Proof nets and explicit substitutions

ROBERTO DI COSMO, DELIA KESNER and EMMANUEL POLONOVSKI

Mathematical Structures in Computer Science / Volume 13 / Issue 03 / June 2003, pp 409 - 450
DOI: 10.1017/S0960129502003791, Published online: 20 May 2003

Math. Struct. in Comp. Science (2003), *vol.* 13, *pp.* 409–450.
DOI: 10.1017/S0960129502003791 Printed in the United Kingdom

Proof nets and explicit substitutions

ROBERTO DI COSMO[†], DELIA KESNER[‡]
and EMMANUEL POLONOVSKI[†]

[†]*PPS (CNRS UMR 7126), Université Paris 7,*
175, rue du Chevaleret, Paris, France
Email: {dicosmo,polonovs}@pps.jussieu.fr

[‡]*LRI (CNRS UMR 8623), Bât 490, Université de Paris-Sud,*
91405 Orsay Cedex, France
Email: kesner@lri.fr

Received 31 March 2001; revised 15 April 2002

We refine the simulation technique introduced in Di Cosmo and Kesner (1997) to show strong normalisation of λ-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 show that cut elimination modulo this equivalence relation is terminating. We then show strong normalisation of the typed version of the λ_{ws}-calculus with de Bruijn indices (a calculus with full composition defined in David and Guillaume (1999)) using a translation from typed λ_{ws} to proof nets. Finally, we propose a version of typed λ_{ws} with named variables, which helps to give a better understanding of the complex mechanism of the explicit weakening notation introduced in the λ_{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 λ-calculus over the past decade. On the one hand, explicit substitutions provide an intermediate formalism that, by decomposing the β rule into more atomic steps, allows a better understanding of the execution models. On the other hand, 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 λ-calculus, which is the computational side of intuitionistic logic[†].

[†] Using various translations of the λ-calculus into proof nets, new abstract machines have been proposed that exploit the Geometry of Interaction and Dynamic Algebras (Girard 1989; Abramsky and Jagadeesan 1992; Danos 1990), leading to work on optimal reduction (Gonthier *et al.* 1992; Lamping 1990).

Abadi *et al* (1991) introduced the pioneer calculus with explicit substitutions, λ_σ, as a bridge between the classical λ-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 Normalisation': a calculus with explicit substitutions has PSN when all λ-terms that are strongly normalising using the traditional β-reduction rule are also strongly normalising with respect to the more refined reduction system defined using explicit substitutions. But λ_σ does *not* preserve β-strong normalisation, as shown by Mellies, who exhibited a well-typed term that, due to the substitution composition rules in λ_σ, is not λ_σ-strongly normalising (Melliès 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 normalisation, but also be confluent (in a very large sense that implies the ability to compose substitutions), and its typed version should be strongly normalising.

Meanwhile, in the linear logic community, many studies have focused on the connection between λ-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 β-reduction, as in Danos and Regnier (1995).

Finally, in Di Cosmo and Kesner (1997), the first two authors of this paper 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 λx (Rose 1992; Bloo and Rose 1995), which is 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 required of a 'good' calculus of explicit substitutions, and especially the strong normalisation in the presence of a reduction rule that is reminiscent of th2e 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 that is flexible enough to allow a natural and simple translation from David and Guillaume's λ_{ws}, which is a complex calculus of explicit substitution with de Bruijn indices and full composition (David and Guillaume 1999; 2001). This translation allows us to prove that typed λ_{ws} is strongly normalising, which is a new result confirming a conjecture in David and Guillaume (1999; 2001). Also, the fact that all information about variable order is lost in the translation suggests a version of typed λ_{ws} with named variables that is immediately proved to be strongly normalising. 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 λ_{ws} gives a better understanding of the mechanisms of labels existing in the calculus. In particular, names allow us to understand the fine manipulation of explicit weakenings in λ_{ws} without entering into the complicated details of renaming used in a de Bruijn setting.

The paper is organised as follows: we first recall the basic definitions of linear logic and proof nets, and introduce our refined reduction system for proof nets (Section 2), then prove that it is strongly normalising (Section 3). In Section 4 we recall the definition

of the λ_{ws} calculus with its type system, present the translation into proof nets, and show strong normalisation of typed λ_{ws}. Finally, we introduce a version of typed λ_{ws} with named variables (Section 5), which enjoys 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 normalisation.

MELL: Multiplicative Exponential Linear Logic

Let $\mathscr{A}$ be a set of *atomic formulae* equipped with an involutive[†] function $\perp : \mathscr{A} \to \mathscr{A}$, called *linear negation*.

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

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

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

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

The name MELL comes from the connectors $\otimes$ and $\parr$, 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:

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

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

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

MELL proof nets

To all sequent derivations in MELL it is possible to associate an object called a 'proof net', which allows us to abstract from many inessential details in a derivation, such as 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.

[†] A function f is involutive iff $f(f(p)) = p$.

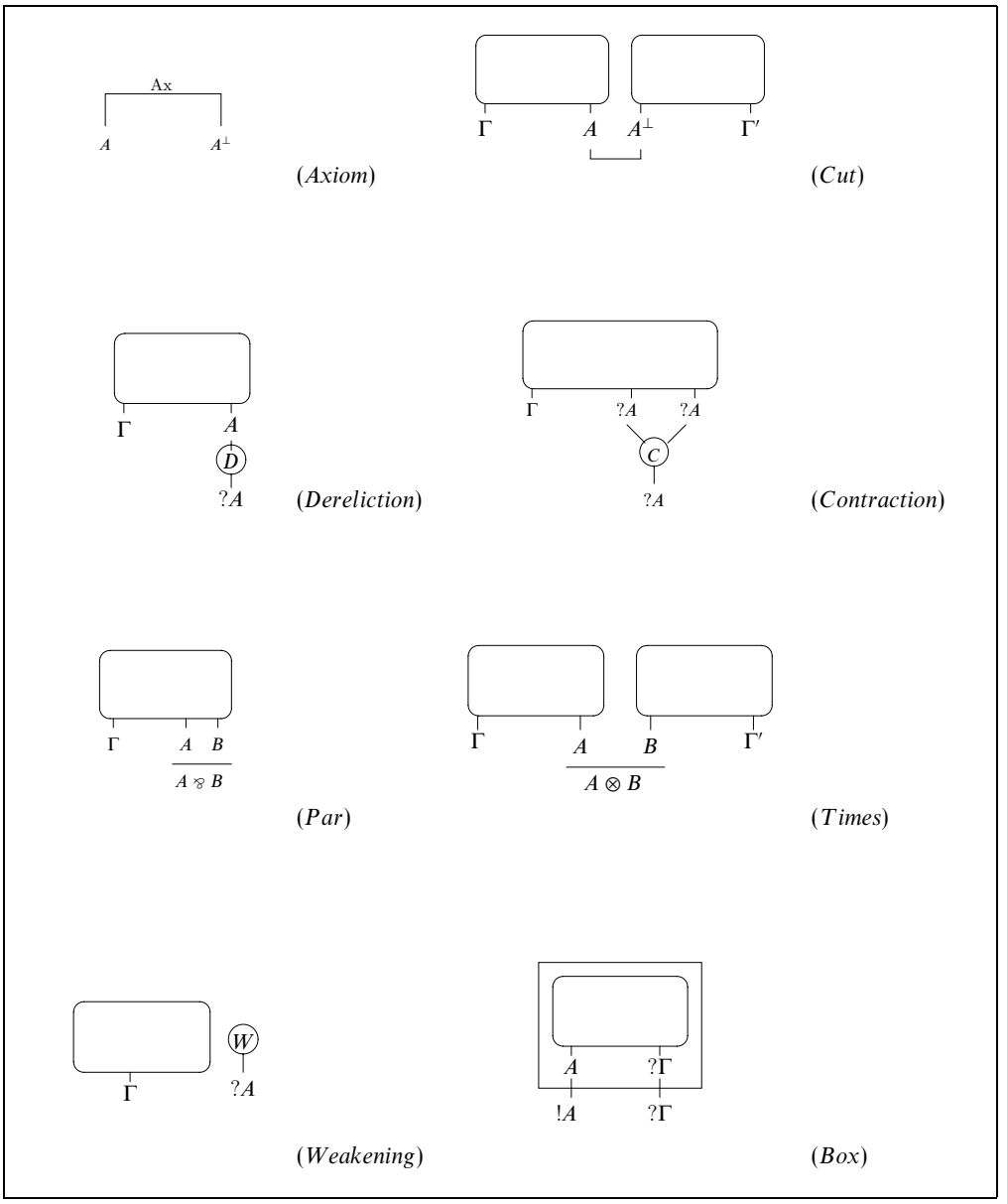


Fig. 1. MELL proof nets.

Proof nets are defined inductively by rules that follow closely the rules 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 $\Gamma, \Delta, \ldots$ really stands for a set of conclusions, each one with its own wire.

Each box has exactly one conclusion preceded by a !, which is called the 'principal' port (or formula), while the other conclusions are called 'auxiliary' ports (or formulae).

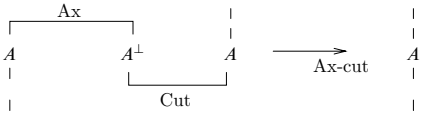
In what follows, we will sometimes write an axiom link as

$$\overline{A \qquad A^{\perp}}.$$

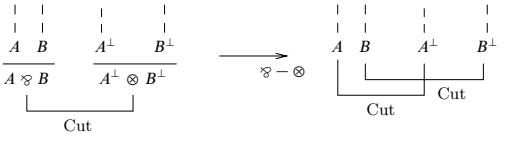
Reduction of proof nets

Proof nets are the 'computational object' behind linear logic, because there is a notion of reduction on them (also called 'cut elimination') that corresponds to the cut-elimination procedure on sequent derivations. The traditional reduction system for MELL is defined as follows:

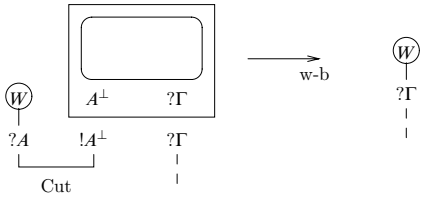
— Reduction acting on a cut $Ax - cut$, removing an axiom:



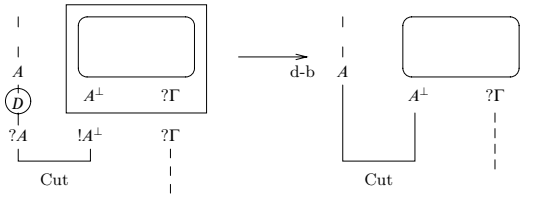
— Reduction acting on a cut $\parr - \otimes$:



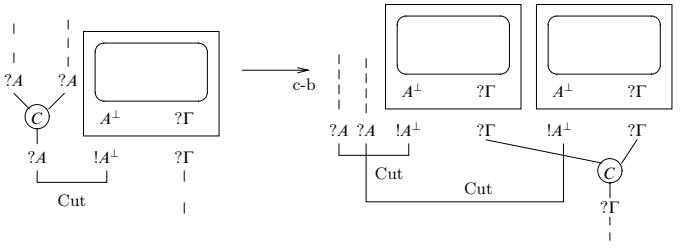
— Reduction acting on a cut $w - b$, erasing a box:



