Explicit operators for λ-calculus

Delia Kesner[1] and Stéphane Lengrand[1,2]

[1] PPS, CNRS and Université Paris 7, France
[2] School of Computer Science, University of St Andrews, United Kingdom.

Abstract We present a simple term language with explicit operators for erasure, duplication and substitution enjoying a sound and complete correspondence with the intuitionistic fragment of Linear Logic's Proof Nets. We establish the good operational behaviour of the language by means of some fundamental properties such as confluence, preservation of strong normalisation, strong normalisation of well-typed terms and step by step simulation. This formalism is the first term calculus with explicit substitutions having full composition and preserving strong normalisation.

1 Introduction

The *Curry-Howard* paradigm, according to which the terms/types/reduction of a term language respectively correspond to the proofs/propositions/normalisation of a logical system, has already shown its numerous merits in the computer science community. Such a correspondence gives a double reading of proofs as programs and programs as proofs, so that insight into one aspect helps the understanding of the other.

A typical example of the Curry-Howard correspondence is obtained by taking the simply typed λ-calculus [Chu41] as term language and Natural Deduction for Intuitionistic Logic as logical system. But both formalisms can be decomposed in the following sense: on the one hand the evaluation rule of λ-calculus, known as β-reduction, can be decomposed into more elementary operations: the various tasks needed to implement β can be achieved by manipulation of various *explicit operators*, such as *erasure, duplication* and *substitution*. On the other hand *Linear Logic* [Gir87] decomposes the intuitionistic logical connectives into more elementary connectives, such as the linear arrow and the exponentials, thus providing a more refined use of resources than that of Intuitionistic Logic.

We show that there is a deep connection between these two elementary decompositions. In order to relate them, we must bridge the conceptual gap between the formalism of a term syntax and that of *Proof Nets* [Gir87] that we use to denote proofs in Linear Logic. Visually convenient to manipulate, the latter retains from the structure of a proof the part that is logically relevant, thus giving geometric insight into proof transformations. However, it is quite cumbersome in proof formalisations. On the other hand, term notation is more convenient to formalise and carry detailed proofs of properties, and also when one wants to implement them via some proof-assistant [Coq,HOL].

Several works [DCK97,DCKP00] have already explored the relation between these two approaches, but none of them has pushed the formalism far enough to obtain a computational counterpart to Proof Nets that is sound and complete with respect to the underlying logical model.

We present a calculus with explicit operators for erasure, duplication and substitution, called $\lambda\mathsf{lxr}$, which can be seen as a functional computational counterpart to Proof Nets. The major features of this calculus are

- Simple syntax and natural semantics via reduction rules and equations;
- Sound and complete correspondence with the Proof Nets model, where the equations and reductions of terms have a natural correspondence with those of Proof Nets;
- Full composition of substitutions;
- Nice properties such as confluence, preservation of strong normalisation, strong normalisation for well-typed terms, and step by step simulation of β-reduction.

Explicit operators and Proof Nets

Much work on explicit substitutions has been done in the last 10 years, for example [ACCL91,BBLRD96,BR95,KR95]. In particular, an unexpected result was given by Melliès [Mel95] who has shown that there are β-strongly normalisable terms in λ-calculus that are not strongly normalisable when evaluated by the reduction rules of an explicit version of the λ-calculus, such as for example $\lambda\sigma$ [ACCL91] or $\lambda\sigma_{\Uparrow}$ [HL89]. In other words, $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$ do not enjoy the property known as Preservation of Strong Normalisation (PSN) [BBLRD96].

This phenomenon shows a defect in the design of these calculi with explicit substitutions because they are supposed to implement their underlying language without losing its good properties. However, there are many ways to avoid Melliès' counter-example in order to recover the PSN property. One of them is to simply forbid the substitution operators to cross lambda-abstractions [LM99,For02]; another consists of avoiding composition of substitutions [BBLRD96]; another one imposes a simple strategy on the calculus with explicit substitutions to mimic exactly the calculus without explicit substitutions [GL98]. The first solution leads to *weak* lambda calculi, not able to express *strong* beta-equality, which is used for example in implementations of proof-assistants [Coq,HOL]. The second solution is drastic as composition of substitutions is needed in implementations of HO unification [DHK95] or functional abstract machines [HMP96]. The last one does not take benefit of the power of explicit operators because substitutions are neither controlled nor delayed.

In order to cope with this problem David and Guillaume [DG01] defined a calculus with *explicit labels*, called λ_{ws}, which allows *controlled* composition of substitutions without losing PSN. These labels are obtained by considering a *weakening* rule in the logical system that specifies the typing rules of λ_{ws}, and then by annotating in the term language the formula introduced by this rule via a label. But the λ_{ws}-calculus has a complicated syntax and its named version [DCKP00] is even more unreadable. On the positive side we should mention that λ_{ws}-calculus has very nice properties as it is confluent (or Church-Rosser) and enjoys PSN. Also, it can be shown [DCKP03] that there is a simple translation preserving reduction from λ_{ws} into the Proof Nets of Linear Logic. This translation gives at the same time an elegant proof of strong normalisation for well-typed λ_{ws}-terms. Moreover, the translation reveals a natural semantics for composition of explicit substitutions, and also suggests that explicit erasure and duplication can be added to the calculus without losing termination. These are the main ideas constituting the starting point of the calculus called $\lambda\mathsf{lxr}$ that we present in this paper.

Explicit operators of typed $\lambda\mathsf{lxr}$ have thus a nice logical interpretation: substitution is *cut*, duplication is *contraction*, erasure is *weakening*. From the point of view of implementation, this can be read as the facts that substitution can be delayed, and that duplication and erasure can be controlled.

Instead of translating a term syntax into Proof Nets, we *extract* a term calculus from Proof Nets, thus defining a simple and natural syntax involving not only reduction rules but also *equations*. Every term equation of $\lambda\mathsf{lxr}$ can be seen as a computational counterpart to an equality between Proof Nets and, vice-versa, every Proof Net equality can be naturally read back as an equality between $\lambda\mathsf{lxr}$-terms.

It is then not surprising that we obtain a full correspondence between typed $\lambda\mathsf{lxr}$ and the Intuitionistic fragment of Linear Logic's Proof Nets in the sense that the interpretation is not only *sound* but also *complete* (in contrast to the translation from λ_{ws} to Proof Nets, which is only sound).

Weakening and Garbage Collection

The *erasure/weakening operator* has an interesting computational behaviour that we illustrate via an example. Let us denote by $W_{\_}(\_)$ the weakening operator, so that a $\lambda\mathsf{lxr}$-term whose variable x is used to weaken the term N is written $W_x(N)$, that is, we explicitly annotate that the variable x does not appear free in the term N. Then, when evaluating the application of a term $\lambda x.W_x(N)$ to another term L, an explicit substitution $\langle x = L\rangle$ is created and the computation will continue with $W_x(N)\langle x = L\rangle$. Then, the weakening operator will be used to prevent the substitution $\langle x = L\rangle$ from going into the term N, thus making more efficient the propagation of a substitution with respect to the original term.

Another interesting feature of our system is that weakening operators are always *pulled out* to the top-level during $\lambda\mathsf{lxr}$-reduction. Moreover, free variables are *never* lost during computation because they get marked as weakening operators. Indeed, if t β-reduces to t', then its $\lambda\mathsf{lxr}$-interpretation reduces to that of t' where weakening operators are added at the top level to keep track of the variables that are lost during the β-reduction. Thus for example, when simulating the β-reduction steps $(\lambda x.\lambda y.x)Nz \longrightarrow^*_\beta N$, the lost variable z will appear in the result of the computation by means of a weakening operator at the top level, i.e. as $W_z(\overline{N})$ (where $\overline{N}$ is the interpretation of N in $\lambda\mathsf{lxr}$), thus preparing the situation for an efficient garbage collection on z.

The explicit weakening operator can thus be seen as a tool for an efficient implementation of *garbage collection*. This feature is not present in λ-calculus, so one can think that λ-calculus is better or simpler, but implementation of functional programming shows that garbage collection improves efficiency and should be taken into account.

It is worth noticing that the labels of the λ_{ws}-calculus cannot be pulled out to the top-level as in $\lambda\mathsf{lxr}$. Also, free variables may be lost during λ_{ws}-computation. Thus, garbage collection within λ_{ws} does not offer the advantages existing in $\lambda\mathsf{lxr}$.

Composition

From a rewriting point of view this calculus can be viewed as the first formalism that is confluent (or Church-Rosser) and strongly normalising on typed terms, *simulates β-reduction step by step*, and has PSN as well as *full composition*. By simulation of β-reduction step by step we mean that every β-reduction step in λ-calculus induces a non-empty $\lambda\mathsf{lxr}$-reduction sequence. By full composition we mean that we can compute the application of an explicit substitution to a term, no matter which substitution remains non-evaluated within that term. In particular, in a term $N\langle y = P\rangle\langle x = L\rangle$, the external substitution is not blocked by the internal one and can be further evaluated without ever requiring any preliminary evaluation of $N\langle y = P\rangle$. In other words, the application of the substitution $\langle x = L\rangle$ to the term N can be evaluated independently from that of $\langle y = P\rangle$. A more technical explanation of the concept of full composition appears in Section 2.

Related work

Besides the λ_{ws}-calculus [DG01] and its encoding in linear logic [DCKP00] already mentioned, other computational meanings of logic via the use of explicit operators have already been proposed.

Herbelin [Her94] proposes a term language with applicative terms and explicit substitutions which corresponds to the Gentzen-style sequent calculus LJT. A similar approach to intuitionistic logic is also studied in [VW01]. In a very different spirit, [CK99] relates the *pattern matching operator* in functional programming to the cut elimination process in sequent calculus for intuitionistic logic.

