Strong Normalization of Explicit Substitutions via Cut Elimination in Proof Nets

(Extended Abstract)

Roberto Di Cosmo
DMI-LIENS (CNRS URA 1347)
Ecole Normale Supérieure
45, Rue d'Ulm
75230 Paris Cedex, France
Email:dicosmo@ens.fr

Delia Kesner
LRI (CNRS URA 410)
Bât 490
Université de Paris-Sud
91405 Orsay Cedex, France
Email:kesner@lri.fr

Abstract

In this paper, we show the correspondence existing between normalization in calculi with explicit substitution and cut elimination in sequent calculus for Linear Logic, via Proof Nets. This correspondence allows us to prove that a typed version of the $\lambda\mathtt{x}$-calculus [34, 5] is strongly normalizing, as well as of all the calculi that can be translated to it keeping normalization properties such as λ_v [27], λ_s [22], λ_d [24], and λ_f [14]. In order to achieve this result, we introduce a new notion of reduction in Proof Nets: this extended reduction is still confluent and strongly normalizing, and is of interest of its own, as it corresponds to more identifications of proofs in Linear Logic that differ by inessential details. These results show that calculi with explicit substitutions are really an intermediate formalism between lambda calculus and proof nets, and suggest a completely new way to look at the problems still open in the field of explicit substitutions.

1 Introduction

This paper is about 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 one side, explicit substitutions provide an intermediate formalism that - by decomposing the β rule into more atomic steps, more similar to what happens in environment machines - allows a better understanding of the execution models. On the other side, Linear Logic decomposes the intuitionistic logical connectives, like the arrow, into more atomic, resource-aware connectives, like the linear arrow and the explicit erasure and duplication operators given by the exponentials: this decomposition is reflected in Proof Nets, which are the natural deduction of Linear Logic, and provide a more refined computational model that the one given by the λ-calculus, which is the natural deduction of Intuitionistic Logic. Using different translations of the λ-calculus into Proof Nets, new abstract machines have been proposed, exploiting the Geometry of Interaction and the Dynamic Algebras [17, 2, 9], culminating in the recent works on optimal reduction [18, 26].

In this paper, we study the relationship between a calculus with explicit substitutions suggested in [29, 30] to study leftmost derivations in the λ-calculus and deeply studied, independently, in [34, 5] as the $\lambda\mathtt{x}$-calculus. We define a typed version of $\lambda\mathtt{x}$ and we show how to translate it into Proof Nets and how to establish, using this translation, a simulation of the reduction rules for explicit substitutions via cut elimination in Proof Nets. As an immediate consequence of this simulation, we prove that a simply typed version of $\lambda\mathtt{x}$ is strongly normalizing, as well as of all the typed calculi that can be translated to it keeping the normalization properties, such as λ_v [27], λ_s [22], λ_d [24], and λ_f [14]. The proof technique developed in the paper to simulate explicit substitution via proof nets gives a clear interpolation between typed λ-calculus and linear logic since calculi with explicit substitutions are already known to simulate β-reduction. This same remark holds for polymorphic λ-calculus as the technique in this paper can be easily extended to a polymorphic version of $\lambda\mathtt{x}$. Another important fallout is an extended notion of reduction on Proof Nets, which is needed in the simulation, and is of interest on its own because it is still confluent and strongly normalizing, but it identifies more proofs which differ by uninteresting details than the original notion of reduction. Last, but not least, by providing a finer analysis of reduction in explicit substitutions, the translation in Proof Nets provides a tool of choice to study the problems still open in the field of explicit substitutions, like the quest for a well-behaved notion

of composition and the like.
In the rest of this introduction we will recall the basic notions about explicit substitutions, point out some relations with cut elimination, recall the basic concepts from Linear Logic and Proof Nets, and survey related works.

Explicit substitutions

In the λ-calculus, the evaluation process is modeled by β-reduction and the replacement of formal parameters by actual arguments is modeled by *substitution*. In classical λ-calculus, this substitution is a meta-level operation described by operators that are external to the language, whereas in λ-calculi with explicit substitutions the substitution is internalized and handled by symbols and reduction rules belonging to the syntax of the calculus. However the two formalisms are still very close: if $M\{x/N\}$ denotes implicit substitution, and one defines β-reduction as

$$(\lambda x.M)N \longrightarrow_\beta \; M\{x/N\}$$

and $M\{x/N\}$ is defined by induction on M as follows:

$$\begin{array}{lcl}
x\{x/N\} & = & N \\
y\{x/N\} & = & y \\
(M_1\ M_2)\{x/N\} & = & (M_1\{x/N\}\ M_2\{x/N\}) \\
(\lambda y : \sigma.M)\{x/N\} & = & \lambda y : A.(M\{x/N\}), \\
 & & \text{if } x \neq y \text{ and } y \notin FV(M)
\end{array}$$