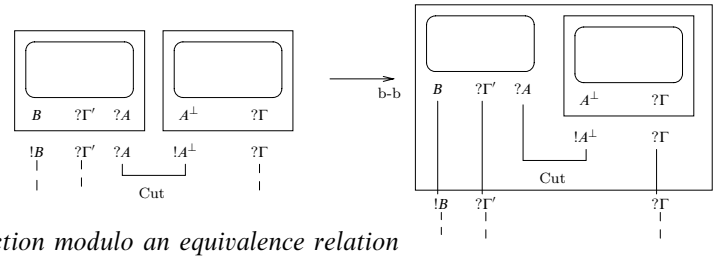
— Reduction acting on a cut $d - b$, opening a box:



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

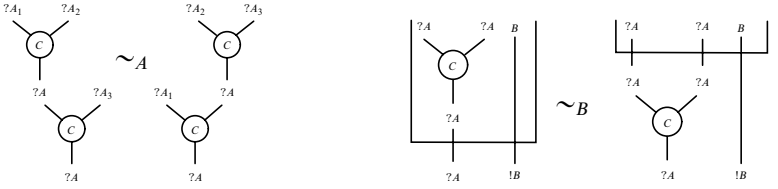


— Reduction acting on a cut $b - b$, absorbing a box into another:



Extended reduction modulo an equivalence relation

Unfortunately, the original notion of reduction on PN is not well adapted to simulating either the β rule of λ-calculus, or the rules dealing with propagation of substitution in explicit substitution calculi. This is because 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:



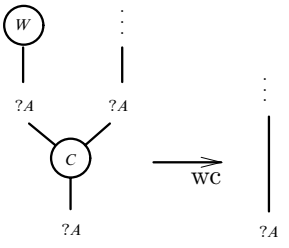
Equivalence A turns contraction into an associative operator, and corresponds to forgetting the order in which the contraction rule is used to build, for example, the derivation:

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

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

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

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



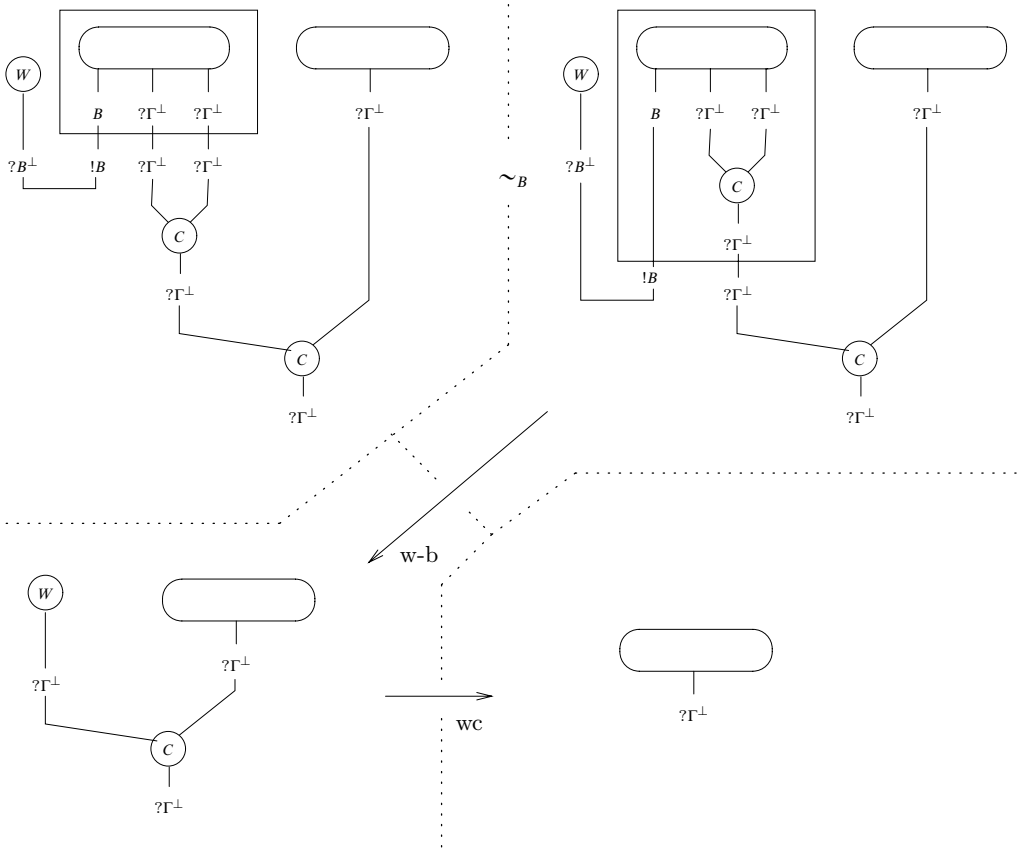
This rule allows us to simplify the proof on the left below into the proof on the right

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

Notation: In the following we will use R to mean the system made of rules $Ax-cut$, $\parr-\otimes$, $w-b$, $d-b$, $c-b$, $b-b$ and wc; we will use E to mean 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 $R_E^{\neg wc}$, which contain E, are actually defining a notion of *reduction modulo an equivalence relation*, so we write, for example, $r \longrightarrow_{R_E} s$ 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.

An example of reduction in R_E is given below:



The reduction R_E is flexible enough to allow an elegant simulation of β reduction and of explicit substitutions, but for that, we first need to establish that R_E is strongly normalising. We will show this property in the next section.

3. Termination of R_E

We know from Di Cosmo and Guerrini (1999) that $R_E^{\neg wc}$ 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 $R_E^{\neg wc}$, so that termination of R_E will follow from a well-known abstract lemma.

Let us first recall the following result from Di Cosmo and Guerrini (1999).

Lemma 3.1 (Termination of $R_E^{\neg wc}$). The relation $\longrightarrow_{R_E^{\neg wc}}$ 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. □

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

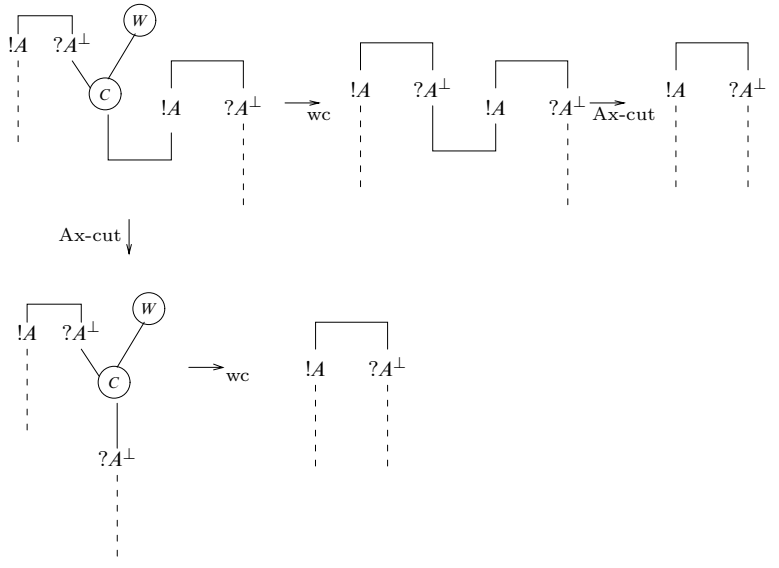
Lemma 3.3 (Postponement of wc with respect to $R_E^{\neg wc}$). Let t be a proof net. If $t \longrightarrow_{wc} \longrightarrow_{R_E^{\neg wc}} t'$, there is a sequence $t \longrightarrow^+_{R_E^{\neg wc}} \longrightarrow^*_{wc} t'$.

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

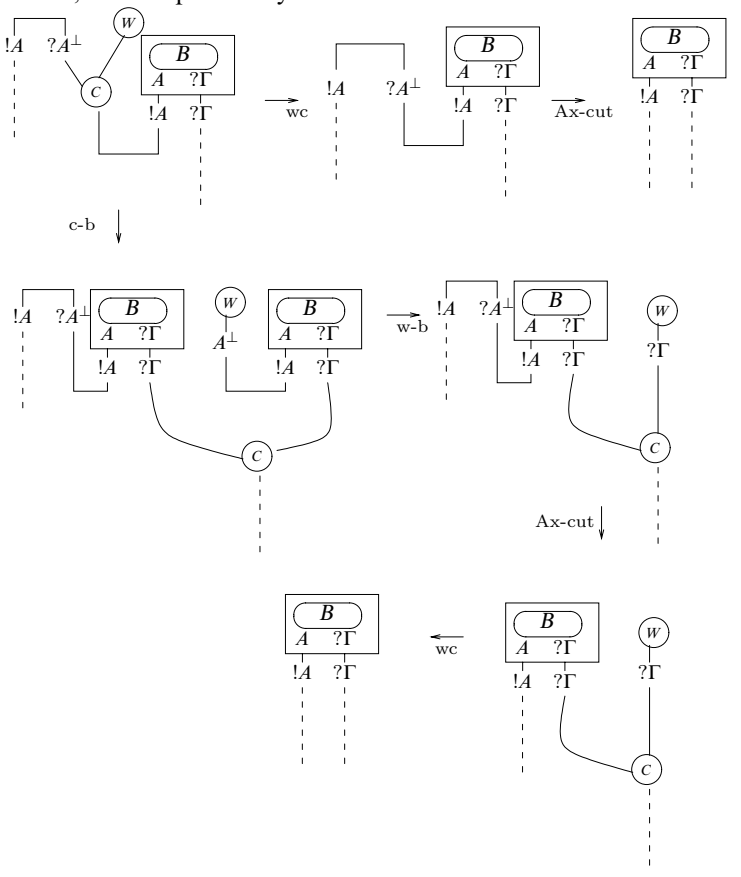
We do not give the details here for the cases of disjoint redexes: if we apply the wc-rule followed by a rule $R1$ in $R_E^{\neg wc}$ and if the redexes occur at disjoint positions, then it is evident that $R1$ can be applied first, followed by wc, to get the same result.

