(N)$
(a)	$(MP)[x, N, \Gamma, \Delta] \longrightarrow (M[x, N, \Gamma, \Delta]P[x, N, \Gamma, \Delta])$	
(e <sub>1</sub> )	$\Lambda M[x, N, \Gamma, \Delta] \longrightarrow (\Delta \cup (\Lambda \setminus x))M$	$x \in \Lambda$
(e <sub>2</sub> )	$\Lambda M[x, N, \Gamma, \Delta] \longrightarrow (\Gamma \cap \Lambda)M[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)]$	$x \notin \Lambda$
(n <sub>1</sub> )	$y[x, N, \Gamma, \Delta] \longrightarrow y$	$y \neq x$
(n <sub>2</sub> )	$x[x, N, \Gamma, \Delta] \longrightarrow \Gamma N$	
(c <sub>1</sub> )	$M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta] \longrightarrow$	$x \in \Phi$
	$M[y, P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus x)]$	
(c <sub>2</sub> )	$M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta] \longrightarrow$	$x \notin \Phi \cup \Lambda$
	$M[x, N, (\Gamma \setminus \Phi) \cup y, \Delta \cup (\Phi \setminus \Gamma)]$ $[y, P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Phi \cap \Gamma]$	
(d)	$\Gamma \Delta M \longrightarrow (\Gamma \cup \Delta)M$	

Fig. 4. Reduction Rules of the  $\lambda_{ws}$ -calculus with named variables

$\frac{}{\Gamma, x : A \vdash x : A} Ax$	$\frac{\Gamma \vdash M : A \quad \Gamma \cap \Delta = \emptyset}{\Gamma, \Delta \vdash \Delta M : A} Weak$
$\frac{\Gamma \vdash M : B \rightarrow A \quad \Gamma \vdash N : B}{\Gamma \vdash (MN) : A} App$	$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightarrow B} Lamb$
$\frac{\Delta, \Pi \vdash N : A \quad \Gamma, x : A, \Pi \vdash M : B \quad (\Gamma, x : A) \cap \Delta = \emptyset}{\Delta, \Gamma, \Pi \vdash M[x, N, \Gamma, \Delta] : B} Sub$	

Fig. 5. Typing rules for the  $\lambda_{ws}$ -calculus with named variables

As expected the  $\lambda_{ws}$ -calculus with names enjoys the subject reduction property.

**Theorem 5.1 (Subject Reduction)** If  $\Psi \vdash R : C$  and  $R \longrightarrow R'$ , then  $\Psi \vdash R' : C$ .

*Proof.* The proof proceeds by induction on the structure of the term  $R$ . If the reduction takes place at an internal position of  $R$ , it is easy to see that one gets the expected result by applying the induction hypothesis to the reduced subterm.

Otherwise, the reduction takes place at the root of the term  $R$ , and we must consider all the possible cases. The pattern of the proof is quite simple: from the shape of  $R$  and the fact that  $\Psi \vdash R : C$ , we determine the last rules applied in the typing derivation, and isolate some subderivations  $\pi_1, \dots, \pi_n$  from which it is easy to reconstruct a typing derivation of  $\Psi \vdash R' : C$ .

— Rule  $b_1$ :  $R = (\lambda x : A.M)N$  reduces to  $M[x, N, \emptyset, \emptyset] = R'$ .

Since  $R = (\lambda x : A.M)N$  the typing derivation must necessarily be of the form

$$\frac{\frac{\frac{\pi_1}{\Psi, x : A \vdash M : C} (Lamb)}{\Psi \vdash \lambda x : A.M : A \rightarrow C} (App)}{\Psi \vdash (\lambda x : A.M)N : C} (App)$$

Now, we can easily build a valid typing derivation for  $M[x, N, \emptyset, \emptyset] = R'$  as follows:

$$\frac{\frac{\pi_1}{\Psi, x : A \vdash M : C} \quad \frac{\pi_2}{\Psi \vdash N : A}}{\Psi \vdash M[x, N, \emptyset, \emptyset] : C} \text{ (Sub)}$$