then, the simplest way to specify a λ-calculus with explicit substitution is to encode explicitly the previous definition, yielding the calculus $\lambda\mathtt{x}$ shown in table 1. This form of explicit substitutions corresponds to the minimal behavior that can be found in most of the well-known calculi in the literature, sometimes in de Bruijn notation [1, 19, 27, 22, 24, 14] or level notation [28], some other times in named variables notation as shown above [29, 30, 34, 5]. We think that the named variable presentation makes some essential properties of explicit substitutions more apparent, by abstracting out the details of renaming and updating, and this is why we choose here to work with this formalism of substitutions. But all the results of this paper immediately carry over to many other calculi such as λ_υ [27], λ_s [22], λ_d [24], and λ_f [14] because they can all that can be translated to $\lambda\mathtt{x}$ [34, 5, 4] keeping the normalization properties.
In these last years there has been a growing interest in λ-calculi with explicit substitutions. The pioneer, λ_σ, was introduced in [1] as a bridge between the classical λ-calculus and concrete implementations of functional programming languages such as CAML [38], Hope [6, 15], SML [32], Miranda [36], Haskell [21]. The calculus λ_σ, which is inspired by de Bruijn notation [11, 12], is also similar to, and inspired by Categorical Combinatory Logic [7]. We find in the λ_σ-calculus the main features which characterize a calculus of explicit substitutions: substitutions are incorporated into the language and manipulated explicitly, objects are divided in two sorts - terms themselves and substitutions - and β-reduction is simulated in two stages, first by the application of one rule, called $Beta$, which activates the calculus of substitutions, then by propagation of the substitution until variables are reached. Substitutions can also be combined by a composition operator allowing many interactions between them. Strong normalization of the calculus of explicit substitutions alone, called the σ-calculus, is proved in [8], but the σ-calculus together with the $Beta$ rule does not preserve β-strong normalization as shown by Mellies [31], who exhibits a well-typed term which is not λ_σ-strongly normalizing. The λ_σ-calculus is confluent on closed terms, and it remains confluent when meta-variables for terms are added to the syntax (we say in this case that λ_σ is confluent on semi-open terms), but is no longer confluent when variables for substitutions are considered (confluence fails for open terms). The result of Mellies has revived the interest in explicit substitutions since after his counterexample there is now a clear “challenge” to find a calculus having all the good properties: correct implementation of one-step β reduction, preservation of β-strong normalization, a typed version being strongly normalizing, confluence on semi-closed terms (at least), and full composition.
There are already several propositions that give partial answers to this challenge: the λ_υ-calculus, proposed in [27], preserves β-strong normalization (see [3] for details) but is not confluent on non-closed terms and has no form of composition. The λ_s-calculus [22] has the same properties as λ_υ, but admits a conservative extension which is confluent on open terms. However, preservation of β-strong normalization of this extension is only conjectured. The λ_d-calculus [24], which also preserves β-strong normalization [14], adds the possibility of weak composition, or concatenation, but is only confluent on closed terms. Finally, λ_ς [33] is confluent on open terms and preserves β-strong normalization but it does not simulate a one-step β-reduction. It turns out that all the known calculi preserving β-strong normalization and simulating one-step β reduction can be translated into $\lambda\mathtt{x}$ by keeping normalization properties (see [35, 14] for details).
We believe that a search for the optimal calculus of substitution must necessarily pass through a clear understanding of the termination properties of the *typed* version of explicit substitution: Mellies' counterexample *is a simply typed term*, and it clearly points out that all the difficulties of composition already appear in a simply typed setting.
However, up to now the only available proof of strong normalization for a typed λ-calculus with explicit substitutions has been the one for λ_s in [23], which proceeds by mapping the terms in λ_s back to the ones in the simply typed λ calculus. While this is a perfectly viable technique to obtain the normalization result, it says nothing about the fine

$$
\begin{array}{llll}
(B) & (\lambda x:\sigma.M)N & \longrightarrow & M[x/N] \\
(Var1) & x[x/N] & \longrightarrow & N \\
(Var2) & x[y/N] & \longrightarrow & x, \text{ if } x \neq y \\
(Lambda) & (\lambda x:A.M)[y/N] & \longrightarrow & \lambda x:A.(M[y/N]), \text{ if } x \neq y \text{ and } x \notin FV(N) \\
(App) & (M_1\ M_2)[x/N] & \longrightarrow & (M_1[x/N]\ M_2[x/N])
\end{array}
$$

Table 1. Reduction rules for the $\lambda\mathtt{x}$-calculus

structure of the reduction in λ_s, since the calculus used in the proof is a calculus where all the details of the substitutions have disappeared: we need a structure richer than explicit substitution, and not poorer, if we hope to find in it a hint to better understand substitution calculi.

Explicit substitution and cut-elimination

We believe that such a richer structure should relate explicit substitution to a more atomic process, like cut -elimination. Indeed, the cut elimination process can, in some cases, be interpreted as the elimination of explicit substitutions in a given term. For example, let us consider the following sequent proof:

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash A} \qquad \Gamma, A \vdash A\ (axiom)}{\Gamma \vdash A}\ (cut)$$

If we want to eliminate the last cut rule used in this proof, it is sufficient to take the proof

$$\dfrac{\mathcal{D}}{\Gamma \vdash A}$$

which proves exactly the same sequent $\Gamma \vdash A$ but without the last cut rule. That is, in the cut elimination process, the first proof reduces to the second one. Now, let us interpret proofs by terms and propositions by types as suggested by the Curry-Howard correspondence. We then get

$$\dfrac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash x : A\ (axiom)}{\Gamma \vdash x[x/M] : A}\ (cut)$$

which suggests that the process of cut elimination consists in reducing the term $x[x/M]$ to the term M, exactly as in the $Var1$ rule of the calculus $\lambda\mathtt{x}$ written as

$$(Var1) \qquad x[x/M] \longrightarrow\ M$$

As another example, if one takes the proof

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash A} \qquad \Gamma, A \vdash B\ (axiom)}{\Gamma \vdash B}\ (cut)$$

then, the process of cut elimination gives

$$\Gamma \vdash B\ (axiom)$$

When interpreting propositions by types and proofs by terms, one obtains that

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash M : A} \qquad \Gamma, x : A \vdash y : B\ (axiom)}{\Gamma \vdash y[x/M] : B}\ (cut)$$

reduces to

$$\Gamma \vdash y : B\ (axiom)$$

yielding exactly the $Var2$ of the $\lambda\mathtt{x}$-calculus written as

$$(Var2) \qquad y[x/M] \longrightarrow\ y$$

These remarks seem to say that explicit substitutions behave precisely like cut elimination rules, but the situation is not so simple: the rule for the distribution of substitution through an application needs more work on a logical derivation than a simple cut-elimination, and indeed, this is precisely the reason why explicit substitution are an *intermediate* formalism between β-reduction and cut-elimination, and not just *the same*.

One can then do basically two things. Either define an explicit substitution calculus interpreting cut-elimination, in such a way to have a perfect Curry-Howard correspondence between them, as is done by Hugo Herbelin in [20]: there terms encode proofs, types encode propositions and reduction encodes cut-elimination in intuitionistic sequent calculus. However, this comes at a price, since, exactly as in [33], his calculus does not allow to simulate every possible one-step β-reduction, because substitutions cannot go through β-redexes.
Or we can look for a more refined logical system, whose proof structures can be adapted to simulate *exactly* the reduction of the explicit substitution calculus, without forcing us to loose anything of the original β reduction. This is precisely the object of this work, and the idea of our translation into Proof Nets: we take as a target Linear Logic, whose sequent calculus is more refined than, and can encode, intuitionistic sequent calculus, and we translate the

terms of our calculus into the natural deduction for Linear Logic, which are the Proof Nets.