We now study all the remaining cases:

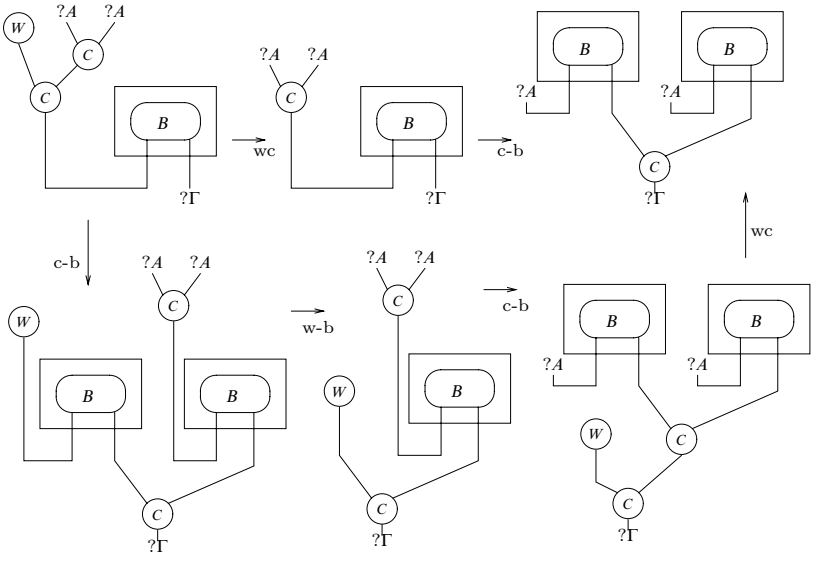
1 The rule $Ax - cut$, first possibility:



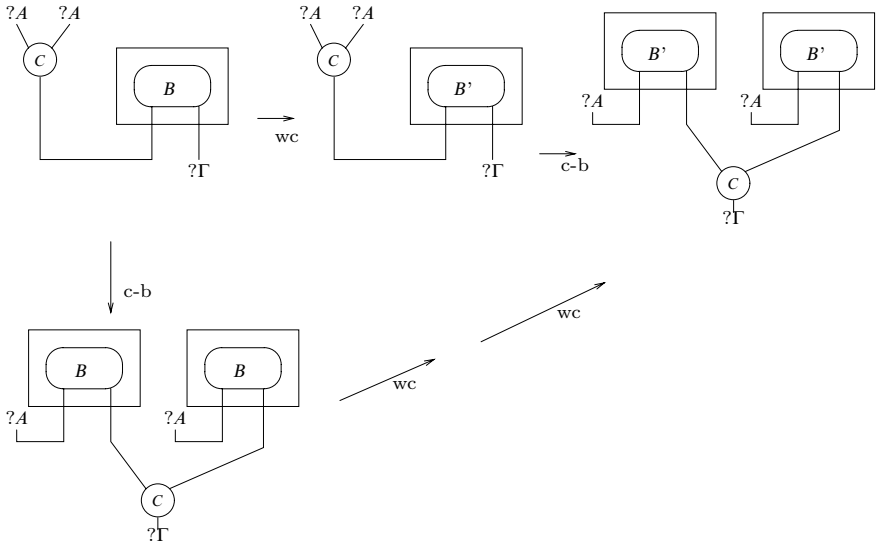
2 The rule $Ax - cut$, second possibility:



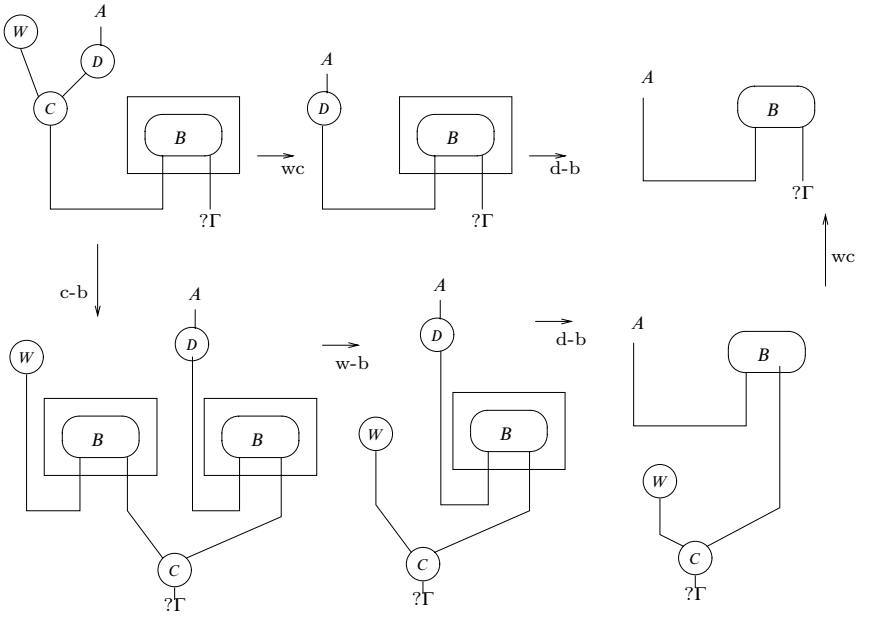
3 The rule $c - b$, first possibility:



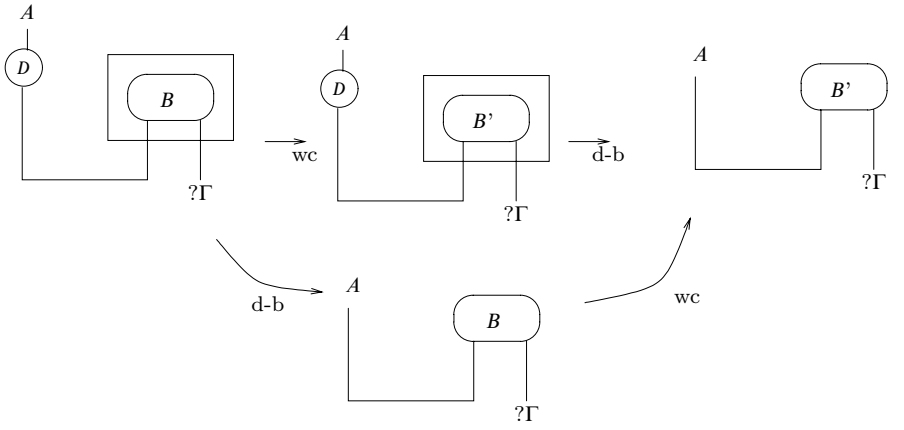
4 The rule $c - b$, second possibility:



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

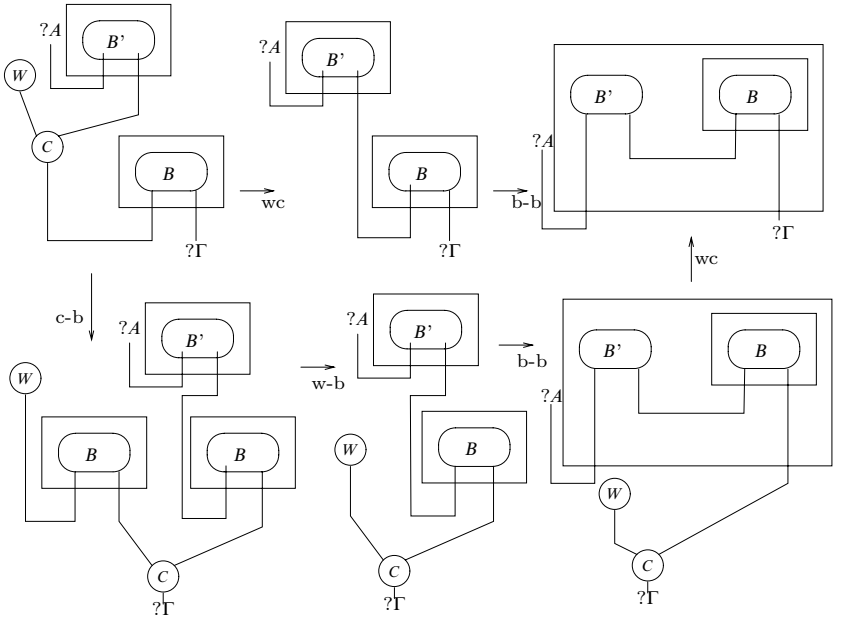


6 The rule $d - b$, second possibility:



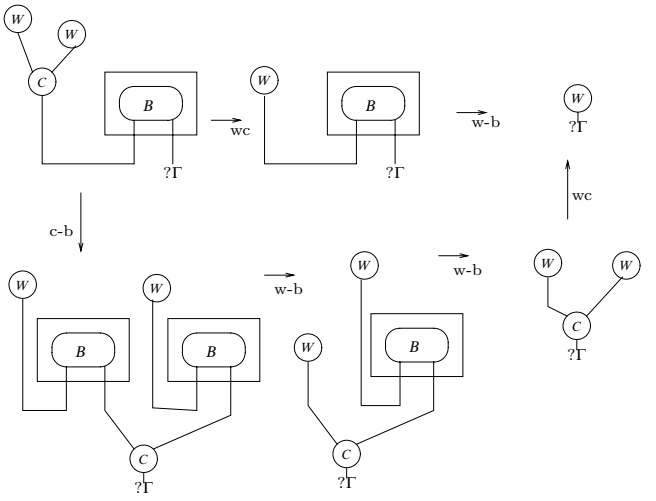
Notice that everything is happening as if the redexes were disjoint. This is due to the fact that the $d - b$ rule is non-duplicating and non-erasing with respect to boxes. As a consequence, the *wc*-redex is still preserved after the application of the $d - b$ rule.

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

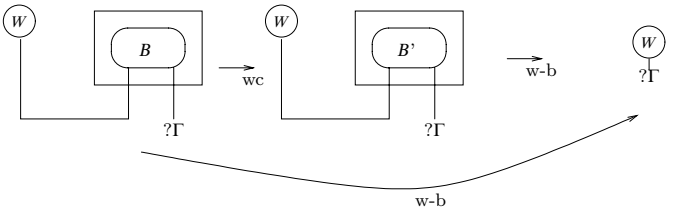


8 The rule $b - b$, second possibility. For the same reason as for $d - b$, the redexes are considered as disjoint.

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



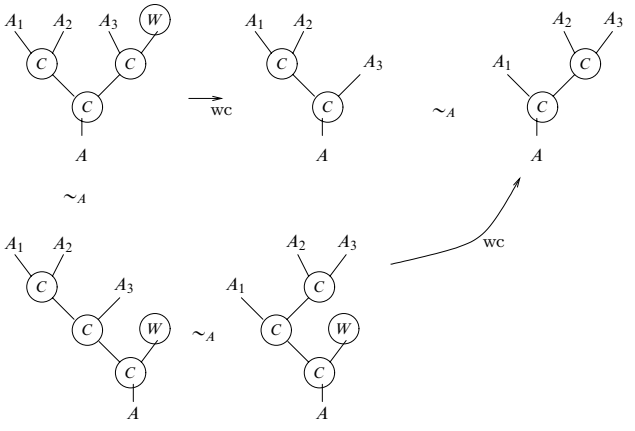
10 The rule $w - b$, second possibility:



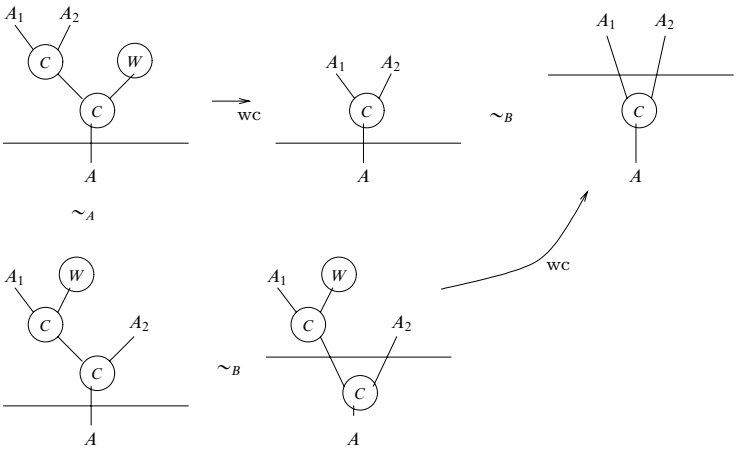
11 The rules ⅋-⊗ cut. Just note that in this case the redexes are disjoint.