Abramsky [Abr93] gives computational interpretations for intuitionistic and classical Linear Logic which are based on sequents rather than Proof Nets. As a consequence, no equalities between terms reflect the fact that some proofs of the sequent calculus approach get identified when expressed as Proof Nets. Many other term calculi based on sequents rather than Proof Nets have been proposed for Linear Logic, as for example [GdR00,BBdH93,RR97,Wad93].

A related approach was independently developed by V. van Oostrom (available in course notes written in Dutch [vO01]), where explicit operators for contraction and weakening are added to the λ-calculus to present optimal reduction in a framework with implicit substitutions. We show here how the same operators allow a fine control of composition when using explicit substitutions, although the proofs of some fundamental properties, such as PSN and confluence, become harder. A complete overview on optimal sharing in functional programming languages, and its connection with linear logic can be found in [AG98].

Another approach is taken in [FM99], where a calculus with explicit operators is defined in order to study the notion of "closed reduction" in λ-calculus. Although reduction rules take enormous advantage of the fact that some subterms are closed (i.e. without free variables), which greatly simplifies the definition of reduction, no deep relation with Proof Nets is exploited and no equalities appear at the level of terms.

Our completeness proof is inspired by [Lau03], where polarised Proof Nets are proposed as a sound and complete model of the $\lambda\mu$ calculus [Par92]. Finally, a revised version of the calculus λ_{ws} with names is developed in [Pol04].

Structure of the paper

The rest of the paper is organised as follows. Section 2 presents the syntax and operational semantics of the $\lambda\mathsf{lxr}$-calculus. Section 3 defines the model of the calculus and establishes soundness and completeness. Section 4 shows the relation between λ-calculus and $\lambda\mathsf{lxr}$-calculus by giving mutual translations from one to the other. In Section 5 we state the main operational properties of $\lambda\mathsf{lxr}$. Finally we conclude and give some ideas for further work.

2 The calculus $\lambda\mathsf{lxr}$

We present in this section the syntax of the untyped $\lambda\mathsf{lxr}$-calculus as well as the notions of congruence and reduction between terms.

The syntax for raw terms, given by the following grammar, is extremely simple [1] and can be just viewed as an extension of that of $\lambda\mathsf{x}$ [BR95].

$$t ::= x \mid \lambda x.t \mid t\ t \mid t\langle x = t\rangle \mid W_x(t) \mid C_x^{y,z}(t)$$

The term x is called a *variable*, $\lambda x.t$ an *abstraction*, $t\ u$ an *application*, $t\langle x = u\rangle$ a *closure*, $W_x(t)$ a *weakening* and $C_x^{y,z}(t)$ a *contraction*. The last three constructors are called *explicit operators*. We use also the terminology *explicit substitution* for the occurrence of $\langle x = u\rangle$ in $t\langle x = u\rangle$, even if it is not a syntactic object per se.

[1] In contrast to λ_{ws} with names [DCKP00,DCKP03], where terms affected by substitutions have a complex format $t[x, u, \Gamma, \Delta]$

The terms $\lambda x.t$ and $t\langle x = u\rangle$ define *binders* for the variable x (said to be *bound*) whose scope is t. The term $C_x^{y,z}(t)$ defines a *binder* for y and z (also said to be *bound*) whose scope is t, whereas x is *free* in the terms x, $C_x^{y,z}(t)$ and $W_x(t)$. We write $\mathcal{V}(t)$ to denote the set of variables of the term t and $\mathcal{FV}(t)$ to denote the subset of $\mathcal{V}(t)$ containing only the free ones, which we define formally as follows.

Definition 1 (Variables and Free Variables). *For each term t we define the set of its variables $\mathcal{V}(t)$ and the set of its free variables $\mathcal{FV}(t)$, the latter being a subset of the former, as follows:*

$$\begin{array}{ll@{\qquad}ll}
\mathcal{V}(x) & = \{x\} & \mathcal{FV}(x) & = \{x\} \\
\mathcal{V}(\lambda x.t) & = \mathcal{V}(t) & \mathcal{FV}(\lambda x.t) & = \mathcal{FV}(t) \setminus \{x\} \\
\mathcal{V}(W_x(t)) & = \mathcal{V}(t) \cup \{x\} & \mathcal{FV}(W_x(t)) & = \mathcal{FV}(t) \cup \{x\} \\
\mathcal{V}(C_x^{y,z}(t)) & = \mathcal{V}(t) \cup \{x\} & \mathcal{FV}(C_x^{y,z}(t)) & = \mathcal{FV}(t) \setminus \{y,z\} \cup \{x\} \\
\mathcal{V}(t\ u) & = \mathcal{V}(t) \cup \mathcal{V}(u) & \mathcal{FV}(t\ u) & = \mathcal{FV}(t) \cup \mathcal{FV}(u) \\
\mathcal{V}(t\langle x = u\rangle) & = \mathcal{V}(t) \cup \mathcal{V}(u) & \mathcal{FV}(t\langle x = u\rangle) & = (\mathcal{FV}(t) \setminus \{x\}) \cup \mathcal{FV}(u)
\end{array}$$

As usual we shall consider α-conversion to guarantee that no variable is free and bound in a term at the same time and that bound variables have all different names.

We say that a term is *linear* if it satisfies the following: in every subterm, every variable has at most one free occurrence, and every binder binds a variable that does have a free occurrence (and hence only one). This condition can be formally expressed as follows.

Definition 2 (Linear terms). *A term t is said to be linear if t linear can be derived from the following rules:*

$$\frac{}{x\ \mathsf{linear}} \qquad \frac{t, v\ \mathsf{linear} \quad x \in \mathcal{FV}(t) \quad \mathcal{FV}(t) \cap \mathcal{FV}(v) = \emptyset}{t\langle x = v\rangle\ \mathsf{linear}}$$

$$\frac{t, v\ \mathsf{linear} \quad \mathcal{FV}(t) \cap \mathcal{FV}(v) = \emptyset}{t\ v\ \mathsf{linear}} \qquad \frac{t\ \mathsf{linear} \qquad x \in \mathcal{FV}(t)}{\lambda x.t\ \mathsf{linear}}$$

$$\frac{t\ \mathsf{linear} \quad x, y \in \mathcal{FV}(t) \quad x \neq y \quad z \notin \mathcal{FV}(t)}{C_z^{x,y}(t)\ \mathsf{linear}} \qquad \frac{t\ \mathsf{linear} \quad x \notin \mathcal{FV}(t)}{W_x(t)\ \mathsf{linear}}$$

For instance, the terms $W_x(x)$ and $\lambda x.xx$ are not linear. However, the latter can be represented in the $\lambda\mathsf{lxr}$-calculus by the linear term $\lambda x.C_x^{y,z}(yz)$. More generally, every λ-term can be translated to a linear $\lambda\mathsf{lxr}$-term (c.f. Section 4).

We use Φ, Δ, Σ, $\Pi, \ldots$ to denote finite *lists* of variables (with no repetition).

The notation Φ, Δ will denote the concatenation of Φ and Δ, and we shall always suppose in that case that no variable appears in both Φ and Δ. We use the notation $W_{x_1,\ldots,x_n}(t)$ for $W_{x_1}(\ldots W_{x_n}(t))$. In the case of the empty list, we define $W_{\emptyset}(t) = t$. We use the notation $C^{(y_1,\ldots,y_n),(z_1,\ldots,z_n)}_{x_1,\ldots,x_n}(t)$ for $C^{y_1,z_1}_{x_1}(\ldots C^{y_n,z_n}_{x_n}(t))$.

The following renaming operation will be also used when necessary to ensure linearity.

Definition 3 (Renaming Operation). *If $\Phi = x_1, \ldots, x_n$ and $\Delta = y_1, \ldots, y_n$, we define a renaming operation on a term t w.r.t Φ and Δ, written $R^{\Phi}_{\Delta}(t)$, as the result of simultaneously substituting y_i for every free occurrence x_i in t where $i \in 1 \ldots n$.*

Thus for instance $R^{x,y}_{x',y'}(C^{y,z}_{w}(x(yz))) = C^{y,z}_{w}(x'(yz))$. We remark that whenever π is any permutation of $1 \ldots n$, we have $R^{\Phi}_{\Delta}(t) = R^{\pi(\Phi)}_{\pi(\Delta)}(t)$.

We now introduce a congruence $\equiv$ (i.e. a symmetric, reflexive, transitive relation closed under any context) on terms which brings the typed version of the calculus closer to the Proof Nets modulo as defined in [DCG99]. The relation $\equiv$ is the smallest congruence that includes the axioms in Figure 1 expressing associativity and commutativity for contraction (equations $A, C1_c, C2_c$), commutativity for weakening (equation C_w) and commutativity for independent substitutions (equation S), which is also called *parallel composition*. Terms up to rule S could also be represented using sets of substitutions instead of atomic ones. Finally, contraction and substitution are treated at the same level using axiom $Cont2$.