- Rule  $b_2$ :  $R = (\Delta(\lambda x : A.M))N$  reduces to  $M[x, N, \emptyset, \Delta] = R'$ . Now, due to the shape of  $R$ , the typing derivation must necessarily be of the form

$$\frac{\frac{\frac{\pi_1}{\Gamma, x : A \vdash M : C}}{\Gamma \vdash \lambda x : A.M : A \rightarrow C} \text{ (Lamb)} \quad \frac{\Gamma \cap \Delta = \emptyset}{\Gamma, \Delta \vdash \Delta(\lambda x : A.M) : A \rightarrow C} \text{ (Weak)} \quad \frac{\pi_2}{\Gamma, \Delta \vdash N : A}}{\Gamma, \Delta \vdash (\Delta(\lambda x : A.M))N : C} \text{ (App)}$$

where  $\Psi$  is actually split into  $\Gamma$  and  $\Delta$ . Since  $x$  is bound in  $\lambda x : A.M$  we can suppose that  $\Delta$  does not contain  $x$ , so that we can construct the derivation

$$\frac{\frac{\pi_1}{\Gamma, x : A \vdash M : C} \quad \frac{\pi_2}{\Gamma, \Delta \vdash N : A} \quad x : A \notin \Delta}{\Gamma, \Delta \vdash M[x, N, \emptyset, \Delta] : C} \text{ (Sub)}$$

- Rule  $f$ :  $R = (\lambda y : A.M)[x, N, \Gamma, \Delta]$  reduces to  $M[x, N, (\Gamma, x : B), \Delta] = R'$ , where  $y \notin FV(N)$ . Due to the shape of  $R$ , the typing derivation must necessarily be of the form

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\pi_2}{\Gamma, \Pi, x : B, y : A \vdash M : C}}{\Gamma, \Pi, x : B \vdash \lambda y : A.M : A \rightarrow C} \text{ (Lamb)} \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash (\lambda y : A.M)[x, N, \Gamma, \Delta] : A \rightarrow C} \text{ (Sub)}$$

where  $\Psi$  is actually split into  $\Gamma$ ,  $\Delta$  and  $\Pi$ . Since  $y$  is bound in  $\lambda y : A.M$  we can suppose that  $\Delta$  does not contain  $y$ , so that we can construct the derivation

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\pi_2}{\Gamma, \Pi, x : B, y : A \vdash M : C}}{\Gamma, \Delta, \Pi, y : A \vdash M[x, N, (\Gamma, y : A), \Delta] : C} \quad (\Gamma, x : B, y : A) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash \lambda y : A.M[x, N, (\Gamma, y : A), \Delta] : A \rightarrow C} \text{ (Sub)}$$

- Rule  $a$ :  $R = (MP)[x, N, \Gamma, \Delta]$  rewrites to  $(M[x, N, \Gamma, \Delta]P[x, N, \Gamma, \Delta]) = R'$  and the typing derivation for  $R$  has the shape

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\frac{\pi_2}{\Gamma, \Pi, x : B \vdash M : A \rightarrow C} \quad \frac{\pi_3}{\Gamma, \Pi, x : B \vdash P : A}}{\Gamma, \Pi, x : B \vdash (MP) : C} \text{ (App)}}{\Gamma, \Delta, \Pi \vdash (MP)[x, N, \Gamma, \Delta] : C} \quad (\Gamma, x : B) \cap \Delta = \emptyset \text{ (Sub)}$$

where  $\Psi$  is decomposed into  $\Gamma$ ,  $\Delta$  and  $\Pi$ . Now we can easily construct a derivation  $\pi'$

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\pi_2}{\Gamma, \Pi, x : B \vdash M : A \rightarrow C}}{\Gamma, \Delta, \Pi \vdash M[x, N, \Gamma, \Delta] : A \rightarrow C} \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash M[x, N, \Gamma, \Delta] : A \rightarrow C} \text{ (Sub)}$$

and a derivation  $\pi''$

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\pi_3}{\Gamma, \Pi, x : B \vdash P : A}}{\Gamma, \Delta, \Pi \vdash P[x, N, \Gamma, \Delta] : A} \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash P[x, N, \Gamma, \Delta] : A} \text{ (Sub)}$$

from which we obtain finally

$$\frac{\frac{\pi'}{\Gamma, \Delta, \Pi \vdash M[x, N, \Gamma, \Delta] : A \rightarrow C} \quad \frac{\pi''}{\Gamma, \Delta, \Pi \vdash P[x, N, \Gamma, \Delta] : A}}{\Gamma, \Delta, \Pi \vdash (M[x, N, \Gamma, \Delta]P[x, N, \Gamma, \Delta]) : C} \text{ (App)}$$