Up to this point 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 with respect to 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 as follows:

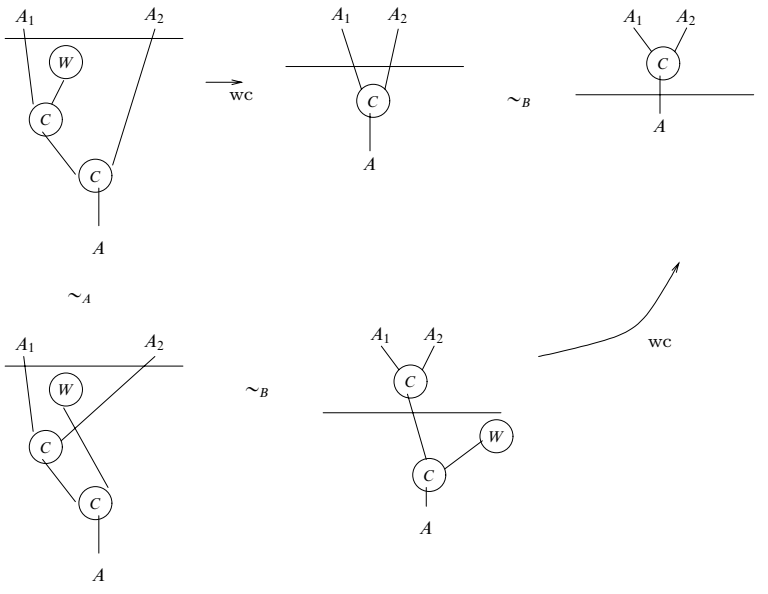
1 Associativity:



2 Box passing, first case:



3 Box passing second case:



□

Lemma 3.4 (Extraction of $R_E^{\neg wc}$). 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 that starts with $t \longrightarrow_{R_E^{\neg wc}} t'$, where t' is also a proof net, and continues with an infinite sequence S'. We write this sequence as $(t \longrightarrow_{R_E^{\neg wc}} t') \cdot S'$.

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

$$t \longrightarrow_{R_E} \ldots \longrightarrow_{R_E} \ldots \longrightarrow_{R_E} \ldots$$

We know, by Lemmas 3.2 and 3.1, that the systems wc and $R_E^{\neg wc}$ are both terminating, so it is not possible to have an infinite sequence made purely from wc or $R_E^{\neg wc}$. As a consequence, the infinite sequence of R_E-reductions must be an infinite alternation of non-empty finite sequences of wc and $R_E^{\neg wc}$.

Now, there are two cases: either the alternation of sequences starts with a $R_E^{\neg wc}$-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:

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

which can be written in another way as

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

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

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

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 $(t \longrightarrow_{R_E^{\neg wc}} t') \cdot S'$ such that S' is infinite again. If we iterate this procedure an arbitrary number times, we obtain an arbitrarily long sequence of $R_E^{\neg wc}$-reduction steps. This contradicts the fact that $R_E^{\neg wc}$ is terminating. □

4. From λ_{ws} with de Bruijn indices to PN

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

4.1. *The λ_{ws}-calculus*

The λ_{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 λ_{ws} are given by the following grammar:

M	::=	$\underline{n}$	*variable*
		λM	*abstraction*
		(MM)	*application*
		$\langle k\rangle M$	*label*
		$[i/M, j]M$	*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 the $j-1$ following indices are not free in M. These indices are used to split the typing environment of $[i/N, j]M$ into three parts: the first for free variables of M; the second for free variables of N; and the third 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 λ_{ws} are given in Figure 2 and the typing rules of λ_{ws} are given in Figure 3, where we suppose that $|\Gamma| = i$ and $|\Delta| = j$.

(b_1)	(λMN)	$\longrightarrow [0/N, 0]M$	
(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* $i \geqslant k$
(n_1)	$[i/N, j]\underline{k}$	$\longrightarrow \underline{k}$	*if* $i > k$
(n_2)	$[i/N, j]\underline{i}$	$\longrightarrow \langle i\rangle N$	
(n_3)	$[i/N, j]\underline{k}$	$\longrightarrow \underline{j+k-1}$	*if* $i < k$
(c_1)	$[i/N, j][k/P, l]M$	$\longrightarrow [k/[i-k/N, j]P, j+l-1]M$	*if* $k \leqslant i < k+l$
(c_2)	$[i/N, j][k/P, l]M$	$\longrightarrow [k/[i-k/N, j]P, l][i-l+1/N, j]M$	*if* $i \geqslant k+l$
(d)	$\langle i\rangle\langle j\rangle M$	$\longrightarrow \langle i+j\rangle M$	

Fig. 2. Reduction rules of λ_{ws} with de Bruijn indices.

Note that for each well-typed term of the λ_{ws}-calculus, there is only one possible typing judgment. This will simplify the proof of simulation of λ_{ws} by considering the unique typing judgment of terms.

As expected, the λ_{ws}-calculus enjoys the subject reduction property (Guillaume 1999).

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

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

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

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

Fig. 3. Typing rules for λ_{ws} with de Bruijn indices.

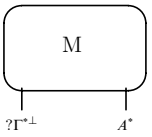
4.2. *Translation of types and terms of* λ_{ws}

We use the translation of types introduced in Danos *et al.* (1995) and 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 λ_{ws} whose typing judgment ends with the conclusion written below on the left, a proof net having the shape sketched below on the right:

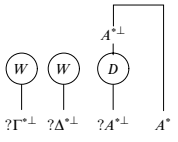
$$\frac{}{\Gamma \vdash M : A}$$



The formal definition of the translation T from λ_{ws}-terms into proof nets is as follows:

— 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

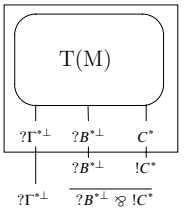
$$\frac{}{\Gamma, A, \Delta \vdash \underline{i} : A}\ Ax$$



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

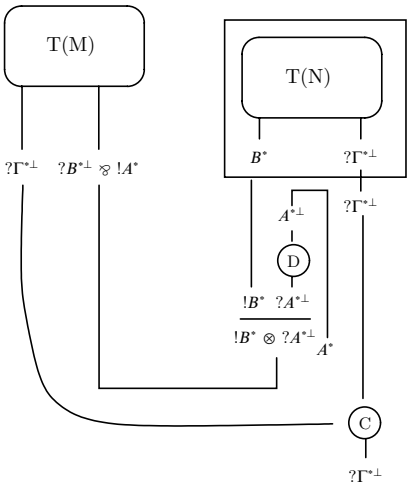
— If the term is a λ-abstraction and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



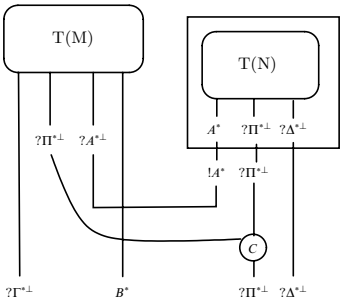
— If the term is an application and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



— 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

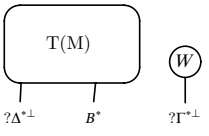
$$\frac{\Delta, \Pi \vdash N : A \quad \Gamma, A, \Pi \vdash M : B}{\Gamma, \Delta, \Pi \vdash [i/N, j]M : B}\ Sub$$



where i is the length of the list Γ and j is the length of the list Δ.

— 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

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



where i is the length of the list Γ.

4.3. *Simulating λ_{ws}-reduction*

We now verify that our notion of reduction R_E on PN simulates the λ_{ws}-reduction on typed λ_{ws}-terms. It is in this proof that we find the motivation for our choice of translation from λ-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 as 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 Γ, Δ, Π, etc. appear repeated many times. In order to make these pictures more readable, in the following proof only, we will abuse the notation slightly by simply writing Γ in place of its correct translation $?\Gamma^{*\perp}$.

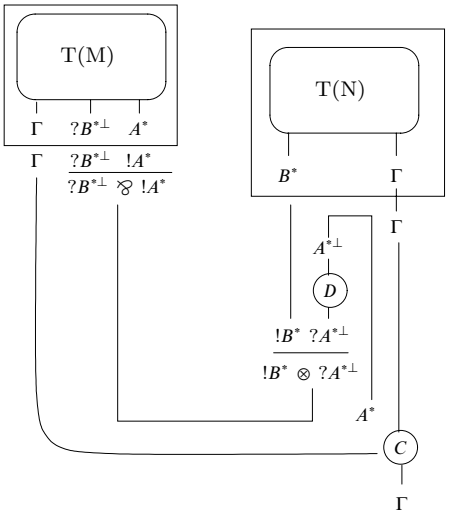
Lemma 4.2 (Simulation of λ_{ws}). The relation R_E simulates the λ_{ws}-reduction on typed terms: if $t \longrightarrow_{\lambda_{ws}} t'$, then $T(t) \longrightarrow^{+}_{R_E} T(t')$, except for the rules e_2 and d for which we have $T(t) = T(t')$.

Proof. The proof proceeds by cases on the reduction rule applied in the step $t \longrightarrow_{\lambda_{ws}} t'$. Since reductions λ_{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**:** $(\lambda MN) \longrightarrow [0/N, 0]M$
The typing judgment of (λMN) ends with

$$\dfrac{\dfrac{B, \Gamma \vdash M : A}{\Gamma \vdash \lambda M : B \to A}\ Lamb \qquad \Gamma \vdash N : B}{\Gamma \vdash ((\lambda M)N) : A}\ App$$

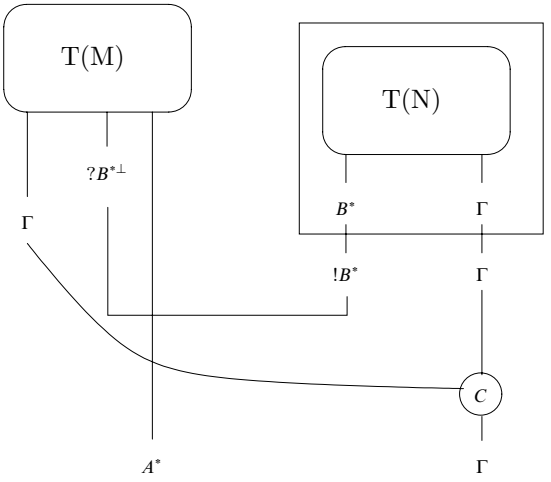
and its translation is the proof net



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

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

and its translation is the proof net



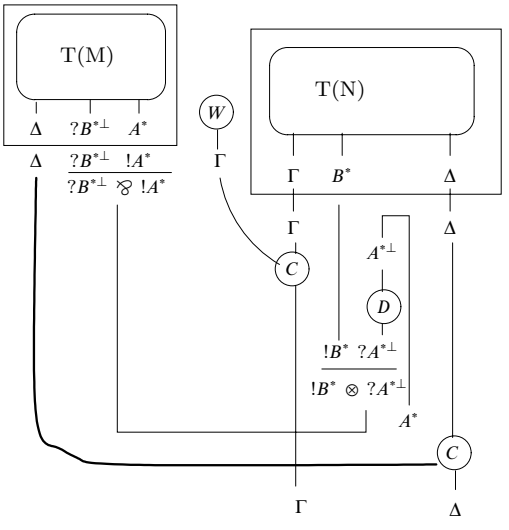
Starting from the first proof net, we eliminate the $\parr$-$\otimes$ cut, then the $d-b$ cut and finally the $Ax-cut$ cut to obtain the final proof net.