$C^{x,v}_{w}(C^{z,y}_{x}(t))$	$\equiv_{A}$	$C^{x,y}_{w}(C^{z,v}_{x}(t))$	if $x \neq y, v$
$C^{y,z}_{x}(t)$	$\equiv_{C1_c}$	$C^{z,y}_{x}(t)$	
$C^{y',z'}_{x'}(C^{y,z}_{x}(t))$	$\equiv_{C2_c}$	$C^{y,z}_{x}(C^{y',z'}_{x'}(t))$	if $x \neq y', z'$ & $x' \neq y, z$
$W_x(W_y(t))$	$\equiv_{C_w}$	$W_y(W_x(t))$	
$t\langle x = u\rangle\langle y = v\rangle$	$\equiv_{S}$	$t\langle y = v\rangle\langle x = u\rangle$	if $y \notin \mathcal{FV}(u)$ & $x \notin \mathcal{FV}(v)$ & $x \neq y$
$C^{y,z}_{w}(t)\langle x = u\rangle$	$\equiv_{Cont2}$	$C^{y,z}_{w}(t\langle x = u\rangle)$	if $x \neq w$ & $y, z \notin \mathcal{FV}(u)$

Figure1. Congruence axioms for λlxr-terms

It can easily be proven that the congruence relation defined by the previous rules preserves free variables and linearity. Since we shall deal with rewriting modulo the congruence $\equiv$, it is worth noticing that $\equiv$ is decidable. More than that, each congruence class contains finitely many terms. Indeed, two congruent terms have clearly the same size, so it is easy to see by induction on the size that the congruence rules generate finitely many possibilities to pick up a representative of the class.

As in the case of the renaming operator, for any permutation π of $1 \ldots n$, we have $W_{\Delta}(t) \equiv W_{\pi(\Delta)}(t)$ and $C^{\Delta,\Pi}_{\Phi}(t) \equiv C^{\pi(\Delta),\pi(\Pi)}_{\pi(\Phi)}(t)$. Moreover, we have $C^{\Delta,\Pi}_{\Phi}(t) \equiv C^{\Pi,\Delta}_{\Phi}(t)$ and $C^{\Delta,\Pi}_{\Phi}(C^{\Sigma,\Psi}_{\Delta}(t)) \equiv C^{\Delta,\Psi}_{\Phi}(C^{\Pi,\Sigma}_{\Delta}(t))$.

The congruence $\equiv$ enables us to write "$W_{\mathcal{S}}(u)$", or "$C_{\Phi}^{\Delta,\Pi}(t)$ where $\Phi := \mathcal{S}$" without ordering the variables in $\mathcal{S}$. Besides, we shall sometimes not specify what the lists Δ and Π are, assuming them to be two *disjoint* lists of *fresh* variables.

We shall often write "$C_{\Phi}^{\Phi',\Phi''}(t)$ where $\Phi := \mathcal{S}$", for some set $\mathcal{S}$. This means that the elements of Φ are those of $\mathcal{S}$, which we choose to order in a certain way. Hence we can have $\Phi := \mathcal{S}$ and $\Delta := \mathcal{S}$ with $\Phi \neq \Delta$ but only $\Phi = \pi(\Delta)$ for some bijective π. The above congruence rules ensure that the order we choose is not relevant at all.

When writing "$C_{\Gamma}^{\Phi',\Phi''}(t)$" the lists Φ' and Φ'' could be not specified, that is, they are two *disjoint* lists of *fresh* variables that do not appear anywhere else so far.

2.1 Reduction Rules

The reduction relation of the calculus is the relation generated by the reduction rules in Figure 2 modulo the congruence relation in Figure 1. We will use xr to denote the set of rules $\mathsf{x} \cup \mathsf{r}$ and $B\mathsf{xr}$ to denote the set $\{B\} \cup \mathsf{xr}$. Now, let i be a set of rules (like B, x, xr, or $B\mathsf{xr}$). We denote by $\longrightarrow_{i_s}$ the reduction relation generated by the set i closed under any context, and call it a *strict* reduction. Moreover, since we have a congruence relation on terms, we denote by $\longrightarrow_i$ the reduction relation modulo this congruence, that is, $t \longrightarrow_i t'$ if and only if there exist two terms u and u' such that $t \equiv u \longrightarrow_{i_s} u' \equiv t'$. Hence, the most general reduction relation of our calculus is $\longrightarrow_{B\mathsf{xr}}$, often written $\longrightarrow_{\lambda\mathsf{lxr}}$.

Reduction modulo can also be seen as a reduction relation on equivalence classes of terms. Thus, we have $t \longrightarrow_i t'$ if and only if $cl(t) \longrightarrow_{i_s} cl(t')$, where $cl(t)$ is the class of t with respect to the congruence $\equiv$.

For any reduction relation $\longrightarrow_j$, we denote by $\longrightarrow^+{}_j$ the transitive closure and by $\longrightarrow^*{}_j$ the reflexive and transitive closure.

In order to avoid variable capture, rules Abs and $CAbs$ need the side-conditions $(y \notin \mathcal{FV}(u))$ and $(x \neq y, z)$ respectively, which can always be satisfied by α-conversion, so that their nature is different from that of the other side-conditions. The rules should be understood in the prospect of applying them to linear terms. Indeed, it can be shown (c.f. Lemma 1) that if t is linear and $t \longrightarrow_{\lambda\mathsf{lxr}} t'$, then t' is linear and $\mathcal{FV}(t) = \mathcal{FV}(t')$. The last statement is achieved in particular by the weakening operator, and coincides with the property called "interface preserving" [Laf90] in interaction nets. The fact that linearity is preserved is a essential requirement of the system, so that we can henceforth consider linear terms only.

It is worth noticing that contraction and weakening play a very special role with respect to optimisation issues. Indeed, weakening and contraction can naturally be viewed, respectively, as explicit *erasure* and *duplication* operators. By orienting the $Cont2$ equation from left to right and removing the reduction rules $WSubs$ and $CSubs$, the resulting substitution sub-calculus remains complete, in the sense that explicit substitution can still be propagated and eliminated.

Likewise, this restriction does not break the simulation of β-reduction (i.e. Theorem 7). Moreover, in substitution-free terms, weakening operators may be pulled out to the top level by using rules $WAbs, WApp1, WApp2$ in order to eliminate void substitutions as soon as possible. Symmetrically, contractions may be pushed in by using rules $CAbs, CApp1, CApp2$ in order to delay the duplication of substitutions as much as possible. This property, which fails in terms with explicit substitutions, can be recovered by also considering reduction rules $WSubs$ and $CSubs$ and the right to left direction of equation $Cont2$.

$$
\begin{array}{llll}
(B) & (\lambda x.t)\ u & \longrightarrow & t\langle x = u\rangle
\end{array}
$$

System x

$$
\begin{array}{llllr}
(Abs) & (\lambda y.t)\langle x = u\rangle & \longrightarrow & \lambda y.t\langle x = u\rangle & \\
(App1) & (t\ v)\langle x = u\rangle & \longrightarrow & t\langle x = u\rangle\ v & x \in \mathcal{FV}(t) \\
(App2) & (t\ v)\langle x = u\rangle & \longrightarrow & t\ v\langle x = u\rangle & x \in \mathcal{FV}(v) \\
(Var) & x\langle x = u\rangle & \longrightarrow & u & \\
(Weak1) & W_x(t)\langle x = u\rangle & \longrightarrow & W_{\mathcal{FV}(u)}(t) & \\
(Weak2) & W_y(t)\langle x = u\rangle & \longrightarrow & W_y(t\langle x = u\rangle) & x \neq y \\
(Cont1) & C_x^{y,z}(t)\langle x = u\rangle & \longrightarrow & C_{\Phi}^{\Delta,\Pi}(t\langle y = u_1\rangle\langle z = u_2\rangle) & \\
 & & & & \text{where } \Phi := \mathcal{FV}(u) \\
 & & & & u_1 = R_{\Delta}^{\Phi}(u) \\
 & & & & u_2 = R_{\Pi}^{\Phi}(u) \\
(Comp) & t\langle y = v\rangle\langle x = u\rangle & \longrightarrow & t\langle y = v\langle x = u\rangle\rangle & x \in \mathcal{FV}(v)
\end{array}
$$

System r

$$
\begin{array}{llllr}
(WAbs) & \lambda x.W_y(t) & \longrightarrow & W_y(\lambda x.t) & x \neq y \\
(WApp1) & W_y(u)\ v & \longrightarrow & W_y(uv) & \\
(WApp2) & u\ W_y(v) & \longrightarrow & W_y(uv) & \\
(WSubs) & t\langle x = W_y(u)\rangle & \longrightarrow & W_y(t\langle x = u\rangle) & \\
 & & & & \\
(Merge) & C_w^{y,z}(W_y(t)) & \longrightarrow & R_w^z(t) & \\
(Cross) & C_w^{y,z}(W_x(t)) & \longrightarrow & W_x(C_w^{y,z}(t)) & x \neq y,\ x \neq z \\
 & & & & \\
(CAbs) & C_w^{y,z}(\lambda x.t) & \longrightarrow & \lambda x.C_w^{y,z}(t) & \\
(CApp1) & C_w^{y,z}(t\ u) & \longrightarrow & C_w^{y,z}(t)\ u & y,z \in \mathcal{FV}(t) \\
(CApp2) & C_w^{y,z}(t\ u) & \longrightarrow & t\ C_w^{y,z}(u) & y,z \in \mathcal{FV}(u) \\
(CSubs) & C_w^{y,z}(t\langle x = u\rangle) & \longrightarrow & t\langle x = C_w^{y,z}(u)\rangle & y,z \in \mathcal{FV}(u)
\end{array}
$$

Figure2. Reduction rules for $\lambda\mathsf{lxr}$-terms

Some basic properties of the reduction relation $\lambda\mathsf{lxr}$ are:

Lemma 1 (Basic properties of $\lambda\mathsf{lxr}$). *Let t be a linear term.*

1. *The reductions preserve the set of free variables,* i.e. *if $t \longrightarrow_{\lambda\mathsf{lxr}} t'$, then $\mathcal{FV}(t) = \mathcal{FV}(t')$.*

2. The reductions preserve linearity, i.e. t' *is linear.*
3. t *is a* x*-normal form if and only if* t *has no closure.*

Proof. By using the fact that the congruence preserves free variables and linearity, the first two properties have only to be satisfied by the strict reduction relation. They can be checked when the reduction takes place at the root by treating carefully each rule. Then the proof follows by a straightforward simultaneous induction on the structure of terms.