Linear logic and proof nets

Proof Nets are the natural deduction of Linear Logic, are confluent and strongly normalizing and are able to simulate the traditional simply typed β-reduction [16, 9], so they provide the ideal setting to exploit the analogy between explicit substitution and cut elimination.
As is the case for classical or intuitionistic sequent calculus, a proof of a given sequent in a sequent calculus presentation of linear logic can contain a lot of details that are uninteresting. For example, consider how many uninterestingly different ways there are to form a proof of $\vdash (A_1 \parr A_2), \ldots, (A_{n-1} \parr A_n)$ starting from a derivation of $\vdash A_1, A_2, \ldots, A_n$. This unpleasant fact derives from the sequential nature of proofs in sequent calculus: if we want to apply a set S of n rules to different parts of a sequent, we cannot apply them in one step, even if they do not interfere with each other! We must *sequentialize* them, i.e. choose a linear order on S and apply the rules in n steps, according to this order. This phenomenon can lead us to $n!$ proofs that are intensionally different, but that we would like to consider as just one proof. The main idea behind Proof Nets (as for natural deduction) is to solve this problem by providing a sort of representative for an equivalence class of proofs in the sequent calculus that differ only by the order of application of some logical or structural rules. Cut elimination over proof nets is then a kind of normalization procedure over these equivalence classes.
Nevertheless, in the presence of exponentials, that are necessary to translate terms with multiple occurrences of a same variable, the traditional presentation of Proof Nets turns out to be inadequate: too many inessential details concerning the order of application of independent structural rules are still present and get into the way of a proper simulation, i.e. proof nets, in their traditional presentation, do not really identify all derivations that differ in inessential ways. One typical example is the order of application of the contraction rule: to contract n copies of an hypothesis $?A$, one can proceed in many different ways in a sequent calculus derivation, and this redundancy can still be found in the associated proof nets, because one finds a different tree of contraction nodes for each of the different derivations. When using proof nets to simulate reduction in the λ-calculus, this redundancy already gets in the way, so that it is necessary to consider an extended notion of reduction over proof nets.
But if we want to simulate the behavior of explicit substitutions we are really forced to consider contraction nodes as a sort of associative-commutative operator, and equip the classical reduction system associated to Proof Nets with some more additional reductions.

The idea is to improve the traditional representation of derivations in linear logic, by introducing n-ary contraction, written as

$$\frac{\vdash \Gamma, \overbrace{?A, \ldots, ?A}^{n\ times}}{\vdash \Gamma, ?A} \quad (n\text{-}contraction)$$

Then, Proof Nets will be equipped with n-ary contraction nodes which will represent all possible trees of binary contractions that can be used to contract n hypotheses. This intuitively corresponds to what is called *flattening* in the term rewriting community. As a consequence, it is also necessary to equip these n-ary contraction nodes with an adequate set of reductions associated to the set of proof nets: we need a cut-elimination rule for this n-ary contraction, and a set of extended rules which do not perform cut elimination, but correspond to permutations in the order of application of the structural rules. This improved notion of reduction is a result of interest on its own, as it gives a more satisfactory equivalence relation on sequent derivations, and it preserves the good properties of the system originally defined by Girard such as confluence and strong normalization.
Once this is done, we can finally present our translation into the extended proof nets and immediately obtain strong normalization for typed $\lambda\mathtt{x}$ from strong normalization of proof nets. An important property of the simulation is that each step in $\lambda\mathtt{x}$ is simulated by a constant number of steps in the net: this shows that the two systems are very close, unlike what happens when simulating the λ-calculus.
The main ideas and results that we present in this paper can be summarized by the following points:

- We prove that the *typed* $\lambda\mathtt{x}$-calculus, and all the typed λ-calculi with explicit substitutions that can be simulated by it, such as λ_v, λ_s, λ_d, λ_{dn} and λ_f are strongly normalizing.

- We point out the relation between strong normalization and cut elimination: in particular, the result of this paper shows that typed $\lambda\mathtt{x}$ is an intermediate calculus between typed λ-calculus and proof nets. This is a consequence of the fact that β-reduction can be simulated by any λ-calculus with explicit substitutions.

- We extend the standard notion of reduction relation associated to proof nets [16] in order to deal with n-ary contractions showing that the extended system is confluent and strongly normalizing. This is an important result, because the order of application of contraction rules is irrelevant, and gives at the same time the intuition that contraction can be treated as an AC operator.

The rest of the paper is organized as follows: section 2 introduces the type system for $\lambda\mathtt{x}$, section 3 introduces proof

nets with n-ary contractions with their cut-elimination rules, section 4 defines the extended reduction rules for proof nets, section 5 presents the fundamental results of confluence and strong normalization for the extended reduction, while section 6 gives the translation into extended proof nets that is used to obtain strong normalization of typed $\lambda\mathtt{x}$.
Throughout the paper we will use standard notations from rewriting, like $\longrightarrow^*_R$ for the reflexive transitive closure of a rewriting relation $\longrightarrow_R$ and the like.

2 Typing Explicit Substitutions

We introduce here a typing system for $\lambda\mathtt{x}$, suitable for the translation into Proof Nets. Let σ range over a set of base types and x over a set of variables. Then, the set of types can be defined be the following grammar: $A ::= \sigma \mid A \to A$, and the set of terms can be defined by $M ::= x \mid (M\ M) \mid \lambda x : A.M \mid M[x/M]$. A term is said to be *pure* if it has no explicit substitutions.
The set of *free variables* of a term M, denoted $FV(M)$, is defined in the usual way, adding the case for explicit substitution $FV(M[x/N]) = (FV(M) \setminus x) \cup FV(N)$.

The Typing Rules of the $\lambda\mathtt{x}$-calculus are the following:

$$\frac{}{\Gamma, x : A \vdash x : A}\ (axiom)$$

$$\frac{\Delta, \Gamma, x : A \vdash M : B \qquad \Gamma \vdash N : A}{\Delta, \Gamma \vdash M[x/N] : B}\ (subs)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B}\ (abs)$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B}\ (app)$$

In this system, a judgment $\Gamma \vdash M : A$ may have several type derivations, but this is not problematic as we will see below. We denote by $\Lambda_{\lambda\mathtt{x}}$ the set of well-typed terms of $\lambda\mathtt{x}$, by Λ the set of pure terms of $\lambda\mathtt{x}$ and by Λ_λ the set of well-typed pure terms of $\lambda\mathtt{x}$.