- Rule  $e_1$ :  $R = \Lambda M[x, N, \Gamma, \Delta]$  rewrites to  $(\Delta \cup (\Lambda \setminus x))M = R'$ , where  $x \in \Lambda$ . Due to the structure of  $R$ , the derivation necessarily has the form

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\pi_2}{\Gamma', \Pi' \vdash M : C} \quad \Lambda \cap (\Gamma', \Pi') = \emptyset \text{ (Weak)}}{\Gamma, \Pi, x : B \vdash \Lambda M : C} \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash \Lambda M[x, N, \Gamma, \Delta] : C} \text{ (Sub)}$$

where  $\Psi$  decomposes into  $\Gamma, \Delta$  and  $\Pi$ . We know also, by the definition of rule  $e_1$ , that  $x \in \Lambda$ , so that  $\Lambda$  is actually composed of  $x$  plus some other variables coming in part from  $\Gamma$  and in part from  $\Pi$ , that is,  $\Lambda = (x : B, \Gamma'', \Pi'')$  with  $\Gamma = \Gamma', \Gamma''$ ,  $\Pi = \Pi', \Pi''$  and such that the set difference  $\Gamma \setminus \Lambda$  is  $\Gamma'$  and  $\Pi \setminus \Lambda$  is  $\Pi'$ .

Since  $\Pi' \subseteq \Pi$ , then it is evident that  $\Delta \cap \Pi' = \emptyset$ , and since  $\Gamma' \subseteq \Gamma$ , then  $\Delta \cap \Gamma' = \emptyset$  comes from the fact that  $(\Gamma, x : B) \cap \Delta = \emptyset$ . Indeed,  $(\Lambda \setminus x) \cap (\Gamma', \Pi') = \emptyset$  is a consequence of the constraint  $\Lambda \cap (\Gamma', \Pi') = \emptyset$  in the above typing derivation. We thus obtain

$$\frac{\frac{\pi_1}{\Gamma', \Pi' \vdash M : C} \quad (\Delta \cup (\Lambda \setminus x)) \cap (\Gamma', \Pi') = \emptyset}{\Gamma, \Delta, \Pi \vdash (\Delta \cup (\Lambda \setminus x))M : C} \text{ (Weak)}$$

- Rule  $e_2$ :  $R = \Lambda M[x, N, \Gamma, \Delta]$  rewrites to  $(\Gamma \cap \Lambda)M[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] = R'$ , where  $x \notin \Lambda$ .

Due to the structure of  $R$ , the typing derivation has necessarily the form

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\frac{\pi_2}{\Gamma', \Pi', x : B \vdash M : C} \quad (\Gamma', \Pi', x : B) \cap \Lambda = \emptyset}{\Gamma, \Pi, x : B \vdash \Lambda M : C} \text{ (Weak)}}{\Gamma, \Delta, \Pi \vdash \Lambda M[x, N, \Gamma, \Delta] : C} \quad (\Gamma, x : B) \cap \Delta = \emptyset \text{ (Sub)}$$

where  $\Psi$  decomposes into  $\Gamma, \Delta$  and  $\Pi$ . Furthermore, by the definition of rule  $e_2$ , we have that  $x \notin \Lambda$ , so that  $\Lambda$  can be written as  $\Gamma'', \Pi''$ , where  $\Gamma = \Gamma', \Gamma''$ ,  $\Pi = \Pi', \Pi''$ , and this means that  $\Gamma' = \Gamma \setminus \Lambda$ ,  $\Pi'' = \Lambda \setminus \Gamma$  and  $\Gamma', \Pi', \Lambda = \Gamma, \Pi$  and  $(\Gamma \cap \Lambda) \cup \Gamma' = \Gamma'' \cup \Gamma' = \Gamma$ . We can then build the required derivation

$$\frac{\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\pi_2}{\Gamma', \Pi', x : B \vdash M : C}}{\Delta, \Pi, \Gamma' \vdash M[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C} \text{ (Sub)}}{\Gamma, \Delta, \Pi \vdash (\Gamma \cap \Lambda)M[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C} \text{ (Weak)}$$