For the third property, we first remark that if t has no closure, then clearly no x-rule can be applied. Conversely, if t has some closure, then in particular there is a "minimal" subterm $u\langle x = v\rangle$ in t, where u not a closure. This subterm can be reduced in x, so that it is not an x-normal form, hence neither is t.

Notice that the fact that linearity is preserved by congruence and reduction (Lemma 1-2) is a minimal requirement of the system, so that we can henceforth consider linear terms only. For instance in rule $Cont1$, it is the introduction of the lists of fresh variables Δ and Π that ensures linearity of the terms.

In contrast to λ-calculus where the set of free variables may decrease during reduction, preservation of free variables (Lemma 1-1) holds in λlxr thanks to the weakening operator. This coincides with the property called "interface preserving" [Laf90] in interaction nets. It is also worth noticing that the set of variables of a term may either increase (c.f. rule $Cont1$) or decrease (c.f. rules $Var, Merge, Weak1, \ldots$).

Owing to the linearity constraints previously imposed, the $Comp$ rule is equivalent to the following rule $Comp_{bad}$:

$$t\langle y = v\rangle\langle x = u\rangle \longrightarrow t\langle y = v\langle x = u\rangle\rangle \quad x \notin \mathcal{FV}(t)$$

However, if the linearity constraints are not taken into account, the $Comp_{bad}$ rule causes failure of the PSN property [BG99].

The combination of our side-condition and the linearity constraints (which would force x to be free in v) is essential to guarantee PSN and strong normalisation of typed λlxr-terms.

Also, when linearity constraints are not considered, four cases may occur when composing two substitutions as in $t\langle y = v\rangle\langle x = u\rangle$: either (1) $x \in \mathcal{FV}(t) \cap \mathcal{FV}(v)$, or (2) $x \in \mathcal{FV}(t) \setminus \mathcal{FV}(v)$, or (3) $x \in \mathcal{FV}(v) \setminus \mathcal{FV}(t)$, or (4) $x \notin \mathcal{FV}(t) \cup \mathcal{FV}(v)$.

Composition is said to be *partial* in calculi like λ_{ws} [DG01] because only cases (1) and (3) are considered by the reduction rules. Because of the linearity constraints of λlxr, cases (1) and (4) have to be dealt with by the introduction of a contraction for case (1) and a weakening for case (4). Those operators will interact with external substitutions by the use of rules ($Weak1$) and ($Cont1$), respectively. Case (3) is treated by rule ($Comp$), and case (2) by the congruence rule $\equiv_S$. More precisely, the congruence rule can be applied to swap the substitutions, thus allowing the evaluation of the external substitution $\langle x = u\rangle$ without forcing the internal one to be evaluated first. We say in this case that composition is *full* as all cases (1)-(4) are treated. Thus, λlxr turns out to be the first term calculus with explicit substitutions having full composition and preserving β-strong normalisation (Corollary 3).

It is clear that rule B will be used to simulate β-reduction. All the remaining rules in xr are due to the presence of the operators that have been made explicit, and a minimal requirement for those rules is to induce a terminating system. We shall also see in Section 5 that xr is confluent. These properties will give us confidence for the coherence of the features specified by the explicit operators, namely, substitution, erasure, and duplication.

The use of explicit resource operators allow us to derive information about the number of times that a substitution can be duplicated along a sequence of xr-reductions. Indeed, this will happen when a substitution meets a contraction that concerns the substituted variable, which leads us to define *the order of multiplicity* of the substituted variable. Intuitively, this notion corresponds to the number of occurrences of a variable in a λlxr-term t when translating t to its corresponding λ-term free from linearity constraints and explicit resource operators (see Section 4 for details). Moreover, multiplicities of substituted variables constitute a measure which do not increase in rule $Cont1$.

Definition 4 (Multiplicity). *Given a free variable x in a (linear) term t, the multiplicity of x in t, written $\mathcal{M}_x(t)$, is defined by induction on terms as follows:*

$$
\begin{array}{lll}
\mathcal{M}_x(x) & := 1 & \\
\mathcal{M}_x(W_x(t)) & := 1 & \\
\mathcal{M}_x(C_x^{z,w}(t)) & := \mathcal{M}_z(t) + \mathcal{M}_w(t) + 1 & \\
\mathcal{M}_x(W_y(t)) & := \mathcal{M}_x(t) & \\
\mathcal{M}_x(C_y^{z,w}(t)) & := \mathcal{M}_x(t) & \\
\mathcal{M}_x(\lambda y.t) & := \mathcal{M}_x(t) & \\
\mathcal{M}_x((t\ u)) & := \mathcal{M}_x(t) & \textit{if } x \in FV(t) \\
\mathcal{M}_x((t\ u)) & := \mathcal{M}_x(u) & \textit{if } x \in FV(u) \\
\mathcal{M}_x(t\langle y = u\rangle) & := \mathcal{M}_x(t) & \textit{if } x \in FV(t) \setminus \{y\} \\
\mathcal{M}_x(t\langle y = u\rangle) & := \mathcal{M}_y(t) \cdot (\mathcal{M}_x(u) + 1) & \textit{if } x \in FV(u)
\end{array}
$$

We suppose in the last eight cases that $x \neq y, x \neq z, x \neq w$.

Note that $\mathcal{M}_x(t) \geq 1$ provided $x \in FV(t)$.

We can now define a notion of *size* for each term, by weighing the size of a subterm in a substitution with the number of times the substitution may be duplicated.

Definition 5 (Size of terms). *The size of a term is defined by induction as follows:*

$$\begin{array}{ll}\mathcal{S}(x) & = 1\\ \mathcal{S}(W_x(t)) & = \mathcal{S}(t)\\ \mathcal{S}(C_x^{y,z}(t)) & = \mathcal{S}(t)\\ \mathcal{S}(\lambda x.t) & = \mathcal{S}(t)\\ \mathcal{S}(t\ u) & = \mathcal{S}(t) + \mathcal{S}(u)\\ \mathcal{S}(t\langle x = u\rangle) & = \mathcal{S}(t) + \mathcal{M}_x(t) \cdot \mathcal{S}(u)\end{array}$$

We remark that the notions of multiplicity and size are well-defined with respect to the congruence. Indeed, as two non-trivial cases, let us consider the case $C_w^{x,v}(C_x^{y,z}(t)) \equiv C_w^{x,y}(C_x^{z,v}(t))$ for which we have:

$$\mathcal{M}_w(C_w^{x,v}(C_x^{y,z}(t))) = \mathcal{M}_y(t) + \mathcal{M}_z(t) + \mathcal{M}_v(t) + 2 = \mathcal{M}_w(C_w^{x,y}(C_x^{z,v}(t)))$$

and let us consider the case $t\langle x = u\rangle\langle y = v\rangle \equiv t\langle y = v\rangle\langle x = u\rangle$, where $y \notin FV(u)$ and $x \notin FV(v)$, for which we have:

- if $w \in FV(t) \setminus \{x, y\}$, then
  $\mathcal{M}_w(t\langle x = u\rangle\langle y = v\rangle) = \mathcal{M}_w(t) = \mathcal{M}_w(t\langle y = v\rangle\langle x = u\rangle)$;
- if $w \in FV(u)$, then
  $\mathcal{M}_w(t\langle x = u\rangle\langle y = v\rangle) = \mathcal{M}_x(t) \cdot (\mathcal{M}_w(u) + 1) = \mathcal{M}_w(t\langle y = v\rangle\langle x = u\rangle)$;
- if $w \in FV(v)$, then
  $\mathcal{M}_w(t\langle x = u\rangle\langle y = v\rangle) = \mathcal{M}_y(t) \cdot (\mathcal{M}_w(v) + 1) = \mathcal{M}_w(t\langle y = v\rangle\langle x = u\rangle)$.

We then obtain

$$\mathcal{S}(t\langle x = u\rangle\langle y = v\rangle) = \mathcal{S}(t) + \mathcal{M}_x(t) \cdot \mathcal{S}(u) + \mathcal{M}_y(t) \cdot \mathcal{S}(v) = \mathcal{S}(t\langle y = v\rangle\langle x = u\rangle)$$

We have now to show that the size of a term t does not increase when t is reduced by xr. In particular, the size of t strictly decreases for some rules and it remains equal for others. We first need to show that the multiplicities cannot increase.

Lemma 2 (Preservation of multiplicities). *If $t \longrightarrow_{\mathsf{xr}} t'$, then $\mathcal{M}_w(t) \geq \mathcal{M}_w(t')$.*

Proof. First of all note that this is completely false for rule B. Since the congruence steps preserve the multiplicities, we only have to consider the strict reduction relation. Since the multiplicity notion is defined by monotonic induction, it is sufficient to check that the lemma holds when the reduction takes place at the root of t. This is done in Figure 3.

Notice that the last line in Figure 3 holds because abstractions and applications do not alter the multiplicity of a variable.