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

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

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

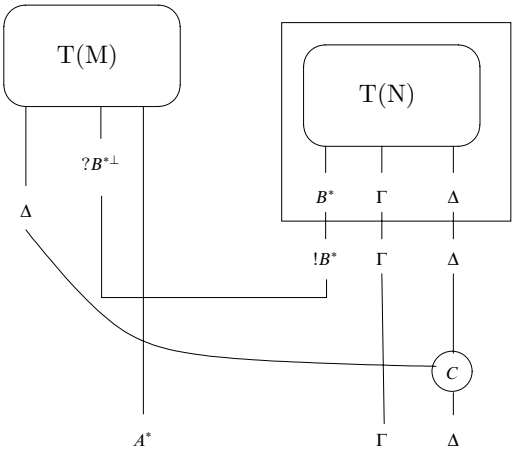
and its translation is the proof net



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

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

and its translation is the proof net



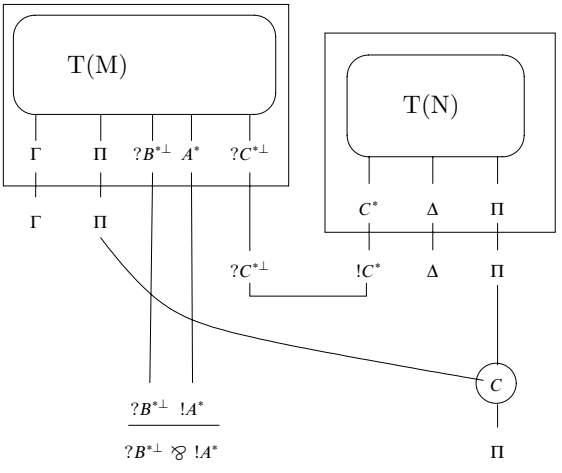
As for the b_1 rule, we eliminate the $\parr$-$\otimes$ cut, then the $d-b$ cut, and the $Ax-cut$ cut. Finally, we apply the *wc* rule to achieve the desired result.

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

The typing environment can be split into three parts Γ, Δ and Π, where i is the length of Γ and j is the length of Δ. The typing judgment of $[i/N, j]\lambda M$ ends with

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

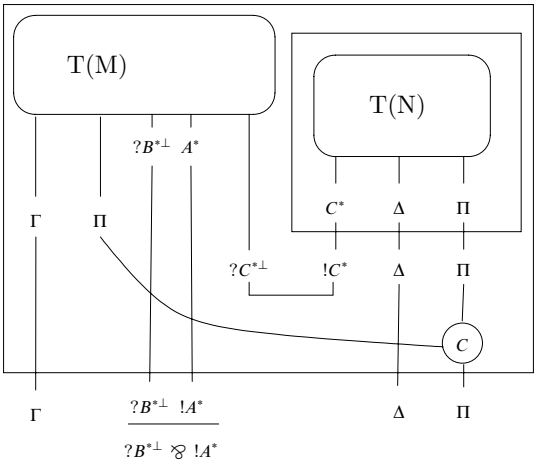
and its translation is the proof net



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

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

and its translation is the proof net



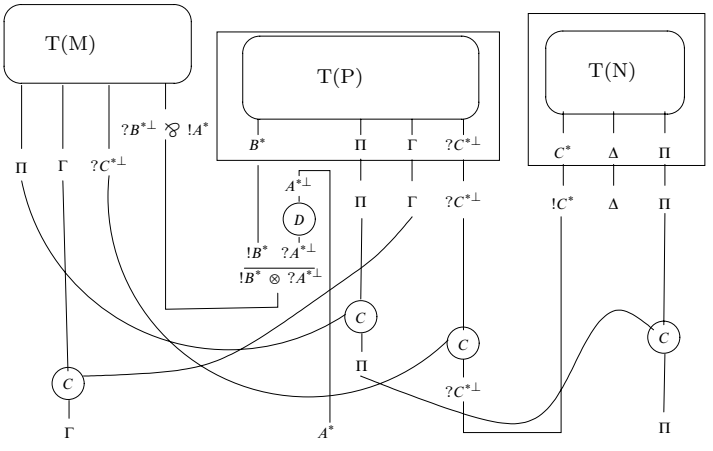
To reduce the first proof net into the second one, we must eliminate the $b - b$ cut, then use the equivalence relation B (we will show exactly how to use the equivalence relations in the case of the rule a).

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

The typing environment can be split into three parts Γ, Δ and Π, where i is the length of Γ and j is the length of Δ. The typing judgment of $[i/N, j](MP)$ ends with

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

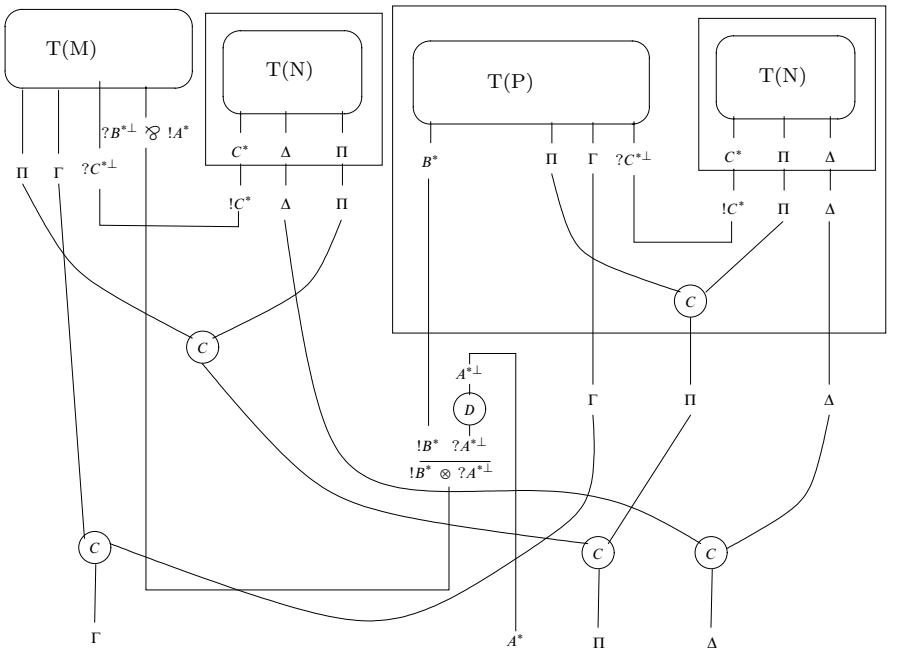
and its translation is the proof net



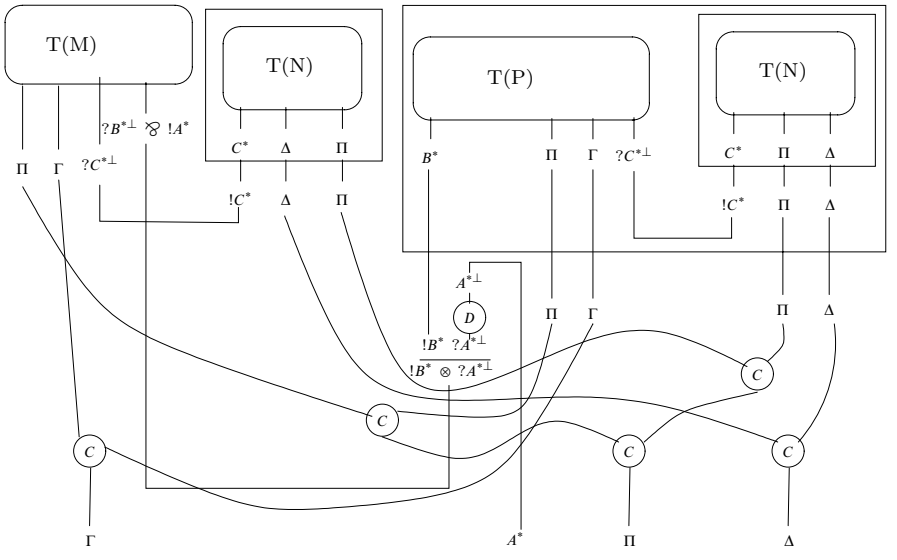
The typing judgment of $(([i/N, j]M)([i/N, j]P))$ must end with

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

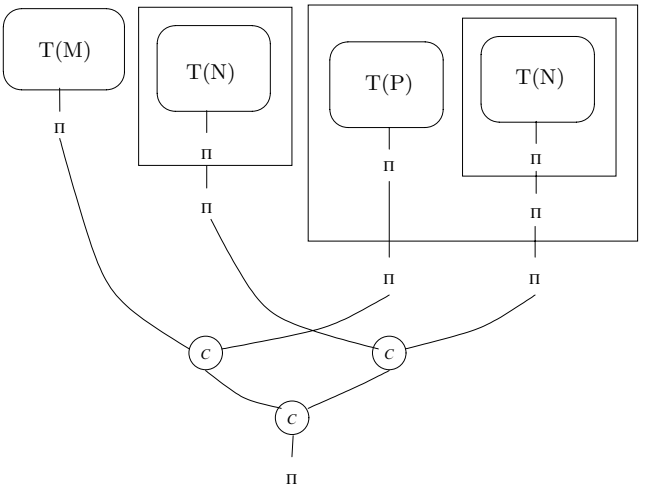
and its translation is the proof net



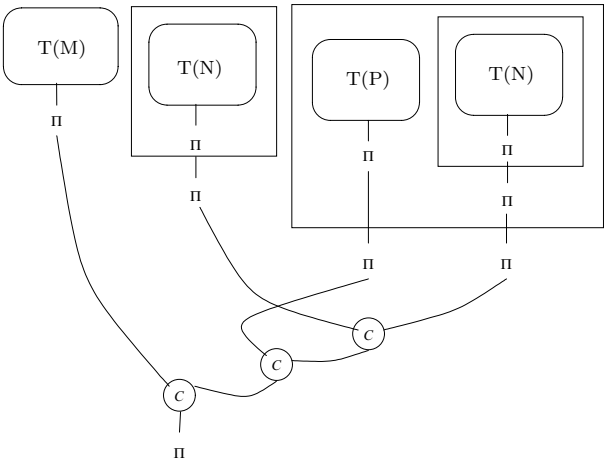
We eliminate the $c-b$ cut, then the $b-b$ cut, and thus we get the following proof net:



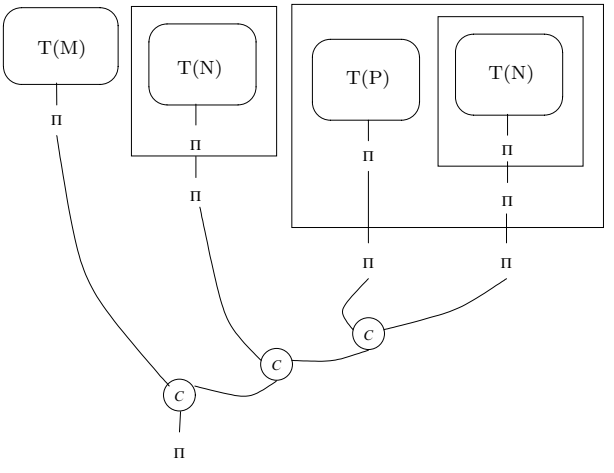
To get to the desired proof net, we need to use the equivalence relations A and B which were introduced in Section 2. To understand better how to use them, we focus on the crucial informations, that is, the contraction nodes and their connections with the nets $T(M)$, $T(N)$ and $T(P)$. The net corresponding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