Notice that one has to check the side conditions of the typing rules. For *(Weak)*, we need  $(\Gamma \cap \Lambda) \cap (\Delta, \Pi, \Gamma') = \emptyset$ , but  $(\Gamma \cap \Lambda) = \Gamma''$ , and  $\Gamma'' \cap (\Delta, \Pi, \Gamma') = \emptyset$  because  $\Gamma = \Gamma', \Gamma''$  and  $(\Gamma, \Delta, \Pi)$  are well-formed environments. For *(Sub)*, we need  $(\Gamma', x : B) \cap (\Delta, \Pi'') = \emptyset$ . First of all we notice that  $(\Gamma, x : B) \cap \Delta = \emptyset$  holds because of the side condition of the *(Sub)* rule in the typing derivation for  $R$ . Secondly,  $\Gamma' \cap \Pi'' = \emptyset$  holds because  $(\Gamma, \Delta, \Pi)$  is a well formed environment. Last,  $x : B \notin \Pi''$  holds because  $x : B \notin \Lambda$  (definition of rule  $e_2$ ) and  $\Pi'' = \Lambda \setminus \Gamma$ .

- Rule  $n_1$ :  $R = y[x, N, \Gamma, \Delta]$  rewrites to  $y = R'$ . From the structure of  $R$ , we know that the derivation must be of the form

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\Gamma, \Pi, x : B \vdash y : C}{\Gamma, \Delta, \Pi \vdash y[x, N, \Gamma, \Delta] : C} (Ax) \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash y[x, N, \Gamma, \Delta] : C} (Sub)$$

where  $\Psi$  decomposes into  $\Gamma$ ,  $\Delta$  and  $\Pi$ . The seeked derivation is then simply

$$\frac{\Gamma, \Delta, \Pi \vdash y : C}{\Gamma, \Delta, \Pi \vdash y : C} (Ax)$$

— Rule  $n_2$ :  $R = y[x, N, \Gamma, \Delta]$  rewrites to  $\Gamma N = R'$

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : C} \quad \frac{\Gamma, \Pi, x : C \vdash x : C}{\Gamma, \Delta, \Pi \vdash y[x, N, \Gamma, \Delta] : C} (Ax) \quad (\Gamma, x : C) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash y[x, N, \Gamma, \Delta] : C} (Sub)$$

where  $\Psi$  decomposes into  $\Gamma$ ,  $\Delta$  and  $\Pi$ . Now, the side condition for  $(Sub)$  tells us that  $\Gamma \cap \Delta = \emptyset$ , and  $\Gamma, \Pi$  is well formed, so we can conclude that  $\Gamma \cap (\Delta, \Pi) = \emptyset$ , so we can build the required derivation as follows

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : C} \quad \Gamma \cap (\Delta, \Pi) = \emptyset}{\Gamma, \Delta, \Pi \vdash \Gamma N : C} (Weak)$$

— Rule  $c_1$ :  $R = M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta]$  rewrites to  $M[y, P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus x)] = R'$ , where  $x \in \Phi$ .

From the structure of  $R$ , we know that the derivation must be of the form

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\pi'}{\Gamma, x : B, \Pi \vdash M[y, P, \Lambda, \Phi] : C} \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta] : C} (Sub)$$

where  $\Psi$  decomposes into  $\Gamma$ ,  $\Delta$  and  $\Pi$ . Now,  $\pi'$  is a derivation that necessarily ends with an application of the  $(Sub)$  rule, so that the environment  $\Gamma, x : B, \Pi$  gets split into several subparts that are used to type the terms  $M$  and  $P$ . Looking at the shape of the  $(Sub)$  rule, we see that in general this splitting divides  $\Pi$  into three pairwise disjoint components (each of which possibly empty):  $\Pi_\Lambda$ , that is part of  $\Lambda$ ,  $\Pi_\Phi$ , which is part of  $\Phi$ , and a  $\Pi'$  which is common to the typing environments used to type  $M$  and  $P$ . Similarly,  $\Gamma$  decomposes into pairwise disjoint  $\Gamma_\Lambda$ ,  $\Gamma_\Phi$  and  $\Gamma'$ . And then  $\Lambda = \Gamma_\Lambda \cup \Pi_\Lambda$  and  $\Phi = \Gamma_\Phi \cup \Pi_\Phi \cup x : B$ . We also know that  $x$  is in the typing environment of  $P$  since  $x \in \Phi$ , so it must appear in the typing environment of  $P$ .