We identify terms modulo α-conversion and we consider the reduction rules given in section 1. The rules $Var1$, $Var2$, *App* and *Lambda* constitute the system $\mathtt{x}$, and $\mathtt{x}$ together the B-rule form the reduction system $\lambda\mathtt{x}$.

As expected, the calculus enjoys the subject reduction property:

Proposition 2.1 If $\Gamma \vdash M : A$ and $M \longrightarrow_{\lambda\mathtt{x}} M'$, then $\Gamma \vdash M' : A$.

As we will show in section 6, well typed $\lambda\mathtt{x}$-terms strongly normalize w.r.t. the system $\longrightarrow_{\lambda\mathtt{x}}$.

3 Proof nets

We recall here some classical notions from Linear Logic, namely the linear sequent calculus and Proof Nets, and some basic result concerning confluence and normalization.
Let $\mathcal{A}$ be a set of *atomic formulae*. We suppose that $\mathcal{A}$ is partitioned in two disjoint subsets representing *positive* and *negative* atoms respectively. For every $p \in \mathcal{A}$, we assume that there is $p' \in \mathcal{A}$, called the *linear negation of the atom* p. We will sometimes write $p^\perp$ instead of p'. The set of formulae is defined by the grammar:

$$F ::= p \mid F \otimes F \mid F \parr F \mid !F \mid ?F, \text{ where } p \in \mathcal{A}$$

The *linear negation* of a formula A, denoted $A^\perp$ is defined by the following De Morgan equations:

$$\begin{array}{lcl} p^\perp & = & p' \\ p'^\perp & = & p \\ (A \otimes B)^\perp & = & (A^\perp) \parr (B^\perp) \\ (A \parr B)^\perp & = & (A^\perp) \otimes B^\perp \\ (!A)^\perp & = & ?(A^\perp) \\ (?A)^\perp & = & !(A^\perp) \end{array}$$

If Γ is the sequence $A_1, \ldots A_m$, we denote by $?\Gamma$ the sequence $?A_1, \ldots, ?A_m$, by $!\Gamma$ the sequence $!A_1, \ldots, !A_m$, and by $\Gamma^\perp$ the sequence $A_1^\perp, \ldots, A_m^\perp$.

Definition 3.1 The sequent calculus for classical multiplicative-exponential linear logic with n-ary contraction is given by table 2 in the Appendix. We only consider sequents of the form $\vdash \Gamma$ as every sequence $\Delta \vdash \Lambda$ can be represented by $\vdash \Delta^\perp, \Lambda$.

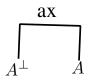
Definition 3.2 The sequent calculus for classical multiplicative-exponential linear logic with binary contraction is defined by the same system in table 2, but for the $(n\text{-}contraction)$ rule which is replaced by the following rule:

$$\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}(contraction)$$

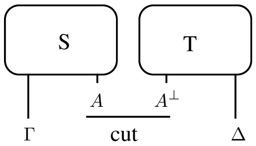
It is clear that $\vdash \Delta$ is derivable in the system of definition 3.2 if and only if $\vdash \Delta$ is derivable in the system of definition 3.1: if $\vdash \Gamma, ?A$ comes from $\vdash \Gamma, ?A, \ldots, ?A$ by an application of the $n\text{-}contraction$ rule, then $\vdash \Gamma, ?A$ is derivable from $\vdash \Gamma, ?A, \ldots, ?A$ by n applications of the $contraction$ rule, and vice-versa, the $contraction$ rule is a special case of the $n\text{-}contraction$ rule for $n = 2$.

Definition 3.3 The set of proof nets, denoted by PN, is the smallest set satisfying the following properties:

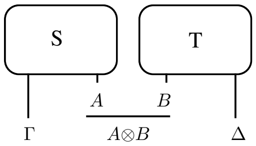
- An A-axiom link is a proof net with output formulae $A, A^{\perp}$, written as



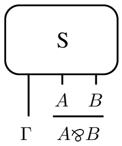
- If S is a proof net with output formulae A, Γ and T is a proof net with output formulae $A^{\perp}, \Delta$, then adding an A-cut link between A and $A^{\perp}$ yields a proof net with output formulae Γ, Δ written as



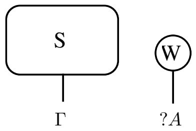
- If S is a proof net with output formulae A, Γ and T is a proof net with output formulae B, Δ, then adding an A-B-times link between A and B yields a proof net with output formulae $\Gamma, A \otimes B, \Delta$ written as



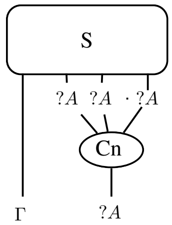
- If S is a proof net with output formulae Γ, A, B, then adding an A-B-par link between A and B yields a proof net with output formulae $\Gamma, A \parr B$ written as



- If S is a proof net with output formulae Γ, then adding an A-weakening link yields a proof net with output formulae $\Gamma, ?A$ written as

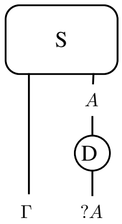


- If S is a proof net with output formulae $\Gamma, \underbrace{?A, \ldots, ?A}_{n\ times}$ ($n \geq 2$), then adding an A-n-contraction link yields a proof net with output formulae $\Gamma, ?A$ written as

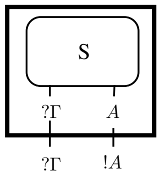


We say that $?A, \ldots, ?A$ are the *inputs* of the contraction node, and the conclusion formula $?A$ is the *output* of the contraction node.

- If S is a proof net with output formulae Γ, A, then adding an A-dereliction link yields a proof net with output formulae $\Gamma, ?A$ written as



- If S is a proof net with output formulae $?\Gamma, A$, then adding an A-$?\Gamma$-box link yields a proof net with output formulae $?\Gamma, !A$ written as



Definition 3.4 The cut-elimination procedure over PN is performed via the rewrite relation $\longrightarrow_{PN}$ defined similarly to [16] and shown in table 3 in the Appendix.

When contraction nodes are only binary as in definition 3.2, we denote by $\widehat{PN}$ the associated reduction relation.

Theorem 3.1 The reduction relation $\longrightarrow_{PN}$ is strongly normalizing.