	Left-hand side		Right-hand side
(Var)	$x\langle x = u\rangle$	$\longrightarrow$	u
	$\mathcal{M}_w(u) + 1$	$>$	$\mathcal{M}_w(u)$
$(Weak1)$	$W_x(t)\langle x = u\rangle$	$\longrightarrow$	$W_{FV(u)}(t)$
$w \in FV(u)$	$\mathcal{M}_w(u) + 1$	$>$	1
$w \in FV(t) \setminus \{x\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$(Cont1)$	$C_x^{y,z}(t)\langle x = u\rangle$	$\longrightarrow$	$C_\Gamma^{\Delta,\Pi}(t\langle y = R_\Delta^\Gamma(u)\rangle\langle z = R_\Pi^\Gamma(u)\rangle)$
$w \in FV(u) = \Gamma$	$(\mathcal{M}_y(t) + \mathcal{M}_z(t) + 1) \cdot (\mathcal{M}_w(u) + 1)$	$>$	$\mathcal{M}_y(t) \cdot (\mathcal{M}_w(u) + 1) + \mathcal{M}_z(t) \cdot (\mathcal{M}_w(u) + 1) + 1$
$w \in FV(t) \setminus \{x, y, z\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$(Comp)$	$t\langle y = v\rangle\langle x = u\rangle$	$\longrightarrow$	$t\langle y = v\langle x = u\rangle\rangle$
$w \in FV(t) \setminus \{y\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w \in FV(v) \setminus \{x\}$	$\mathcal{M}_y(t) \cdot (\mathcal{M}_w(v) + 1)$	$=$	$\mathcal{M}_y(t) \cdot (\mathcal{M}_w(v) + 1)$
$w \in FV(u)$	$\mathcal{M}_y(t) \cdot (\mathcal{M}_x(v) + 1) \cdot (\mathcal{M}_w(u) + 1)$	$>$	$\mathcal{M}_y(t) \cdot (\mathcal{M}_x(v) \cdot (\mathcal{M}_w(u) + 1) + 1)$
Other x	$t\langle x = u\rangle$	$\longrightarrow$	t'
$w \in FV(t) \setminus \{x\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w \in FV(u)$	$\mathcal{M}_w(u)$	$=$	$\mathcal{M}_w(u)$
(Merge)	$C_{w'}^{y,z}(W_y(t))$	$\longrightarrow$	$R_{w'}^z(t)$
$w = w'$	$\mathcal{M}_z(t) + 2$	$>$	$\mathcal{M}_z(t)$
$w \neq w'$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
(WSubs)	$t\langle x = W_y(u)\rangle$	$\longrightarrow$	$W_y(t\langle x = u\rangle)$
$w \in FV(t) \setminus \{x\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w = y$	$\mathcal{M}_x(t) \cdot (1 + 1)$	$>$	1
$w \in FV(u)$	$\mathcal{M}_x(t) \cdot (\mathcal{M}_w(u) + 1)$	$=$	$\mathcal{M}_x(t) \cdot (\mathcal{M}_w(u) + 1)$
(CSubs)	$C_z^{x,y}(t\langle y' = u\rangle)$	$\longrightarrow$	$t\langle y' = C_z^{x,y}(u)\rangle$
$w \in FV(t) \setminus \{y'\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w = z$	$\mathcal{M}_{y'}(t) \cdot (\mathcal{M}_x(u) + \mathcal{M}_y(u) + 2) + 1$	$>$	$\mathcal{M}_{y'}(t) \cdot (\mathcal{M}_x(u) + \mathcal{M}_y(u) + 1 + 1)$
$w \in FV(u)$	$\mathcal{M}_{y'}(t) \cdot (\mathcal{M}_w(u) + 1)$	$=$	$\mathcal{M}_{y'}(t) \cdot (\mathcal{M}_w(u) + 1)$
Other r	t	$\longrightarrow$	t'
	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t')$

Figure3. Preservation of multiplicities

Lemma 3 (Preservation of sizes). *If* $t \longrightarrow_{\mathsf{xr}} u$*, then* $\mathcal{S}(t) \geq \mathcal{S}(u)$*. Moreover, if* $t \longrightarrow_{Var,Weak1,Cont1,Comp} u$*, then* $\mathcal{S}(t) > \mathcal{S}(u)$*.*

Proof. Since the congruence steps preserve the size, we only have to consider the strict reduction relation. The proof can be done by structural induction on terms. The inductive cases are straightforward by using Lemma 2 and the fact that the size of a term is defined by monotonic induction from the sizes of its subterms. We thus only show here the interesting cases which are the root reductions.

Left-hand side		Right-hand side
$x\langle x = u\rangle$	$\longrightarrow_{Var}$	u
$1 + \mathcal{S}(u)$	$>$	$\mathcal{S}(u)$
$W_x(t)\langle x = u\rangle$	$\longrightarrow_{Weak1}$	$W_{FV(u)}(t)$
$\mathcal{S}(t) + \mathcal{S}(u)$	$>$	$\mathcal{S}(t)$
$C_x^{y,z}(t)\langle x = u\rangle$	$\longrightarrow_{Cont1}$	$C_{\Gamma}^{\Delta,\Pi}(t\langle y = R_{\Delta}^{\Gamma}(u)\rangle\langle z = R_{\Pi}^{\Gamma}(u)\rangle)$
$\mathcal{S}(t) + \mathcal{S}(u) \cdot (\mathcal{M}_y(t) + \mathcal{M}_z(t) + 1)$	$>$	$\mathcal{S}(t) + \mathcal{S}(u) \cdot \mathcal{M}_y(t) + \mathcal{S}(u) \cdot \mathcal{M}_z(t)$
$t\langle y = v\rangle\langle x = u\rangle$	$\longrightarrow_{Comp}$	$t\langle y = v\langle x = u\rangle\rangle$
$\mathcal{S}(t) + \mathcal{M}_y(t) \cdot (\mathcal{S}(v) + (\mathcal{M}_x(v) + 1) \cdot \mathcal{S}(u))$	$>$	$\mathcal{S}(t) + \mathcal{M}_y(t) \cdot (\mathcal{S}(v) + \mathcal{M}_x(v) \cdot \mathcal{S}(u))$
t	$\longrightarrow_{\text{Other } \mathsf{xr}}$	t'
$\mathcal{S}(t)$	$=$	$\mathcal{S}(t')$

Left-hand side		Right-hand side
$x\langle x = u\rangle$	$\longrightarrow_{Var}$	u
$1 + \mathcal{S}(u)$	$>$	$\mathcal{S}(u)$
$W_x(t)\langle x = u\rangle$	$\longrightarrow_{Weak1}$	$W_{FV(u)}(t)$
$\mathcal{S}(t) + \mathcal{S}(u)$	$>$	$\mathcal{S}(t)$
$C_x^{y,z}(t)\langle x = u\rangle$	$\longrightarrow_{Cont1}$	$C_{\Gamma}^{\Delta,\Pi}(t\langle y = R_{\Delta}^{\Gamma}(u)\rangle\langle z = R_{\Pi}^{\Gamma}(u)\rangle)$
$\mathcal{S}(t) + \mathcal{S}(u) \cdot (\mathcal{M}_y(t) + \mathcal{M}_z(t) + 1)$	$>$	$\mathcal{S}(t) + \mathcal{S}(u) \cdot \mathcal{M}_y(t) + \mathcal{S}(u) \cdot \mathcal{M}_z(t)$
$t\langle y = v\rangle\langle x = u\rangle$	$\longrightarrow_{Comp}$	$t\langle y = v\langle x = u\rangle\rangle$
$\mathcal{S}(t) + \mathcal{M}_y(t) \cdot \mathcal{S}(v) +$		
$\mathcal{M}_y(t) \cdot (\mathcal{M}_x(v) + 1) \cdot \mathcal{S}(u)$	$>$	$\mathcal{S}(t) + \mathcal{M}_y(t) \cdot (\mathcal{S}(v) + \mathcal{M}_x(v) \cdot \mathcal{S}(u))$
t	$\longrightarrow$Other xr	t'
$\mathcal{S}(t)$	$=$	$\mathcal{S}(t')$

The last line holds because the size measure forgets weakenings, contractions, abstractions and applications.

We now use a polynomial interpretation to show termination of the subsystem of xr containing only the rules that might not decrease the size.

Definition 6. *We define a polynomial interpretation $\mathcal{I}(\_)$ from $\lambda\mathsf{lxr}$-terms to natural numbers as follows:*

$$
\begin{aligned}
\mathcal{I}(x) &= 2\\
\mathcal{I}(W_x(t)) &= \mathcal{I}(t) + 1\\
\mathcal{I}(\lambda x.t) &= 2 \cdot \mathcal{I}(t) + 2\\
\mathcal{I}(t\ u) &= 2 \cdot (\mathcal{I}(t) + \mathcal{I}(u)) + 2\\
\mathcal{I}(C_x^{y,z}(t)) &= 2 \cdot \mathcal{I}(t)\\
\mathcal{I}(t\langle x = u\rangle) &= \mathcal{I}(t) \cdot (\mathcal{I}(u) + 1)
\end{aligned}
$$

Remark 1. For any term t, $\mathcal{I}(t) \geq 2$.