To sum all this up, the derivation  $\pi'$  must be of the form

$$\frac{\frac{\pi_2}{\Gamma', \Gamma_\Phi, x : B, \Pi', \Pi_\Phi \vdash P : A} \quad \frac{\pi_3}{\Gamma', \Gamma_\Lambda, y : A, \Pi', \Pi_\Lambda \vdash M : C} \quad (\Lambda, y : A) \cap \Phi = \emptyset}{\Gamma, x : B, \Pi \vdash M[y, P, \Lambda, \Phi] : C} (Sub)$$

Now, from  $\pi_1$  and  $\pi_2$  we can first of all build the derivation  $\pi''$ , where we use the fact that, since  $\Pi$  and  $\Gamma$  are disjoint, and  $\Lambda = \Gamma_\Lambda \cup \Pi_\Lambda$ , we know that  $\Pi_\Lambda$  can be written as  $\Lambda \setminus \Gamma$

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\pi_2}{\Gamma', \Gamma_\Phi, x : B, \Pi', \Pi_\Phi \vdash P : A} \quad (\Gamma', \Gamma_\Phi, x : B) \cap (\Delta \cup (\Lambda \setminus \Gamma)) = \emptyset}{\Delta, \Pi_\Lambda, \Gamma', \Gamma_\Phi, \Pi', \Pi_\Phi \vdash P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : A} (Sub)$$

Notice that the side condition holds because, on one side,  $\Pi$  and  $\Gamma$  are disjoint, on the other side  $x \notin \Pi$  since  $\Gamma, x : B, \Pi$  is well-formed, and finally because we know from the

side conditions of the typing derivation for  $R$  that  $\Delta$  is disjoint from  $\Gamma, x : B$ .

Now, since  $y$  is bound in  $M[y, P, \Lambda, \Phi]$ , we can also suppose that  $\Delta$  does not contain  $y$ , so that we can build the following derivation  $\pi'''$

$$\frac{\frac{\pi''}{\Delta, \Pi_\Lambda, \Gamma', \Gamma_\Phi, \Pi', \Pi_\Phi \vdash P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : A} \quad \frac{\pi_3}{\Gamma', \Gamma_\Lambda, y : A, \Pi', \Pi_\Lambda \vdash M : C}}{\Delta, \Gamma', \Gamma_\Lambda, \Gamma_\Phi, \Pi', \Pi_\Lambda, \Pi_\Phi \vdash M[y, P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Gamma_\Lambda, \Delta \cup (\Gamma_\Phi \cup \Pi_\Phi)] : C} \text{ (Sub)}$$

Where the side condition reads

$$(y : A, \Gamma_\Lambda) \cap (\Delta, \Gamma_\Phi, \Pi_\Phi) = \emptyset$$

This holds because, on one side,  $\Gamma_\Lambda$  is disjoint from  $\Gamma_\Phi$  by definition, disjoint from  $\Pi_\Phi$  because  $\Gamma \cap \Pi = \emptyset$  and disjoint from  $\Delta$  because  $\Gamma \cap \Delta = \emptyset$  by the side conditions of the typing derivation for  $R$ ; on the other side, we have already seen that we can assume  $y$  is not in  $\Delta$  and we know from the side conditions in the typing derivation of  $R$  that  $y$  is not in  $\Phi$ , which is precisely  $\Gamma_\Phi \cup \Pi_\Phi$ .

Now, the conclusion of this derivation, once we apply all the equalities of environments that we have established up to now, actually becomes

$$\Delta, \Gamma, \Pi \vdash M[y, P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus x)] : C$$

so that  $\pi'''$  is the sought derivation.

- Rule  $c_2$ : the term  $R = M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta]$  rewrites to the term  $M[x, N, (\Gamma \setminus \Phi) \cup y, \Delta \cup (\Lambda \setminus \Gamma)][y, P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Phi \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] = R'$ , where  $x \notin \Phi \cup \Lambda$ .