Proof. The proof of strong normalization for $\longrightarrow_{\widehat{PN}}$ can be found in [16, 37]. It is then straightforward to obtain normalization of the system $\longrightarrow_{PN}$ presented here: n-ary contraction can be simulated by a tree of binary contractions and each cut-elimination on an n-ary contraction gives raise to $n-1$ cut-elimination steps in the translation.

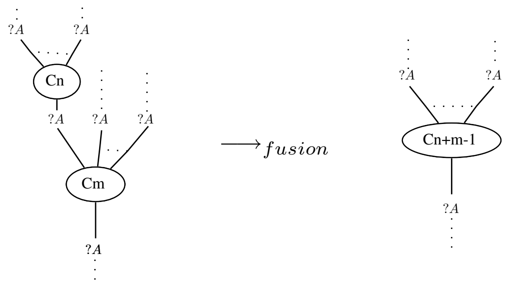
Since the system $\widehat{PN}$ is locally confluent, it is straightforward that:

Theorem 3.2 The reduction relation $\longrightarrow_{PN}$ is confluent.

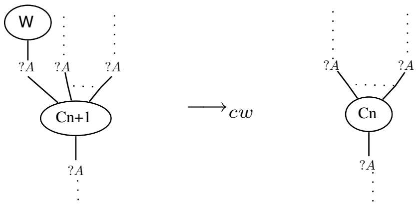
4 Extended reduction for proof nets

In this section, we introduce an extended notion of reduction that is suitable for our simulation, and we prove that it is confluent and strongly normalizing. This result is of interest in its own, because it allows in general to work on proof nets up to a set of transformations which capture more equivalent sequent derivations than the original proof nets do. In some sense, we are working modulo associativity and commutativity of contraction. Due to lack of space, we can only give here a sketch of the proofs, and refer the interested reader to the full paper [13] for details.

First of all, one should be able to fuse two n-ary contraction nodes together, as in

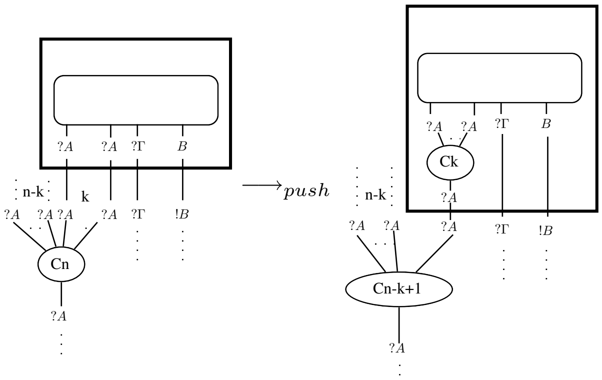


where $n, m \geq 2$. We also need a rule to simplify contractions with weakenings:



including the case for $n = 1$, where the result is just a connecting wire. This corresponds to the elimination of irrelevant "detours" in the derivation.

Finally, the order of contraction and box formation should also be irrelevant, and this leads to the final reduction rule:



that allows to push some contractions inside a box. Here $n \geq 2$ and $n \geq k \geq 2$, so in the case $n = k$, then the outer $n - k + 1$-contraction node is just a connecting wire.

The system $PN + fusion + cw + push$ is called PN_C. In the following, we will show that the system PN_C is confluent and strongly normalizing. Notice that the reductions we introduced above do not satisfy the requirements of interaction nets [25], so that we do not get confluence for free. Even though confluence is not necessary for our simulation, we will nevertheless establish it directly (via local confluence and strong normalization), as a result of interest on its own.

Lemma 4.1 (Local Confluence of PN_C) The reduction relation PN_C is locally confluent.

Proof. By inspecting all the possible cases. See [13] for details.

5 Strong Normalization of PN_C

We will prove in the following that the PN_C system is strongly normalizing. This will be done by showing that each of the rules $fusion$, cw and $push$ separately does not increase the longest reduction path of a net w.r.t. cut-elimination, and proving that the new rules are strongly normalizing themselves. This will allow us to obtain strong normalization by a simple argument using the lexicographic ordering of the longest reduction-path of cut-elimination and the measure used to show strong normalization of $fusion + cw + push$.

Definition 5.1 (R-reduction tree of a net) Given a finitely branching strongly normalizing reduction R on nets, and any net S, we define the tree $T_R(S)$ of all possible R-reductions of S to normal form. Due to Konig's lemma, this tree is finite. We denote by $\nu(T_R(S))$ the maximum length of a path in the tree $T_R(S)$ and by $size(T_R(S))$ the sum of the lengths of the paths in $T_R(S)$. A path in $T_R(S)$ is called *worst* if it has maximal length.

5.1 Properties of the new rules

We study now some properties of the $fusion$, cw and $push$ rules, which will be used latter to show that PN_C is a strongly normalizing system.

Lemma 5.1 (fusion, cw and $\nu(T_{PN}(S))$) For any net S such that $S \longrightarrow_{fusion,cw} S'$, $\nu(T_{PN}(S')) \leq \nu(T_{PN}(S))$.

Proof. It is sufficient to show that to every reduction path in $T_{PN}(S')$ we can associate injectively a reduction path in $T_{PN}(S)$ having at least the same length. The proof can be done by induction on the length of the reduction path in $T_{PN}(S')$. See [13] for full details.

Unlike what happens with the fusion and cw rules, if $S \longrightarrow_{push} S'$, then it is not always possible to build directly from *any* reduction path in S' a longer (or equal) one in S: we need to take a longest reduction path in S', that we will call a *worst* path, and then we can build from it a longer or equal path in S (which is not necessarily a worst path for S). This will suffice for the strong normalization argument.

Lemma 5.2 (push does not increase $\nu(T_{PN}(S))$) For any net S such that $S \longrightarrow_{push} S'$, $\nu(T_{PN}(S')) \leq \nu(T_{PN}(S))$.

Proof. The proof technique is quite similar to the one used for the fusion and cw rules, but with the additional hypothesis that the path in S' is a worst path. One maintains an association between the nets S' and S and builds a reduction path in S by induction on the length of the (worst) reduction path in S', using various results on worst paths. All the details are given in the full paper.

Lemma 5.3 (fusion, cw and push alone are SN) The system $fusion + cw + push$ is strongly normalizing.