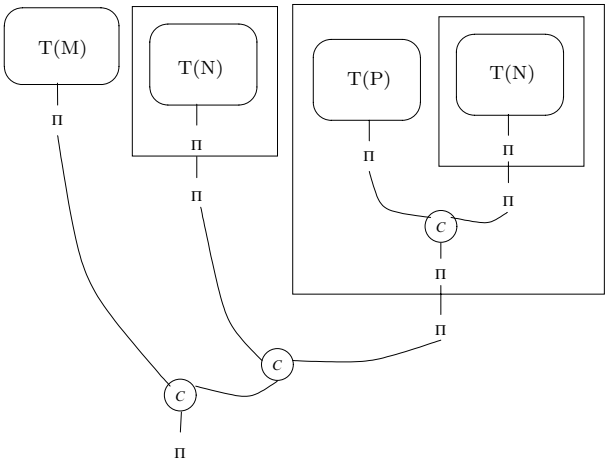
We use the associativity axiom A to obtain



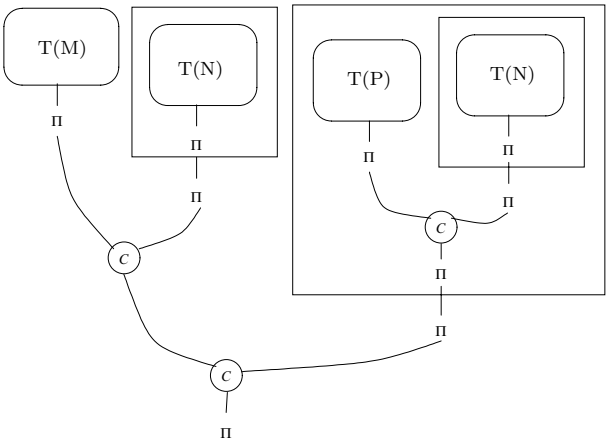
Again by associativity, we get



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



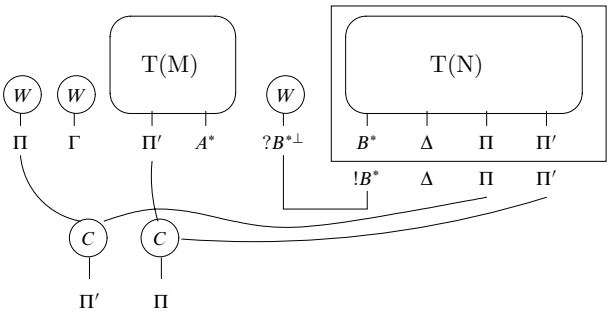
And, finally, we use the A axiom again to obtain the desired proof net:



— **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 into four parts Γ, Δ, Π and Π', where i is the length of Γ, j is the length of Δ, and k ($k > i$) is the length of Γ plus the length of Π plus 1. The typing judgment of $[i/N, j]\langle k\rangle M$ ends with

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

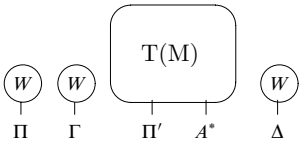
and its translation is the proof net



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

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

and its translation is the net



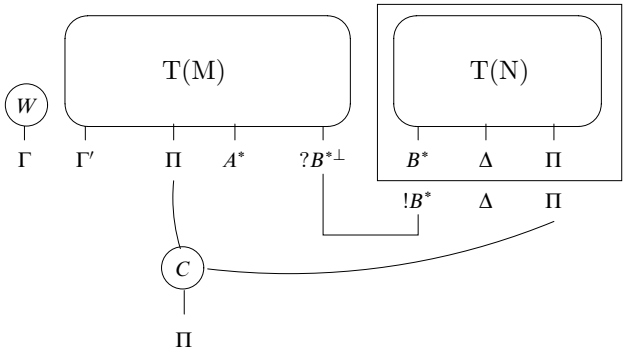
Starting from the first proof net, we eliminate the $w-b$ cut, then we apply the *wc* rule, and finally we obtain the desired proof net.

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

The typing environment can be split into four parts Γ, Γ', Δ and Π, where i is the length of Γ plus the length of Γ', j is the length of Δ and k ($k \leqslant i$) is the length of Γ. The typing judgment of $[i/N, j]\langle k\rangle M$ ends with

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

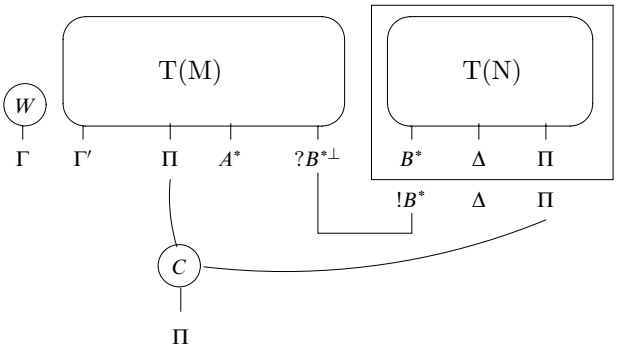
and its translation is the proof net



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

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

and its translation is the proof net



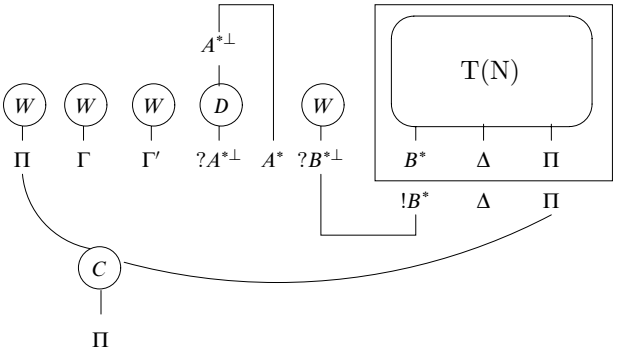
Note that the two nets are already the same. This is the first of the exception cases of the lemma.

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

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

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

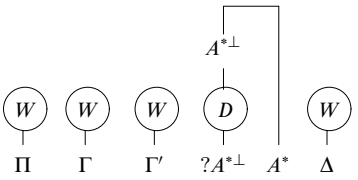
and its translation is the proof net



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

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

and its translation is the proof net



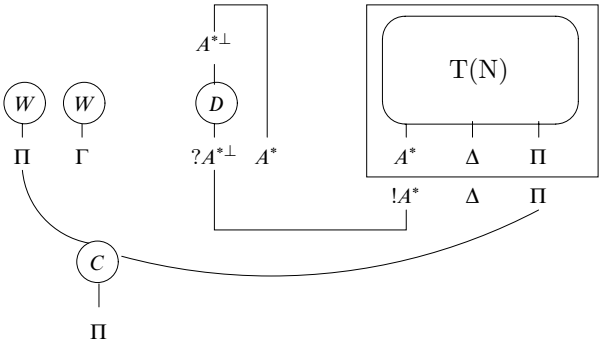
To reduce the first proof net into the second one, it is enough to eliminate the $w-b$ cut and to apply the *wc* rule.

— **Rule** n_2 **:** $[i/N, j]\underline{i} \longrightarrow \langle i\rangle N$

The typing environment can be split into three parts Γ, Δ and Π, where i is the length of Γ and j is the length of Δ. The typing judgment of $[i/N, j]\underline{i}$ ends with

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

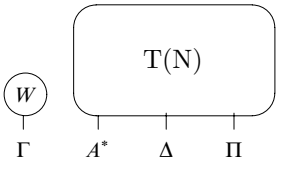
and its translation is the proof net



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



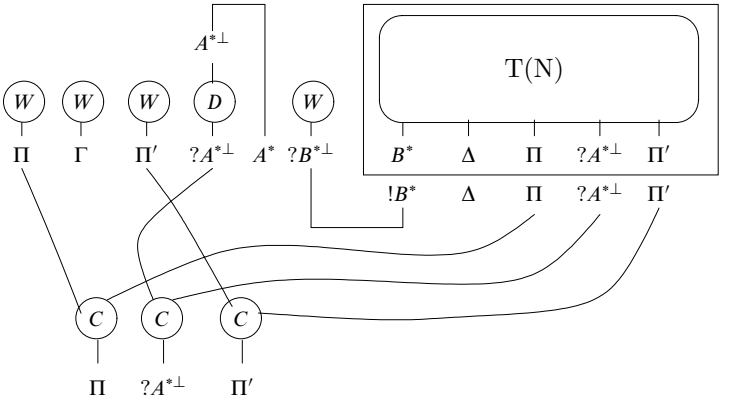
Starting from the first proof net, we eliminate the $d-b$ cut, then the $Ax-cut$ cut, and, finally, we apply the *wc* rule to obtain the desired proof net.

— **Rule** n_3 **:** $[i/N, j]\underline{k} \longrightarrow \underline{j+k-1}$ if $i < k$

The typing environment can be split into five parts Γ, Δ, Π, A and Π', where i is the length of Γ, j is the length of Δ and k ($k > i$) is the length of Γ plus the length of Π plus 1. The typing judgment of $[i/N, j]\underline{k}$ ends with

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

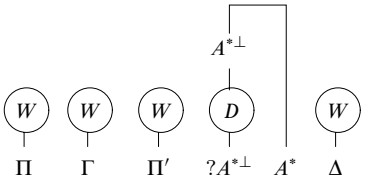
and its translation is the proof net



The typing judgment of $\underline{j + k - 1}$ must end with

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

and its translation is the proof net



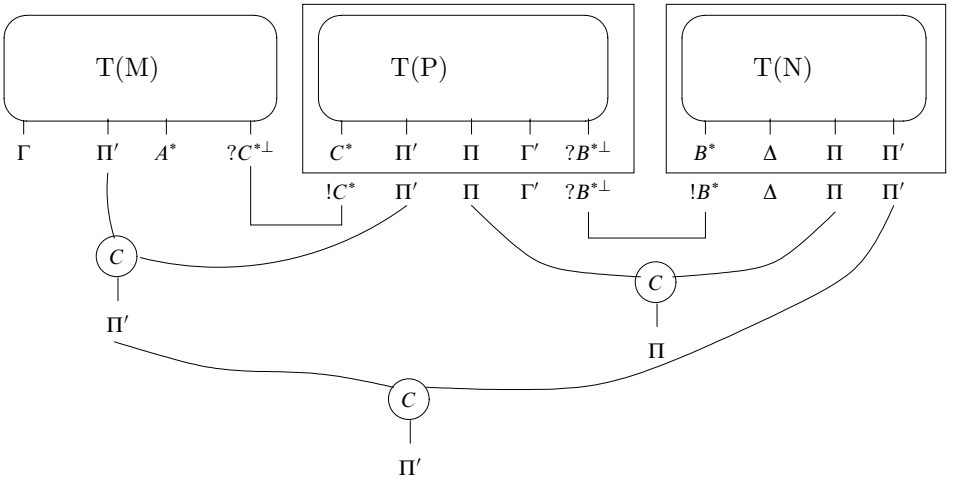
As for the n_1 rule, we eliminate the $w - b$ cut, then we apply the wc rule three times to achieve the desired result.