Proposition 1. *The polynomial interpretation $\mathcal{I}(\_)$ is sound with respect to the congruence $\equiv$.*

Proof. The polynomial interpretation is blind to the variables' names, so it is trivially sound with respect to α-conversion, and rules A, $C1_c$, $C2_c$ and C_w. For the equivalence rule S we have by commutativity of multiplication the following equality:

$$\mathcal{I}(t\langle x = u\rangle\langle y = v\rangle) = \mathcal{I}(t) \cdot \mathcal{I}(u) \cdot \mathcal{I}(v) = \mathcal{I}(t\langle y = v\rangle\langle x = u\rangle)$$

For the equivalence rule $Cont2$ we have:

$$\mathcal{I}(C_w^{y,z}(t)\langle x = u\rangle) = 2 \cdot \mathcal{I}(t) \cdot \mathcal{I}(u) + 2 \cdot \mathcal{I}(t) = \mathcal{I}(C_w^{y,z}(t\langle x = u\rangle))$$

Lemma 4 (Decrease of $\mathcal{I}(\_)$). *If $t \longrightarrow_{\mathsf{xr}} u$ and the reduction is neither Var, $Weak1$, $Cont1$ nor $Comp$, then $\mathcal{I}(t) > \mathcal{I}(u)$.*

Proof. Since the congruence steps preserve the polynomial interpretation, we only have to consider the strict reduction relation. The proof can be done by structural induction on terms. The cases of root reductions are:

Rule	left-hand side		right-hand side
(Abs)	$(2 \cdot \mathcal{I}(t) + 2) \cdot (\mathcal{I}(u) + 1)$	$>$	$2 \cdot \mathcal{I}(t) \cdot (\mathcal{I}(u) + 1) + 2$
$(App1)$	$(2 \cdot (\mathcal{I}(t) + \mathcal{I}(v)) + 2) \cdot (\mathcal{I}(u) + 1)$	$>$	$2 \cdot (\mathcal{I}(t) \cdot (\mathcal{I}(u) + 1) + \mathcal{I}(v)) + 2$
$(App2)$	$(2 \cdot (\mathcal{I}(t) + \mathcal{I}(v)) + 2) \cdot (\mathcal{I}(u) + 1)$	$>$	$2 \cdot (\mathcal{I}(t) + \mathcal{I}(v) \cdot (\mathcal{I}(u) + 1)) + 2$
$(Weak2)$	$(\mathcal{I}(t) + 1) \cdot (\mathcal{I}(u) + 1)$	$>$	$\mathcal{I}(t) \cdot (\mathcal{I}(u) + 1) + 1$
$(WAbs)$	$2 \cdot (\mathcal{I}(t) + 1) + 2$	$>$	$2 \cdot \mathcal{I}(t) + 2 + 1$
$(WApp1)$	$2 \cdot (\mathcal{I}(u) + 1 + \mathcal{I}(v)) + 2$	$>$	$2 \cdot (\mathcal{I}(u) + \mathcal{I}(v)) + 2 + 1$
$(WApp2)$	$2 \cdot (\mathcal{I}(u) + \mathcal{I}(v) + 1) + 2$	$>$	$2 \cdot (\mathcal{I}(u) + \mathcal{I}(v)) + 2 + 1$
$(WSubs)$	$\mathcal{I}(t) \cdot (\mathcal{I}(u) + 1 + 1)$	$>$	$\mathcal{I}(t) \cdot (\mathcal{I}(u) + 1) + 1$
$(Merge)$	$2 \cdot (\mathcal{I}(t) + 1)$	$>$	$\mathcal{I}(t)$
$(Cross)$	$2 \cdot (\mathcal{I}(t) + 1)$	$>$	$2 \cdot \mathcal{I}(t) + 1$
$(CAbs)$	$2 \cdot (2 \cdot \mathcal{I}(t) + 2)$	$>$	$2 \cdot (2 \cdot \mathcal{I}(t)) + 2$
$(CApp1)$	$2 \cdot (2 \cdot (\mathcal{I}(t) + \mathcal{I}(u)) + 2)$	$>$	$2 \cdot (2 \cdot \mathcal{I}(t) + \mathcal{I}(u)) + 2$
$(CApp2)$	$2 \cdot (2 \cdot (\mathcal{I}(t) + \mathcal{I}(u)) + 2)$	$>$	$2 \cdot (\mathcal{I}(t) + 2 \cdot \mathcal{I}(u)) + 2$
$(CSubs)$	$2 \cdot \mathcal{I}(t) \cdot (\mathcal{I}(u) + 1)$	$>$	$\mathcal{I}(t) \cdot (2 \cdot \mathcal{I}(u) + 1)$

We can conclude this section with the following property

Theorem 1. *The system xr is strongly normalising.*

Proof. Every rule of xr decreases the pair of integers $(\mathcal{S}(t), \mathcal{I}(t))$ w.r.t the lexicographical order induced by them.

2.2 Typing Rules

We only consider here linear terms to which we will add type information.

We now define a typing system for λlxr. *Types* are defined by means of the following syntax, where σ ranges over a countable set of atomic types.

$$A ::= \sigma \mid A \to A$$

An *environment* is a *set* of decorated variables of the form $x : A$, where A is any type. A *judgement* is a triplet of the form $\Gamma \vdash t : A$, where Γ is an environment, t is a λlxr-term, and A is a type.

We shall sometimes write $\Gamma = \mathcal{S}$ (where $\mathcal{S}$ is a set of variables) forgetting the types in Γ. We shall still use this notation within terms, forgetting the types, and assigning a random order, which still does not matter because of the congruence $\equiv$.

We say that t is *well-typed* if there is an environment Γ and a type A such that the judgement $\Gamma \vdash t : A$ is derivable from the set of *typing rules* in Figure 4.

$$\frac{}{x : A \vdash x : A}\ (Axiom) \qquad \frac{\Gamma, x : B \vdash t : A \qquad \Delta \vdash M : B}{\Gamma, \Delta \vdash t\langle x = M\rangle : A}\ (Subs)$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Delta \vdash v : A}{\Gamma, \Delta \vdash (t\ v) : B}\ (App) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (Lambda)$$

$$\frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash C_z^{x,y}(M) : B}\ (Cont) \qquad \frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash W_x(t) : A}\ (Weak)$$

Figure4. Typing Rules for λlxr-terms

Remark that $\Gamma \vdash t : A$ always implies that the domain of Γ is exactly $\mathcal{FV}(t)$. Also, when writing $\Gamma, x : B$, the variable x is supposed to be fresh w.r.t Γ. As expected, the following holds:

Theorem 2 (Subject Reduction).

- *If $\Gamma \vdash s : A$ and $s \equiv s'$, then $\Gamma \vdash s' : A$.*
- *If $\Gamma \vdash s : A$ and $s \longrightarrow_{\lambda lxr} s'$, then $\Gamma \vdash s' : A$.*

Proof. To prove the first point one proceeds by induction on terms. The proof is straightforward so we can skip to the proof of the second statement for which we already know that the possible congruent steps taking place before and after the reduction step preserve the typing judgement. We can now look at each reduction step. We only show here the cases where the reduction takes place at the root, as the inductive definition of the typing trees makes the closure under context straightforward.

- (B): We have $s = (\lambda x.t)\ u$ and $s' = t\langle x = u\rangle$.

$$\frac{\dfrac{\Gamma, x : B \vdash t : A}{\Gamma \vdash (\lambda x.t) : B \to A} \quad \Delta \vdash u : B}{\Gamma, \Delta \vdash (\lambda x.t)\ u : A} \qquad\qquad \frac{\Gamma, x : B \vdash t : A \quad \Delta \vdash u : B}{\Gamma, \Delta \vdash t\langle x = u\rangle : A}$$

- (Abs): We have $s = (\lambda y.t)\langle x = u\rangle$, $s' = \lambda y.t\langle x = u\rangle$ and $A = B \to C$.

$$\frac{\dfrac{\Gamma, x : D, y : B \vdash t : C}{\Gamma, x : D \vdash (\lambda y.t) : B \to C} \quad \Delta \vdash u : D}{\Gamma, \Delta \vdash (\lambda y.t)\langle x = u\rangle : B \to C} \qquad\qquad \frac{\dfrac{\Gamma, x : D, y : B \vdash t : C \quad \Delta \vdash u : D}{\Gamma, y : B, \Delta \vdash t\langle x = u\rangle : C}}{\Gamma, \Delta \vdash \lambda y.t\langle x = u\rangle : B \to C}$$

- $(App1)$: We have $s = (t\ v)\langle x = u\rangle$ and $s' = t\langle x = u\rangle\ v$.

$$\frac{\dfrac{\Gamma, x : B \vdash t : C \to A \quad \Delta \vdash v : C}{\Gamma, \Delta, x : B \vdash t\ v : A} \quad \Pi \vdash u : B}{\Gamma, \Delta, \Pi \vdash (t\ v)\langle x = u\rangle : A} \qquad \frac{\dfrac{\Gamma, x : B \vdash t : C \to A \quad \Pi \vdash u : B}{\Gamma, \Pi \vdash t\langle x = u\rangle : C \to A} \quad \Delta \vdash v : C}{\Gamma, \Pi, \Delta \vdash t\langle x = u\rangle\ v : A}$$

- $(App2)$: Idem.
- (Var): We have $s = x\langle x = u\rangle$ and $s' = u$.

$$\frac{x : A \vdash x : A \quad \Delta \vdash u : A}{\Delta \vdash x\langle x = u\rangle : A} \qquad\qquad \Delta \vdash u : A$$

- $(Weak1)$: We have $s = W_x(t)\langle x = u\rangle$ and $s' = W_{\mathcal{FV}(u)}(t)$.

$$\frac{\dfrac{\Gamma \vdash t : A}{\Gamma, x : B \vdash W_x(t) : A} \quad \Pi \vdash u : B}{\Gamma, \Pi \vdash W_x(t)\langle x = u\rangle : A} \qquad\qquad \frac{\Gamma \vdash t : A}{\Gamma, \Pi \vdash W_\Pi(t) : A}$$

- $(Weak2)$: We have $s = W_y(t)\langle x = u\rangle$ and $s' = W_y(t\langle x = u\rangle)$.

$$\frac{\dfrac{\Gamma, x : B \vdash t : A}{\Gamma, y : C, x : B \vdash W_y(t) : A} \quad \Pi \vdash u : B}{\Gamma, y : C, \Pi \vdash W_y(t)\langle x = u\rangle : A} \qquad\qquad \frac{\dfrac{\Gamma, x : B \vdash t : A \quad \Pi \vdash u : B}{\Gamma, \Pi \vdash t\langle x = u\rangle : A}}{\Gamma, y : C, \Pi \vdash W_y(t\langle x = u\rangle) : A}$$