Proof. It suffices to define a *weight* for a proof net which is strictly decreased by each of the rules. We take $weight(S) = \Sigma\{n.3^j \mid Cn \text{ is a contraction node in } S \text{ having height } j\}$, where the height of a contraction node C_n is just 1 plus the maximum number of imbricated boxes that are contained in the smallest box including C_n (or in the net S if C_n is not included in any box). All details of the proof, including a formal definition of the height of a node, are given in the full paper.

We can finally establish the main results of this section.

Theorem 5.4 (Strong normalization of PN_C) The system PN_C is strongly normalizing.

Proof. The system $fusion + cw + push$ is strongly normalizing by lemma 5.3, and we showed in lemmas 5.1 and 5.2 that each of the three reductions $fusion$, cw and $push$ does not increase the longest PN-reduction path in a net. Since the reduction relation PN is strongly normalizing by theorem 3.1, the measure obtained by the lexicographic ordering on $\langle \nu(T_{PN}(S)), weight(S) \rangle$ strictly decreases when $S \longrightarrow_{PN_C} S'$. As a consequence, PN_C is strongly normalizing.

From the theorem above and lemma 4.1, we obtain then, by Newman's Lemma,

Theorem 5.5 (Confluence of PN_C) The system PN_C is confluent.

As a final remark, it is interesting to notice that the modified proof nets used in [10] can be seen as a way to reduce nets that are in normal form w.r.t. our rules $fusion$, cw and $push$ (this last taken in the reverse direction). The notion of reduction that one gets that way is very well suitable for simulating β-reduction, but too coarse to handle explicit substitutions.

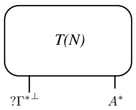
6 From $\lambda\mathtt{x}$-terms to proof nets

In this section we present our translation of the $\lambda\mathtt{x}$-calculus into proof nets. To do so, we will translate terms and types in a way similar to [37], adding the case for substitution, and applying the fusion rule wherever possible. Then we will show how this translation yields a simulation over the extended notion of reduction. Due to lack of space, we can only give here a sketch of the proofs, and refer the interested reader to the full paper [13] for details.

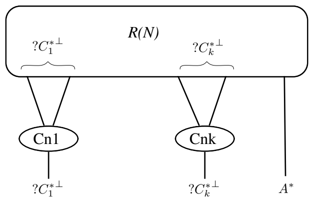
Let's start by the usual translation of types into linear formulae:

$$\begin{array}{lcll} A^* & = & A & \text{if } A \text{ is atomic} \\ (A \to B)^* & = & ?((A^*)^\perp) \parr B^* & \end{array}$$

Now we can give our translation of terms, which is defined by induction on the derivation of a typing judgment: to each derivation of a typing judgment $\Gamma \vdash N : A$ we associate a proof net $T(N)$ with conclusions $?\Gamma^{*\perp}$ and A^*

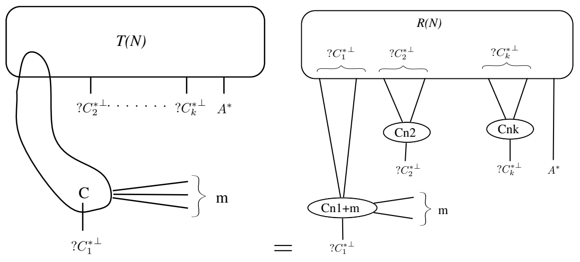


The key idea is to merge together - using the fusion rule - all adjacent contraction nodes that are kept separated in the traditional translations. One could define the translation as the fusion normal form of the translation given in [37], but it is more convenient to give a direct inductive definition that builds nets without adjacent contraction nodes. We will denote by $R(N)$ the net obtained from $T(N)$ by removing the contraction nodes over the conclusions of $T(N)$ so that $T(N)$ can be seen as:



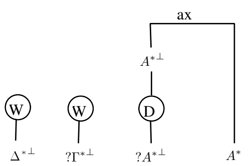
To improve the readability of the drawings that will follow, we will often note the result of merging a contraction node(s) over a (set of) conclusion(s) in a net $T(N)$ with a contraction node(s) outside it by drawing the resulting contraction node(s) as an oval partially overlapping the original network

$T(N)$, as in the following example, where this is done just over the conclusion ${C_1}^*$:



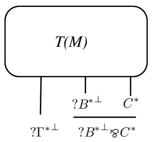
Definition 6.1 (Translation) A family of translations are defined for every typing judgment $\Gamma \vdash M : A$, as we define *one* translation for *each* type derivation of $\Gamma \vdash M : A$. A translation associates to each derivation a net *in fusion normal form* by induction on the structure of the derivation of the judgment as follows:

- If $\Gamma, x : A \vdash x : A$ is an axiom, with $\Gamma = C_1, \cdots, C_n$, then its translation is:



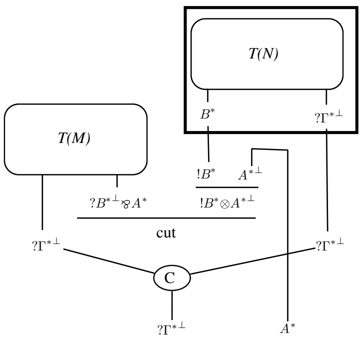
  This net is clearly in fusion normal form .

- If $\Gamma \vdash \lambda x : B.M : B \to C$, with $\Gamma, x : B \vdash M : C$, then its translation is

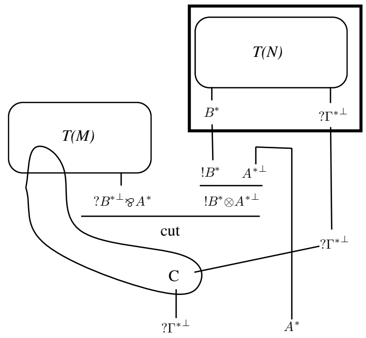


  Since $T(M)$ is in fusion normal form , and we do not add any contraction node, this net is also in fusion normal form .