From the structure of  $R$ , we know that the derivation must be of the form

$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\pi'}{\Gamma, x : B, \Pi \vdash M[y, P, \Lambda, \Phi] : C} \quad (\Gamma, x : B) \cap \Delta = \emptyset}{\Gamma, \Delta, \Pi \vdash M[y, P, \Lambda, \Phi][x, N, \Gamma, \Delta] : C} \text{ (Sub)}$$

where  $\Psi$  decomposes as  $\Gamma, \Delta, \Pi$ . As for the rule  $c_1$  the environment  $\Gamma$  decomposes as  $\Gamma_\Lambda, \Gamma_\Phi, \Gamma'$ ; the environment  $\Pi$  decomposes as  $\Pi_\Lambda, \Pi_\Phi, \Pi'$ ;  $\Lambda = \Gamma_\Lambda \cup \Pi_\Lambda$  and  $\Phi = \Gamma_\Phi \cup \Pi_\Phi$ . We also know that  $x \notin \Phi \cup \Lambda$ , so it must appear in the common typing environment of  $P$  and  $M$ .

The derivation  $\pi'$  must be of the form

$$\frac{\frac{\pi_2}{\Gamma', \Gamma_\Phi, x : B, \Pi', \Pi_\Phi \vdash P : A} \quad \frac{\pi_3}{\Gamma', \Gamma_\Lambda, y : A, x : B, \Pi', \Pi_\Lambda \vdash M : C} \quad (\Lambda, y : A) \cap \Phi = \emptyset}{\Gamma, x : B, \Pi \vdash M[y, P, \Lambda, \Phi] : C} \text{ (Sub)}$$

We know that  $\Pi_\Lambda$  can be written as  $\Lambda \setminus \Gamma$  and  $\Pi_\Phi$  can be written as  $\Phi \setminus \Gamma$ . Now, we can first of all build the derivation  $\pi''$

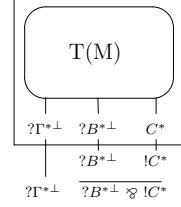
$$\frac{\frac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \frac{\pi_2}{\Gamma', \Gamma_\Phi, x : B, \Pi', \Pi_\Phi \vdash P : A} \quad (\Gamma', \Gamma_\Phi, x : B) \cap (\Delta \cup (\Lambda \setminus \Gamma)) = \emptyset}{\Delta, \Pi_\Lambda, \Gamma', \Gamma_\Phi, \Pi', \Pi_\Phi \vdash P[x, N, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C} \text{ (Sub)}$$

As in  $c_1$ , the side condition holds.

Since  $y$  is bound in  $M[y, P, \Lambda, \Phi]$ , then we can suppose by  $\alpha$ -conversion that  $\Delta$  does not contain  $y$ , and we also obtain the derivation  $\pi'''$

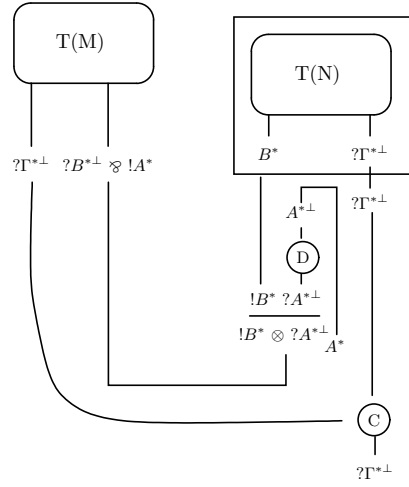


$$\frac{\Gamma, x : B \vdash M : C}{\Gamma \vdash \lambda x : B. M : B \rightarrow C} \textit{Lamb}$$



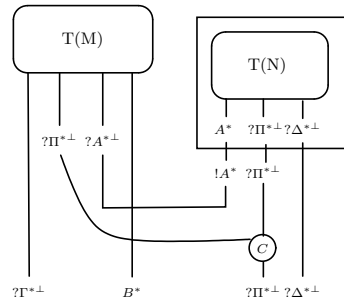
— If the term is an application and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



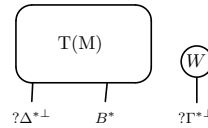
— If the term is a substitution and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

$$\frac{\Delta, \Pi \vdash N : A \quad \Gamma, x : A, \Pi \vdash M : B}{\Delta, \Gamma, \Pi \vdash M[x, N, \Gamma, \Delta] : B} \textit{Sub}$$



— Finally, if the term is a label and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

$$\frac{\Delta \vdash M : B}{\Gamma, \Delta \vdash \Gamma M : B} \textit{Weak}$$



We can clearly verify that the translation is identical to that given for  $\lambda_{ws}$  with de Bruijn indices. This is not surprising since the type derivations are similar in both formalisms.