- $(Cont1)$: $s = C_x^{y,z}(t)\langle x = v\rangle$
and $s' = C_\Pi^{\Delta,\Sigma}(t\langle y = R_\Delta^\Pi(v)\rangle\langle z = R_\Sigma^\Pi(v)\rangle)$ where $\Pi := \mathcal{FV}(v)$.

$$\frac{\dfrac{\Gamma, y : B, z : B \vdash t : A}{\Gamma, x : B \vdash C_x^{y,z}(t) : A} \quad \Pi \vdash v : B}{\Gamma, \Pi \vdash C_x^{y,z}(t)\langle x = v\rangle : A}$$

$$\frac{\dfrac{\dfrac{\Gamma, y : B, z : B \vdash t : A \quad \dfrac{\Pi \vdash v : B}{\Delta \vdash R_\Delta^\Pi(v) : B}}{\Gamma, z : B, \Delta \vdash t\langle y = R_\Delta^\Pi(v)\rangle : A} \quad \dfrac{\Pi \vdash v : B}{\Sigma \vdash R_\Sigma^\Pi(v) : B}}{\Gamma, \Delta, \Sigma \vdash t\langle y = R_\Delta^\Pi(v)\rangle\langle z = R_\Sigma^\Pi(v)\rangle : A}}{\Gamma, \Pi \vdash C_\Pi^{\Delta,\Sigma}(t\langle y = R_\Delta^\Pi(v)\rangle\langle z = R_\Sigma^\Pi(v)\rangle) : A}$$

– $(Comp)$: $s = t\langle y = u\rangle\langle x = v\rangle$ and $s' = t\langle y = u\langle x = v\rangle\rangle$.

$$\dfrac{\dfrac{\Lambda, y : C \vdash t : A \quad \Pi, x : B \vdash u : C}{\Lambda, \Pi, x : B \vdash t\langle y = u\rangle : A} \quad \Delta \vdash v : B}{\Lambda, \Pi, \Delta \vdash t\langle y = u\rangle\langle x = v\rangle : A} \qquad \dfrac{\Lambda, y : C \vdash t : A \quad \dfrac{\Pi, x : B \vdash u : C \quad \Delta \vdash v : B}{\Pi, \Delta \vdash u\langle x = v\rangle : C}}{\Lambda, \Pi, \Delta \vdash t\langle y = u\langle x = v\rangle\rangle : A}$$

– $(WAbs)$: We have $s = W_y(\lambda x.t)$ and $s' = \lambda x.W_y(t)$.

$$\dfrac{\dfrac{\Gamma, x : B \vdash t : C}{\Gamma \vdash \lambda x.t : B \to C}}{y : D, \Gamma \vdash W_y(\lambda x.t) : B \to C} \qquad \dfrac{\dfrac{\Gamma, x : B \vdash t : C}{y : D, \Gamma, x : B \vdash W_y(t) : C}}{y : D, \Gamma \vdash \lambda x.W_y(t) : B \to C}$$

– $(WApp1)$: We have $s = W_y(u)\ v$ and $s' = W_y(uv)$.

$$\dfrac{\dfrac{\Gamma \vdash u : B \to C}{\Gamma, y : D \vdash W_y(u) : B \to C} \quad \Delta \vdash v : B}{\Gamma, y : D, \Delta \vdash W_y(u)\ v : C} \qquad \dfrac{\dfrac{\Gamma \vdash u : B \to C \quad \Delta \vdash v : B}{\Gamma, \Delta \vdash u\ v : C}}{\Gamma, y : D, \Delta \vdash W_y(uv) : C}$$

– $(WApp2)$: We have $s = u\ W_y(v)$ and $s' = W_y(uv)$.

$$\dfrac{\Gamma \vdash u : B \to C \quad \dfrac{\Delta \vdash v : B}{\Delta, y : D \vdash W_y(v) : B}}{\Gamma, y : D, \Delta \vdash u\ W_y(v) : C} \qquad \dfrac{\dfrac{\Gamma \vdash u : B \to C \quad \Delta \vdash v : B}{\Gamma, \Delta \vdash u\ v : C}}{\Gamma, y : D, \Delta \vdash W_y(uv) : C}$$

– $(WSubs)$: We have $s = t\langle x = W_y(u)\rangle$ and $s' = W_y(t\langle x = u\rangle)$.

$$\dfrac{\dfrac{\Gamma \vdash u : B}{\Gamma, y : C \vdash W_y(u) : B} \quad \Delta, x : B \vdash t : A}{\Gamma, y : C, \Delta \vdash t\langle x = W_y(u)\rangle : A} \qquad \dfrac{\dfrac{\Gamma \vdash u : B \quad \Delta, x : B \vdash t : A}{\Gamma, \Delta \vdash t\langle x = u\rangle : A}}{\Gamma, y : C, \Delta \vdash W_y(t\langle x = u\rangle) : A}$$

– $(Merge)$: $s = C^{y,z}_w(W_y(t))$ and $s' = t$.

$$\dfrac{\dfrac{\Gamma, z : C \vdash t : A}{\Gamma, y : C, z : C \vdash W_y(t) : A}}{\Gamma, w : C \vdash C^{y,z}_w(W_y(t)) : A} \qquad \dfrac{\Gamma, z : C \vdash t : A}{\Gamma, w : C \vdash R^z_w(t) : A}$$

– $(Cross)$: $s = C^{y,z}_w(W_x(t))$ and $s' = W_x(C^{y,z}_w(t))$.

$$\dfrac{\dfrac{\Gamma, y : C, z : C \vdash t : A}{\Gamma, y : C, z : C, x : B \vdash W_x(t) : A}}{\Gamma, w : C, x : B \vdash C^{y,z}_w(W_x(t)) : A} \qquad \dfrac{\dfrac{\Gamma, y : C, z : C \vdash t : A}{\Gamma, w : C \vdash C^{y,z}_w(t) : A}}{\Gamma, w : C, x : B \vdash W_x(C^{y,z}_w(t)) : A}$$

– $(CAbs)$: $s = C^{y,z}_w(\lambda x.t)$ and $s' = \lambda x.C^{y,z}_w(t)$.

$$\dfrac{\dfrac{\Gamma, y : D, z : D, x : B \vdash t : C}{\Gamma, y : D, z : D \vdash \lambda x.t : B \to C}}{\Gamma, w : D \vdash C^{y,z}_w(\lambda x.t) : B \to C} \qquad \dfrac{\dfrac{\Gamma, y : D, z : D, x : B \vdash t : C}{\Gamma, w : D, x : B \vdash C^{y,z}_w(t) : C}}{\Gamma, w : D \vdash \lambda x.C^{y,z}_w(t) : B \to C}$$

– $(CApp1)$: $s = C^{y,z}_w(t\ u)$ and $s' = C^{y,z}_w(t)\ u$.

$$\dfrac{\dfrac{\Gamma, y : C, z : C \vdash t : A \to B \quad \Delta \vdash u : A}{\Gamma, y : C, z : C, \Delta \vdash (t\ u) : B}}{\Gamma, w : C, \Delta \vdash C^{y,z}_w(t\ u) : B} \qquad \dfrac{\dfrac{\Gamma, y : C, z : C \vdash t : A \to B}{\Gamma, w : C \vdash C^{y,z}_w(t) : A \to B} \quad \Delta \vdash u : A}{\Gamma, w : C, \Delta \vdash (C^{y,z}_w(t)\ u) : B}$$

– $(CApp2)$: Idem

– $(CSubs)$: $s = C^{y,z}_w(t\langle x = u\rangle)$ and $s' = t\langle x = C^{y,z}_w(u)\rangle$.

$$\dfrac{\dfrac{\Gamma, y : B, z : B \vdash u : C \quad \Delta, x : C \vdash t : A}{\Gamma, \Delta, y : B, z : B \vdash t\langle x = u\rangle : A}}{\Gamma, \Delta, w : B \vdash C^{y,z}_w(t\langle x = u\rangle) : A} \qquad \dfrac{\dfrac{\Gamma, y : B, z : B \vdash u : C}{\Gamma, w : B \vdash C^{y,z}_w(u) : C} \quad \Delta, x : C \vdash t : A}{\Gamma, \Delta, w : B \vdash t\langle x = C^{y,z}_w(u)\rangle}$$

3 A model for λlxr

This section is devoted to show two of the main properties of our calculus. The first one (Theorem 4) concerns strong normalisation of well-typed terms, which is achieved by translating well-typed λlxr-terms to MELL Proof Nets.

Theorem 5 and Theorem 6 show that the translation from λlxr to Proof Nets is sound complete w.r.t the appropriate equivalence relations on terms and proof nets respectively.

We briefly recall here the traditional notion of Proof Nets of Linear Logic and some of its basic properties. We refer the interested reader to [Gir87] for more details.

Let $\mathcal{A}$ be a set of *atomic formulae* equipped with an involutive function $\perp : \mathcal{A} \to \mathcal{A}$, called *linear negation*. The set of formulae of the multiplicative exponential fragment of linear logic (called MELL) is defined as follows:

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

The formula $\mathcal{F} \parr \mathcal{G}$ denotes a "non-economic" version of the classical disjunction, whereas $?\mathcal{F}$ and $!\mathcal{F}$ are used to indicate where contraction or weakening can take place.

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

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

The set of Proof Nets, denoted PN, is defined inductively in Figure 5.

Proof Nets are the *computational objects* behind Linear Logic, where the notion of reduction (called also "cut elimination") corresponds exactly to the cut-elimination procedure on sequent derivations. The traditional reduction system for MELL consists in the set of *cut elimination rules* appearing in Figure 6.