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

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

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

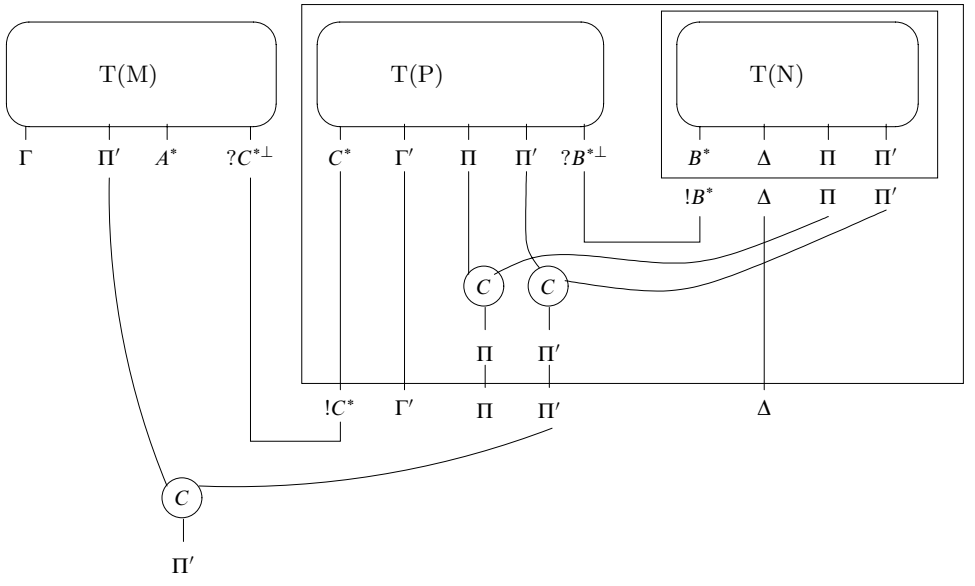
and its translation is the proof net



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

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

and its translation is the proof net



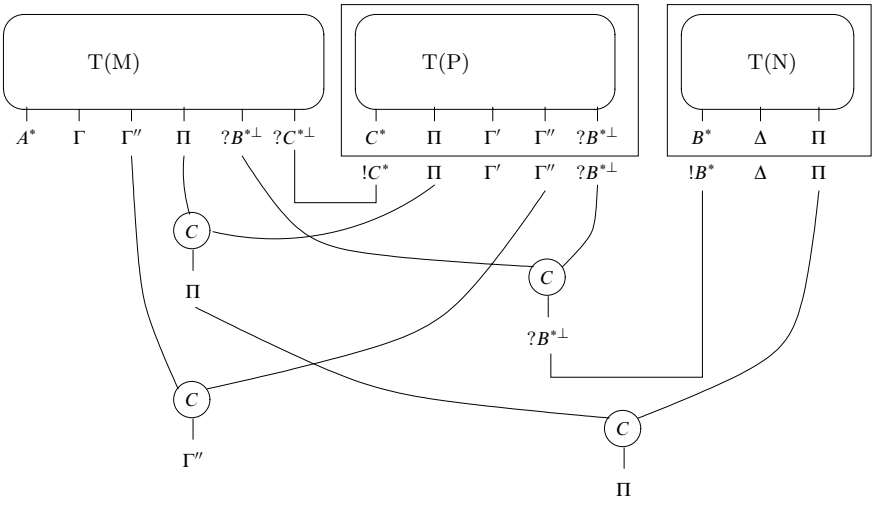
To reduce the first proof net into the second one, we must eliminate the $b-b$ cut, then apply the equivalence relations A and B.

— **Rule** c_2**:** $[i/N,j][k/P,l]M \longrightarrow [k/[i-k/N,j]P,l][i-l+1/N,j]M$ if $k+l \leqslant i$
The typing environment can be split into five parts Γ, Γ', Γ'', Δ and Π, where i is the length of Γ plus the length of Γ' plus the length of Γ'', j is the length of Δ, k $(k+l \leqslant i)$ is the length of Γ and l is the length of Γ'. The typing judgment of $[i/N,j][k/P,l]M$

ends with

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

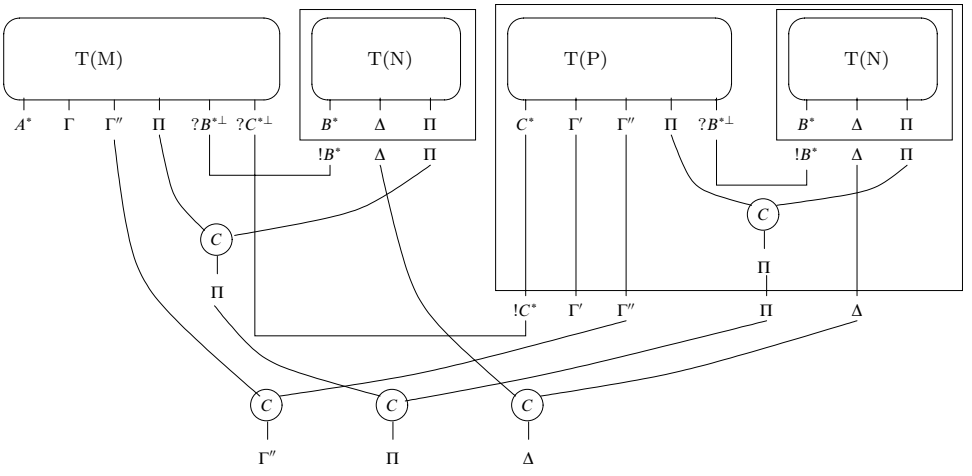
and its translation is the proof net



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

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

and its translation is the proof net



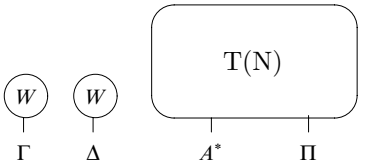
Starting from the first proof net, we eliminate the $c - b$ cut, then the $b - b$ cut, and, finally, we apply the equivalence rules A and B to obtain the desired proof net.

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

The typing environment can be split into three parts Γ, Δ and Π, where i is the length of Γ and j is the length of Δ. The typing judgment of $\langle i\rangle\langle j\rangle M$ ends with

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

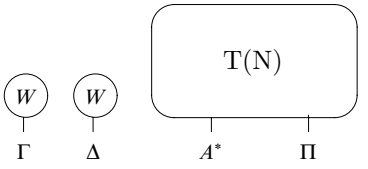
and its translation is the proof net



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

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

and its translation is the proof net



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 normalisation of* λ_{ws}

We are now able to show strong normalisation of λ_{ws}. To achieve this result, we use the following abstract theorem (see, for example, Ferreira *et al.* (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 normalising 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 normalising, $R_1 \cup R_2$ is also strongly normalising.

If we take $\mathcal{O}$ as the set of typed λ_{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 Theorem 4.3 and the fact that the system including the rules $\{e_2, d\}$ is strongly normalising (David and Guillaume 1999; 2001), we get the following theorem.

Theorem 4.4 (Strong normalisation of λ_{ws}**).** The typed λ_{ws}-calculus is strongly normalising.

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

Fig. 4. Reduction rules of the λ_{ws}-calculus with named variables.

5. A named version of the λ_{ws}-calculus

In this section we present a version of typed λ_{ws} with named variables *à la Church*[†]. 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 Γ and Δ denote sets of variables:

$$
\begin{array}{lll}
M ::= & x & \textit{variable} \\
| & \lambda x : A.M & \textit{abstraction} \\
| & (MM) & \textit{application} \\
| & \Delta M & \textit{label} \\
| & M[x,M,\Gamma,\Delta] & \textit{substitution}
\end{array}
$$

Intuitively, the term ΔM means that the variables in Δ are not in M, and the term $M[x,N,\Gamma,\Delta]$ means that the variables in Γ do not appear in N (Γ is a subset of the type environment of M, not containing x) and the variables Δ do not appear in M (Δ 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 α-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 (observe that rule b_1 is a particular case of rule b_2 with $\Delta = \emptyset$). Note that these rules may be applied to

[†] It is, of course, possible to give a presentation *à la Curry* without type annotations on the abstracted variables.

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 α-conversion, we can always find a term α-equivalent to an abstraction $\lambda y : A.M$ such that the condition imposed on the rule is true and thus no variable capture arises.

Note 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. Note also 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 abuse the notation slightly 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).

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

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

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

Fig. 5. Typing rules for the λ_{ws}-calculus with named variables.

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

As expected, the λ_{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 $\pi_1, \ldots \pi_n$, 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

$$\frac{\dfrac{\dfrac{\pi_1}{\Psi, x : A \vdash M : C}}{\Psi \vdash \lambda x : A.M : A \to C}\ (Lamb) \qquad \dfrac{\pi_2}{\Psi \vdash N : A}}{\Psi \vdash (\lambda x : A.M)N : C}\ (App)$$

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

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

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

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

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

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

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

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

where Ψ is actually split into Γ, Δ and Π. Since y is bound in $\lambda y : A.M$, we can suppose that Δ does not contain y, so we can construct the derivation

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

— 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

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

where Ψ is decomposed into Γ, Δ and Π. Now we can easily construct a derivation π'

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

and a derivation π''

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

from which we finally obtain

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

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

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

where Ψ decomposes into Γ,Δ and Π. We know also, by the definition of rule $e1$, that $x \in \Lambda$, so Λ is actually composed of x plus some other variables coming in part from Γ and in part from Π, that is, $\Lambda = (x : B, \Gamma'', \Pi'')$ with $\Gamma = \Gamma', \Gamma''$, $\Pi = \Pi', \Pi''$ and such that the set difference $\Gamma \setminus \Lambda$ is Γ' and $\Pi \setminus \Lambda$ is Π'.
Since $\Pi' \subseteq \Pi$, it is evident that $\Delta \cap \Pi' = \emptyset$, and since $\Gamma' \subseteq \Gamma$, we have that $\Delta \cap \Gamma' = \emptyset$ 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

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

— 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 $x \notin \Lambda$.
Because of the structure of R, the typing derivation necessarily has the form

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

where Ψ decomposes into Γ, Δ and Π. Furthermore, by the definition of rule e_2, we have that $x \notin \Lambda$, so Λ can be written as Γ'', Π'', 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

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

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 (Γ, Δ, Π) are well-formed environments. For (Sub), we need $(\Gamma', x : B) \cap (\Delta, \Pi'') = \emptyset$. First of all, note that $(\Gamma, x : B) \cap \Delta = \emptyset$ holds because of the side condition of the (Sub) rule in the typing derivation for R. Second, $\Gamma' \cap \Pi'' = \emptyset$ holds because (Γ, Δ, Π) is a well-formed environment. Finally, $x : B \notin \Pi''$ 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

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

where Ψ decomposes into Γ, Δ and Π. The required derivation is then simply

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

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

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

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

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

— 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

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

where Ψ decomposes into Γ, Δ and Π. Now, π' is a derivation that necessarily ends with an application of the (Sub) rule, so 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 Π into three pairwise disjoint components (each of which may be empty): Π_Λ, which is part of Λ, Π_Φ, which is part of Φ, and a Π' which is common to the typing environments used to type M and P. Similarly, Γ decomposes into pairwise disjoint Γ_Λ, Γ_Φ and Γ'. 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 $x \in \Phi$.
To sum all this up, the derivation π' must be of the form