The simulation of the reduction rules of the  $\lambda_{ws}$ -calculus with names by the reduction  $R_E$  is identical to that given in Section 4.2 for the  $\lambda_{ws}$ -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 6.1 (Simulation of  $\lambda_{ws}$  with names)** If  $t$   $\lambda_{ws}$ -reduces to  $t'$  in the formalism with names, then  $T(t) \rightarrow_{R_E}^+ T(t')$ , except for the rules  $e_2$  and  $d$  for which we have  $T(t) = T(t')$ .

We can then conclude the following:

**Theorem 6.2 (Strong Normalization of  $\lambda_{ws}$  with names)** The typed  $\lambda_{ws}$ -calculus with names is strongly normalizing.

## 7. 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 (Di Cosmo and Kesner 1997). We have proved that this system is strongly normalizing.

We have then proposed a natural translation from  $\lambda_{ws}$  into proof nets that immediately provides strong normalization of the typed version of  $\lambda_{ws}$ , a calculus featuring full composition of substitutions. The proof is extremely simple w.r.t the proof of PSN of  $\lambda_{ws}$  given in (David and Guillaume 1999; David and Guillaume 2001) and shows in some sense that  $\lambda_{ws}$ , 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  $\lambda_{ws}$  with named variables, instead of de Bruijn indices. This typed named version of  $\lambda_{ws}$  gives a better understanding of the mechanisms of the calculus. In particular, names allow to understand the manipulation of explicit weakenings in  $\lambda_{ws}$  without entering into the details of renaming of de Bruijn indices. However, the study of the properties of reduction, such as confluence and PSN, for non-typed or non well-formed 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

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

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



## Acknowledgments

We would like to thank Bruno Guillaume and Pierre-Louis Curien for their interesting remarks. We are grateful to José Espírito Santo for suggesting a simpler termination proof for  $R_E$  and to the anonymous referees of the current paper and of (Di Cosmo, Kesner and Polonovski 2000) for their contributions to improve the presentation of this document.

## References

- M. Abadi, L. Cardelli, P. L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.
- S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Logic in Computer Science (LICS)*, pages 211–222, 1992.
- 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.
- 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 en mathématiques.
- 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.
- 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.
- R. David and B. Guillaume. The  $\lambda_l$ -calculus. In D. Kesner, editor, *Proceedings of the 2nd Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, pages 2–13, July 1999.
- R. David and B. Guillaume. A  $\lambda$ -calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11, 2001.
- R. Di Cosmo and S. Guerrini. Strong normalization of proof nets modulo structural congruences. In P. Narendran and M. Rusinowitch, editors, *Tenth International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 75–89. Springer-Verlag, July 1999.
- 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. IEEE Computer Society Press, July 1997.
- R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. In J. Tiuryn, editor, *Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1784 of *Lecture Notes in Computer Science*, pages 63–81. Springer-Verlag, Mar. 2000.
- 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.
- J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.
- 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.
- G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of POPL*, pages 15–26, Albuquerque, New Mexico, 1992. Association for Computing Machinery.
- B. Guillaume. *Un calcul de substitution avec Étiquettes*. PhD thesis, Université de Savoie, 1999.
- J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of POPL*, pages 16–30, San Francisco, California, 1990. Association for Computing Machinery.

- P.-A. Mellès. Typed  $\lambda$ -calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the 2nd International Conference of Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*. Springer-Verlag, Apr. 1995.
- K. Rose. Explicit cyclic substitutions. In M. Rusinowitch and J.-L. Rémy, editors, *Proceedings of the Third International Workshop on Conditional Term Rewriting Systems (CTRS)*, number 656 in *Lecture Notes in Computer Science*, pages 36–50, 1992.