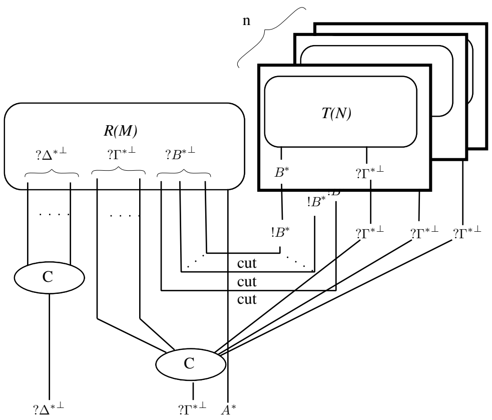
- If $\Gamma \vdash (M\ \ N) : A$, with $\Gamma \vdash M : B \to A$ and $\Gamma \vdash N : B$, then its translation is obtained from:



  by merging the contraction node introduced above with all adjacent contraction nodes in the net. As $T(N)$ is isolated inside a box, the only adjacent contraction node (and only one, since $T(M)$ is in fusion normal form) in this case can be in the translation $T(M)$ of M, so the translation can be drawn as:



- If $\Gamma, \Delta \vdash M[x/N] : A$, with $\Gamma, \Delta, x : B \vdash M : A$ and $\Gamma \vdash N : B$, then the translation is obtained from $T(M)$ by removing (only) the n-contraction node corresponding to the output $?({}^\perp B^*)$ to create the n cut-links with $T(N)$, and linking to the contraction node(s) for Γ in $T(M)$ the ocurrences of Γ in the copies of $T(N)$.



  This net is still in fusion normal form .

Now we can state our main theorem

Theorem 6.1 (PN_C simulates the *typed* $\lambda\mathtt{x}$ **)** For every term a, if $\Gamma \vdash a : A$ and $a \longrightarrow_{\lambda\mathtt{x}} b$, then for every translation $T(\Gamma \vdash a : A)$ of $\Gamma \vdash a : A$ there is a translation $T(\Gamma \vdash b : A)$ of $\Gamma \vdash b : A$ such that $T(\Gamma \vdash a : A) {\longrightarrow^*}_{PN_C} T(\Gamma \vdash b : A)$. Moreover, if $a \longrightarrow_{\lambda\mathtt{x}-Lambda} b$, then $T(\Gamma \vdash a : A) {\longrightarrow^+}_{PN_C} T(\Gamma \vdash b : A)$ and if $a \longrightarrow_{Lambda} b$, then $T(\Gamma \vdash A : a) = T(\Gamma \vdash b : A)$.

Proof. The proof proceeds by cases:

- If $a \longrightarrow_{\lambda\mathtt{x}} b$ is a root reduction step, one has to consider all the cases.

- If $a \longrightarrow_{\lambda\mathtt{x}} b$ is an internal reduction step, then the result follows from the fact that PN_C as well as $\lambda\mathtt{x}$ are congruences.

The following abstract theorem is a classical tool to show strong normalization from theorem 6.1 (see [14] for details):

Theorem 6.2 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 exists a reduction system $S = \langle \mathcal{O}', R' \rangle$, with a translation T from $\mathcal{O}$ to $\mathcal{O}'$ 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 one takes $\mathcal{O} = \Lambda_{\lambda\mathtt{x}}$, $R_1 = \lambda\mathtt{x} - Lambda$, $R_2 = Lambda$, $\mathcal{O}' = PN$ and $R' = PN_C$, then by theorem 6.1 and the fact that $Lambda$ alone is strongly normalizing, one is able to conclude:

Corollary 6.3 $\longrightarrow_{\lambda\mathtt{x}}$ is strongly normalizing.

7 Conclusion and Future Work

In this paper we establish the connexion between λ-calculi with explicit substitutions and proof nets, and we point out the relation between strong normalization and cut elimination: by this, we formally show that explicit substitutions are an intermediate calculi between typed λ-calculus and proof nets.

The translation proposed in the paper allows us to show that a typed version of $\lambda\mathtt{x}$, and of all the calculi that can be simulated by it such as λ_s, λ_υ, λ_d and λ_f, are strongly normalizing. This allows to conclude that all these calculi are well behaved with respect to normalization, as their untyped versions are already known to preserve β-strong normalization. Our proof technique consists in extending the notion of reduction for Proof Nets in order to deal with n-ary contraction. This extension allows to identify many more proofs than the original system proposed by Girard and is still confluent and strongly normalizing. But there is more to the proof technique than this result alone: our translation suggests that there really exists a typed calculus of explicit substitution with full composition, being able to simulate any one-step β-reduction and yet strongly normalizing (thus avoiding Mellies' counterexample); this is the subject of our current investigation and opens the way to a whole new understanding of the mechanics of explicit substitution. The translation also suggests that the same result can be obtained for a polymorphic version of $\lambda\mathtt{x}$, by using polymorphic proof nets.

References

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

[2] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 211–222, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press.

[3] Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5), 1996.

[4] R. Bloo and R. Geuvers. Explicit substitution: on the edge of strong normalisation. Report 96-10, Eindhoven University of Technology, Department of Mathematics and Computing, Apr. 1996. Accepted for publication in Theoretical Computer Science.

[5] R. Bloo and K. Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. In *Computer Science in the Netherlands (CSN)*, pages 62–72, 1995.

[6] R. Burstall, D. MacQueen, and D. Sanella. Hope: An experimental applicative language. In *Proceedings of the LISP Conference*, pages 136–143, Stanford University, Computer Science Department, July 1980.

[7] P.-L. Curien. *Categorical combinators, sequential algorithms and functional programming*. Progress in Theoretical Computer Science. Birkhaüser, 1986. first edition.

[8] P.-L. Curien, T. Hardin, and A. Rios. Strong normalisation of substitutions. In *MFCS'92*, number 629 in LNCS, pages 209–218, 1992.

[9] V. Danos. *La logique linéaire appliquée à l'étude de divers processus de normalisation (et principalement du λ-calcul).* PhD thesis, Université de Paris VII, 1990. Thèse de doctorat de mathématiques.

[10] V. Danos and L. Regnier. Proof-nets and Hilbert space. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 307–328. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[11] N. de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indag. Mat.*, 5(35):381–392, 1972.

[12] N. de Bruijn. Lambda-calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Mat.*, (40):384–356, 1978.

[13] R. Di Cosmo and K. Delia. Strong normalization of explicit substitutions via cut elimination in proof nets, 1996. Available as `http://www.dmi.ens.fr/~dicosmo/Pub/esll.ps.gz`.