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

Now, from π_1 and π_2 we can first build the derivation π'', where we use the fact that, since Π and Γ are disjoint, and $\Lambda = \Gamma_\Lambda \cup \Pi_\Lambda$, we know that Π_Λ can be written as $\Lambda \setminus \Gamma$

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

Note that the side condition holds because, on the one hand, Π and Γ are disjoint, on the other hand, $x \notin \Pi$ since $\Gamma, x : B, \Pi$ is well-formed, and finally because we know from the side conditions of the typing derivation for R that Δ is disjoint from $\Gamma, x : B$. Now, since y is bound in $M[y, P, \Lambda, \Phi]$, we can also suppose that Δ does not contain y, so we can build the following derivation π'''

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

Where the side condition reads

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

This holds because, on the one hand, Γ_Λ is disjoint from Γ_Φ by definition, disjoint from Π_Φ because $\Gamma \cap \Pi = \emptyset$ and disjoint from Δ because $\Gamma \cap \Delta = \emptyset$ by the side conditions of the typing derivation for R; on the other hand, we have already seen that we can assume y is not in Δ and we know from the side conditions in the typing derivation of R that y is not in Φ, 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 π''' is the required 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

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

where Ψ decomposes as Γ, Δ, Π. As for the rule c_1, the environment Γ decomposes as $\Gamma_\Lambda, \Gamma_\Phi, \Gamma'$; the environment Π 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 π' must be of the form

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

We know that Π_Λ can be written as $\Lambda \setminus \Gamma$ and Π_Φ can be written as $\Phi \setminus \Gamma$. Now, we can first build the derivation π''

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

As in c_1, the side condition holds.
Since y is bound in $M[y, P, \Lambda, \Phi]$, we can suppose by α-conversion that Δ does not contain y, and we also obtain the derivation π'''

$$\dfrac{\dfrac{\pi_1}{\Delta, \Pi \vdash N : B} \quad \dfrac{\pi_3}{\Gamma', \Gamma_\Lambda, y : A, x : B, \Pi', \Pi_\Lambda \vdash M : C} \quad ((\Gamma \setminus \Phi) \cup y : A \cup x : B) \cap (\Delta \cup (\Phi \setminus \Gamma)) = \emptyset}{\Gamma', \Gamma_\Lambda, y : A, \Delta, \Pi \vdash M[x, N, (\Gamma \setminus \Phi) \cup y, \Delta \cup (\Phi \setminus \Gamma)] : C} \; (Sub)$$

The side conditions also hold here: on the one hand Γ and Δ are disjoint, and $(\Gamma \setminus \Phi)$ and $(\Phi \setminus \Gamma)$ are trivially disjoint; on the other hand y is not in Δ, by α-conversion, and x is not in Δ by hypothesis on the side condition of the type derivation of R; finally, y is not in Φ by the side condition of derivation π', and x is not in Φ by hypothesis of the rule c_2.
Now, since $(\Lambda, y : A) \cap \Phi = \emptyset$, by the side condition of the (Sub) rule in the derivation π', we can finally build

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

which is the required derivation.

— Rule d: $R = \Gamma \Delta M$ rewrites to $(\Gamma \cup \Delta)M = R'$ The derivation of R must be of the form

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Pi \vdash M : C} \quad \Delta \cap \Pi = \emptyset}{\Delta, \Pi \vdash \Delta M : C} \; (Weak) \quad \Gamma \cap (\Delta, \Pi) = \emptyset}{\Gamma, \Delta, \Pi \vdash \Gamma \Delta M : C} \; (Weak)$$

with Ψ that decomposes into Γ, Δ and Π.
It is easy to rebuild the required derivation

$$\dfrac{\dfrac{\pi_1}{\Pi \vdash M : C} \quad (\Gamma \cup \Delta) \cap \Pi = \emptyset}{\Gamma, \Delta, \Pi \vdash (\Gamma \cup \Delta)M : C} \; (Weak)$$

□

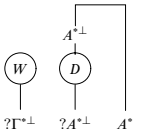
6. Strong normalisation of the λ_{ws} calculus with names

We now give the translation of the terms of λ_{ws} with names into proof nets in PN, and the proof of strong normalisation of λ_{ws}.

In order to translate a term of λ_{ws} 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.

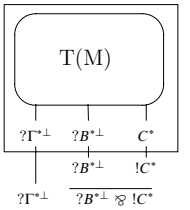
— If the term is a variable and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

$$\frac{}{\Gamma, x : A \vdash x : A}\ Ax$$



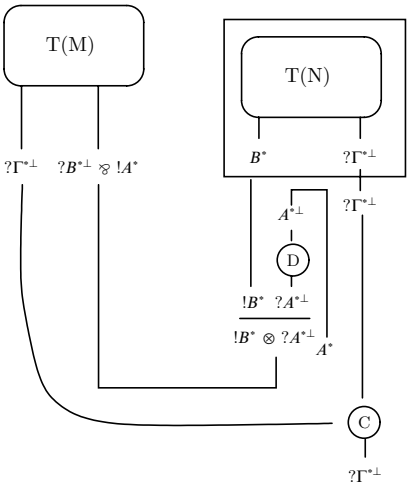
— If the term is a λ-abstraction and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



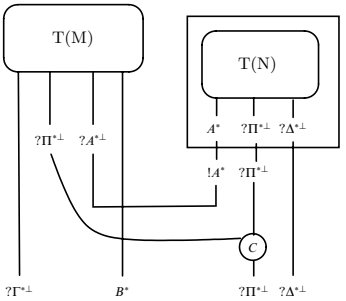
— If the term is an application and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



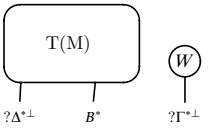
— If the term is a substitution and its typing judgement ends with the rule written below on the left, then its translation is the proof net on the right

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



— 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

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



We can clearly verify that the translation is identical to that given for λ_{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 λ_{ws}-calculus with names by the reduction R_E is identical to that given in Section 4.2 for the λ_{ws}-calculus with indices. We just note that rule n_3 makes no sense in the formalism with names so that the proof has one less case. We will just state the result without repeating a boring verification.

Lemma 6.1 (Simulation of λ_{ws} with names). If t λ_{ws}-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 with the following theorem.

Theorem 6.2 (Strong normalisation of λ_{ws} with names). The typed λ_{ws}-calculus with names is strongly normalising.

7. Conclusion and future work

In this paper we have enriched the standard notion of cut elimination in proof nets in order to obtain a system R_E that is flexible enough to provide an interpretation of λ-calculi with explicit substitutions and is much simpler than the one proposed in Di Cosmo and Kesner (1997). We have proved that this system is strongly normalising.

We have then proposed a natural translation from λ_{ws} into proof nets that immediately provides strong normalisation of the typed version of λ_{ws}, a calculus featuring full composition of substitutions. The proof is extremely simple with respect to the proof of PSN of λ_{ws} given in David and Guillaume (1999; 2001) and shows in some sense that λ_{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 λ_{ws} with named variables, instead of de Bruijn indices. This typed named version of λ_{ws} gives a better understanding of the mechanisms of the calculus. In particular, names allow us to understand the manipulation of explicit weakenings in λ_{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 for interpreting β reduction, or whether we need to add some more equivalences. 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, such as, for example,

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

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

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 *et al.* (2000) for their contributions to improve the presentation of this document.

References

Abadi, M., Cardelli, L., Curien, P. L. and Lévy, J.-J. (1991) Explicit substitutions. *Journal of Functional Programming* **4** (1) 375–416.

Abramsky, S. and Jagadeesan, R. (1992) New foundations for the geometry of interaction. *Logic in Computer Science (LICS)* 211–222.

Bloo, R. and Rose, K. (1995) Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. *Computing Science in the Netherlands*, Netherlands Computer Science Research Foundation 62–72.

Danos, V. (1990) *La logique linéaire appliquée à l'étude de divers processus de normalisation (et principalement du λ-calcul)*, Ph. D. thesis, Université de Paris VII.

Danos, V., Joinet, J.-B. and Schellinx, H. (1995) Sequent calculi for second order logic. In: Girard, J.-Y., Lafont, Y. and Regnier, L. (eds.) *Advances in Linear Logic*, London Mathematical Society Lecture Notes, Cambridge University Press.

Danos, V. and Regnier, L. (1995) Proof-nets and the Hilbert space. In: Girard, J.-Y., Lafont, Y. and Regnier, L. (eds.) *Advances in Linear Logic*, London Mathematical Society Lecture Notes, Cambridge University Press 307–328.

David, R. and Guillaume, B. (1999) The λ_l-calculus. In: Kesner, D. (ed.) *Proceedings of the 2nd Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs* 2–13.

David, R. and Guillaume, B. (2001) A λ-calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science* **11**.

Di Cosmo, R. and Guerrini, S. (1999) Strong normalization of proof nets modulo structural congruences. In: Narendran, P. and Rusinowitch, M. (eds.) Tenth International Conference on Rewriting Techniques and Applications. *Springer-Verlag Lecture Notes in Computer Science* **1631** 75–89.

Di Cosmo, R. and Kesner, D. (1997) Strong normalization of explicit substitutions via cut elimination in proof nets. In: *Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press 35–46.

Di Cosmo, R., Kesner, D. and Polonovski, E. (2000) Proof nets and explicit substitutions. In: Tiuryn, J. (ed.) Foundations of Software Science and Computation Structures (FOSSACS). *Springer-Verlag Lecture Notes in Computer Science* **1784** 63–81.

Ferreira, M. C., Kesner, D. and Puel, L. (1999) Lambda-calculi with explicit substitutions preserving strong normalization. *Applicable Algebra in Engineering Communication and Computing* **9** (4) 333–371.

Girard, J.-Y. (1987) Linear logic. *Theoretical Computer Science* **50** (1) 1–101.

Girard, J.-Y. (1989) Geometry of interaction I: interpretation of system F. In: Ferro, R., Bonotto, C., Valentini, S. and Zanardo, A. (eds.) *Logic colloquium 1988*, North-Holland 221–260.

Gonthier, G., Abadi, M. and Lévy, J.-J. (1992) The geometry of optimal lambda reduction. In: *Proceedings of POPL, Albuquerque, New Mexico*, Association for Computing Machinery 15–26.

Guillaume, B. (1999) *Un calcul de substitution avec étiquettes*, Ph. D. thesis, Université de Savoie.

Lamping, J. (1990) An algorithm for optimal lambda calculus reduction. In: *Procedings of POPL, San Francisco, California*, Association for Computing Machinery 16–30.

Melliès, P.-A. (1995) Typed λ-calculi with explicit substitutions may not terminate. In: Dezani-Ciancaglini, M. and Plotkin, G. (eds.) Proceedings of the 2nd International Conference of Typed Lambda Calculus and Applications. *Springer-Verlag Lecture Notes in Computer Science* **902**.

Rose, K. (1992) Explicit cyclic substitutions. In: Rusinowitch, M. and Rémy, J.-L. (eds.) Proceedings of the Third International Workshop on Conditional Term Rewriting Systems (CTRS). *Springer-Verlag Lecture Notes in Computer Science* **656** 36–50.