[14] M. C. Ferreira, D. Kesner, and L. Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalization (extended abstract). In M. Hanus and M. Rodríguez-Artalejo, editors, *Proc. of the Fifth International Conference on Algebraic and Logic Programming (ALP)*, number 1139 in LNCS, pages 284–298, 1996.

[15] A. Field and P. Harrison. *Functional programming*. Addison-Wesley, 1988.

[16] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.

[17] 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.

[18] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *19thAnn. ACM Symp. on Principles of Programming Languages (POPL)*, pages 15–26, Albuquerque, New Mexico, 1992. ACM Press.

[19] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[20] H. Herbelin. A λ-calculus structure isomorphic to sequent calculus structure. In *Proceedings of Annual Conference of the European Association for Computer Science Logic (CSL)*, LNCS, 1994.

[21] P. Hudak, S. Peyton-Jones, and P. W. (editors). Report on the programming language haskell, a non-strict, purely functional language (version 1.2). Sigplan Notices, 1992.

[22] F. Kamareddine and A. Ríos. A λ-calculus à la de bruijn with explicit substitutions. In *Proceedings of the International Symposium on Programming Language Implementation and Logic Programming*, number 982 in LNCS. Springer-Verlag, 1995.

[23] F. Kamareddine and A. Ríos. The λ-calculus: its typed and its extended versions. Technical Report TR-95-13, Computing Science department, University of Glasgow, 1995.

[24] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions. In H. Ganzinger, editor, *Proc. of the Seventh International Conference on Rewriting Techniques and Applications (RTA)*, number 1103 in LNCS, pages 184–199, 1996.

[25] Y. Lafont. Interaction nets. In *17thAnn. ACM Symp. on Principles of Programming Languages (POPL)*, pages 95–108, San Francisco, California, 1990. ACM Press.

[26] J. Lamping. An algorithm for optimal lambda calculus reduction. In *19thAnn. ACM Symp. on Principles of Programming Languages (POPL)*, pages 16–30, San Francisco, California, 1990. ACM Press.

[27] P. Lescanne. From λ_σ to λ_υ, a journey through calculi of explicit substitutions. In *Ann. ACM Symp. on Principles of Programming Languages (POPL)*, pages 60–69. ACM, 1994.

[28] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de bruijn's levels. In J. Hsiang, editor, *Proc. of the Sixth International Conference on Rewriting Techniques and Applications (RTA)*, number 914 in LNCS, pages 294–308, 1995.

[29] R. Lins. A new formula for the execution of categorical combinators. In *8th International Conference on Automated Deduction*, number 230 in LNCS, pages 89–98. Springer-Verlag, 1986.

[30] R. Lins. Partial categorical multi-combinators and church rosser theorems. Technical Report 7/92, Computing Laboratory, University of Kent at Canterbury, May 1992.

[31] P.-A. Mellies. Typed λ-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of Int. Conf. on Typed Lambda Calculi and Applications (TLCA)*, volume 902 of *Lecture Notes in Computer Science*, April 1995.

[32] R. Milner, M. Tofte, and R. Harper. *The definition of Standard ML*. MIT Press, 1990.

[33] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, 1996.

[34] K. Rose. Explicit cyclic substitutions. In Rusinowitch and Rémy, editors, *Proc. of the Third International Workshop on Conditional Term Rewriting Systems (CTRS)*, number 656 in LNCS, pages 36–50, 1992.

[35] K. Rose. On explicit binding and substitution preserving strong normlization, 1996. Draft.

[36] D. A. Turner. Miranda: A non-strict functional language with polymorphic types. In J. P. Jouannaud, editor, *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, pages 1–16. Springer-Verlag, 1985. Lecture Notes in Computer Science 201.

[37] F. van Raamsdonk. *Confluence and normalization for higher order rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, 1996.

[38] P. Weis, M. V. Aponte, A. Laville, M. Mauny, and A. Suárez. The CAML reference manual. Technical Report 121, INRIA, Roquencourt B.P.105 - 78153 Le Chesnay Cedex - France, September 1990.

Appendix

$$\frac{}{\vdash A, A^{\perp}} \quad (axiom)$$

$$\frac{\vdash \Gamma, A \qquad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta} \quad (cut)$$

$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta} \quad (times)$$

$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \mathbin{\rotatebox[origin=c]{180}{$\&$}} B} \quad (par)$$

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A} \quad (weakening)$$

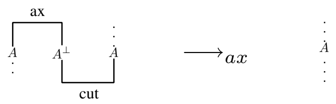
$$\frac{\vdash \Gamma, \overbrace{?A, \ldots, ?A}^{n\ times}}{\vdash \Gamma, ?A} \quad (n\text{-}contraction)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \quad (dereliction)$$

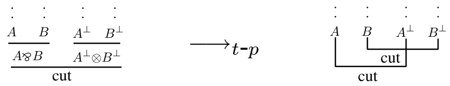
$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A} \quad (box)$$

Table 2. Classical multiplicative-exponential linear logic with n-ary contraction

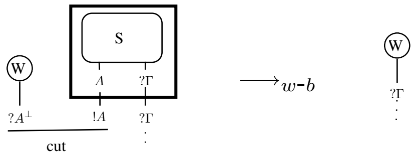
- axiom:



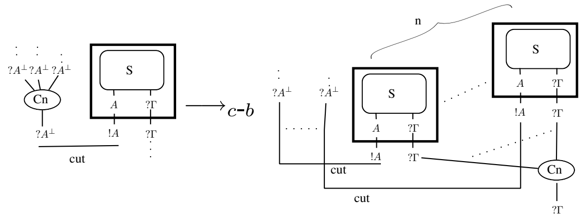
- times-par:



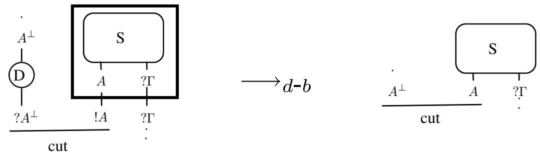
- weakening-box:



- contraction-box:



- dereliction-box:



- box-box:

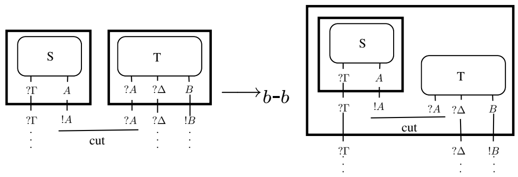


Table 3. The reduction system PN