Unfortunately, the original notion of reduction on PN is not well adapted to simulate either the β rule of λ-calculus, or the rules dealing with propagation of

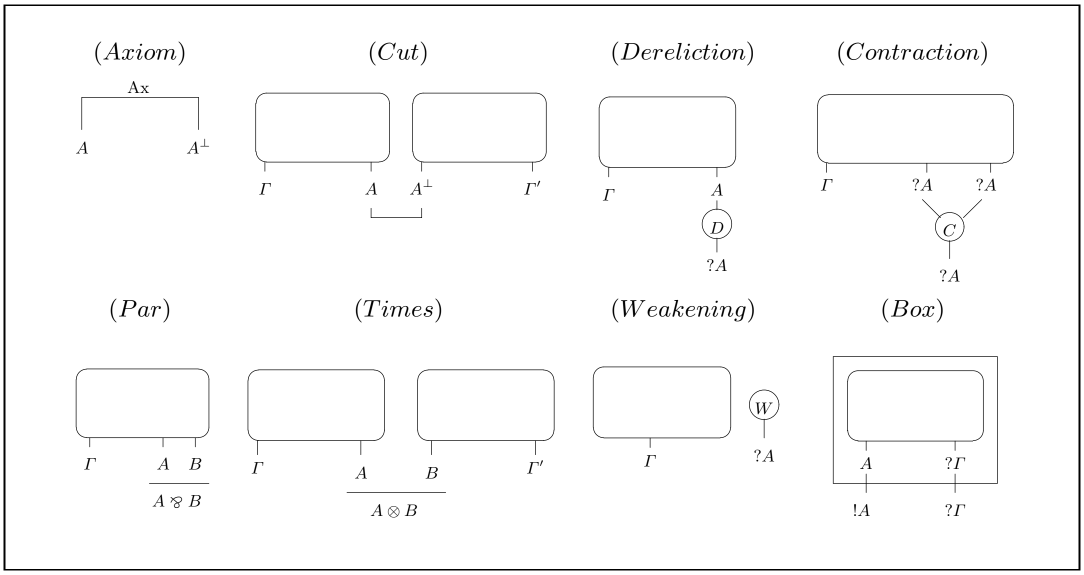


Figure5. MELL Proof Nets

substitution in explicit substitution calculi: too many inessential details about the order of application of the rules are still present, and in order to get rid of them, one is naturally led to define an equivalence relation on PN, as is done in [DCG99], where the axioms $\sim_A$ and $\sim_B$ are introduced (see Figure 7).

Equivalence A turns contraction into an associative operator, and corresponds to forgetting the order in which the contraction rule is used. Equivalence B abstracts away the relative order of application of the rules of box-formation and contraction on the premises of a box. Finally, besides the equivalence relation defined in [DCG99], we shall also need two extra reduction rules in Figure 8 : *elim* is used to remove unneeded weakening links when simulating explicit substitutions and *pull* allows weakening links to go outside boxes.

Notation: Henceforth, we shall call R the system made of rules *Ax-cut*, $⅋$-$\otimes$, *w-b*, *d-b*, *c-b*, *b-b* and *elim* and *pull*. We shall write $\sim_E$ for the congruence (reflexive, symmetric, transitive, closed by contexts) relation on Proof Nets generated by axioms A, B. We shall write R/E for the system made of the rules in R and the axioms in A, B. System R/E is actually defining a notion of *reduction modulo an equivalence relation*, so we write $r \longrightarrow_{R/E} s$ if and only if there exist r' and s' such that $r \sim_E r' \longrightarrow_R s' \sim_E s$.

In order to prove the properties of the translation from typed λlxr to Proof Nets, we shall use the following result:

Theorem 3. *The reduction relation $\longrightarrow_{R/E}$ terminates.*

Proof. This result is proved in [Pol04] for which we refer the interested reader for full details.

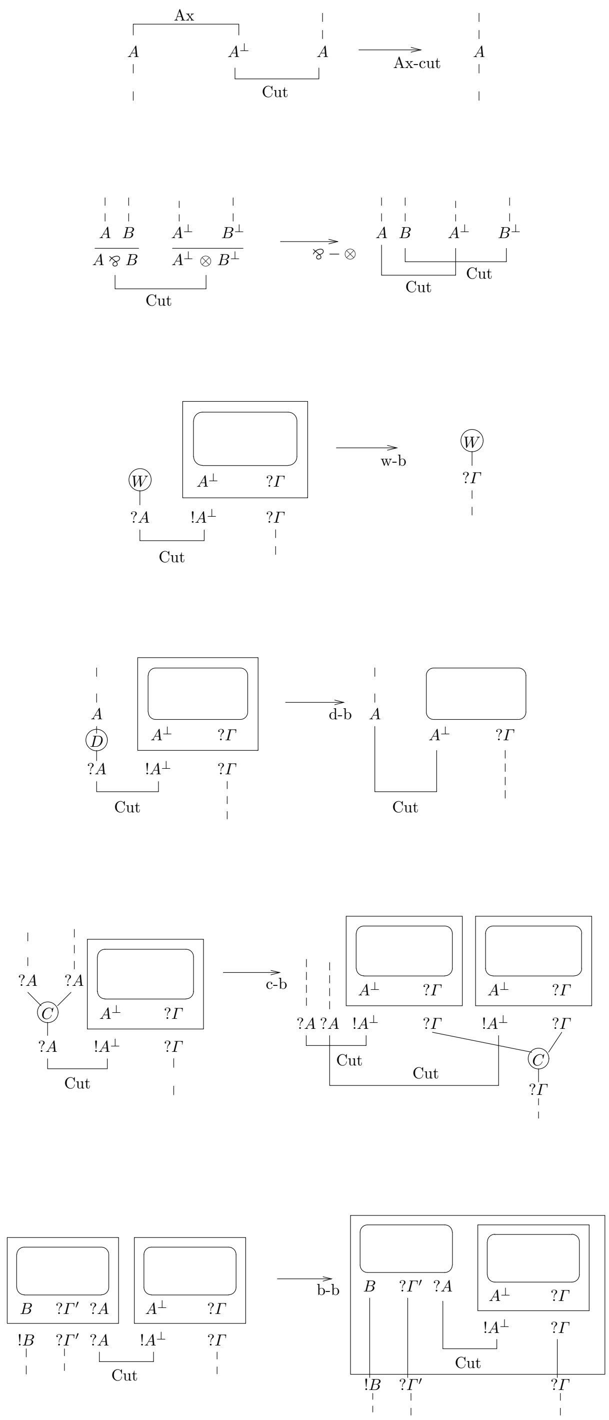


Figure6. Cut elimination rules for MELL Proof Nets

3.1 Interpreting Terms into Proof Nets

We now present the natural interpretation of typed λlxr-terms as Proof Nets. For that, we use the translation of types introduced in [DJS95] given by :

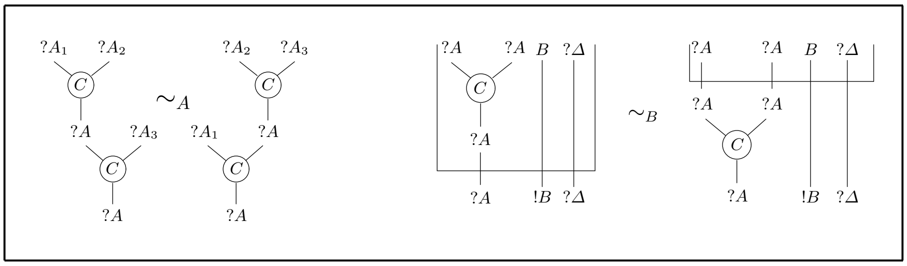


Figure7. Axioms for MELL Proof Nets

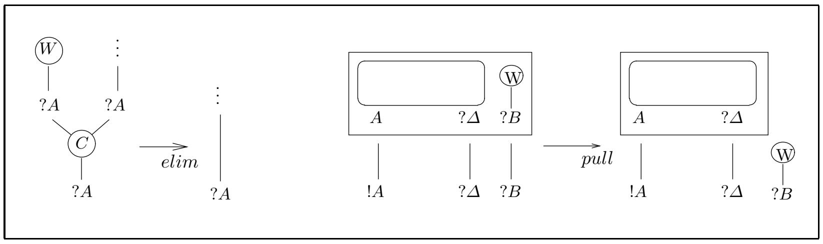


Figure8. Extra reduction rules for MELL Proof Nets

$$\begin{array}{lll} A^* & = A & \text{for atomic types} \\ (A \to B)^* & = ?((A^*)^\perp) \parr B^* & \text{otherwise} \end{array}$$

Figure 9 defines the translation $T(\_)$ from derivable typing judgements of λlxr to Proof Nets. Every proof net $T(\Gamma \vdash t : A)$ has one wire labelled with $?(B^*)^\perp$ for every $B \in \Gamma$ and one unique wire labelled with A^*. We shall often write $T(t)$ instead of $T(\Gamma \vdash t : A)$ when Γ and A do not matter or are clear (from Subject Reduction, for example).

The translation $T(\_)$ satisfies the following properties:

Lemma 5 (Simulation of λlxr-reduction). *Let t be a well-typed λlxr-term.*

- *If $t \equiv t'$, then $T(t) \sim_E T(t')$.*
- *If $t \longrightarrow_B t'$, then $T(t) \longrightarrow^+{}_{R/E} T(t')$.*
- *If $t \longrightarrow_{\mathsf{xr}} t'$, then $T(t) \longrightarrow^*{}_{R/E} T(t')$.*

Proof. In order to show the first property it is sufficient to check the base cases as the relations $\equiv$ and $\sim_E$ are congruences.

– For $C_w^{x,v}(C_x^{y,z}(t)) \equiv_A C_w^{x,y}(C_x^{z,v}(t))$ we have the two following equivalent proof nets.

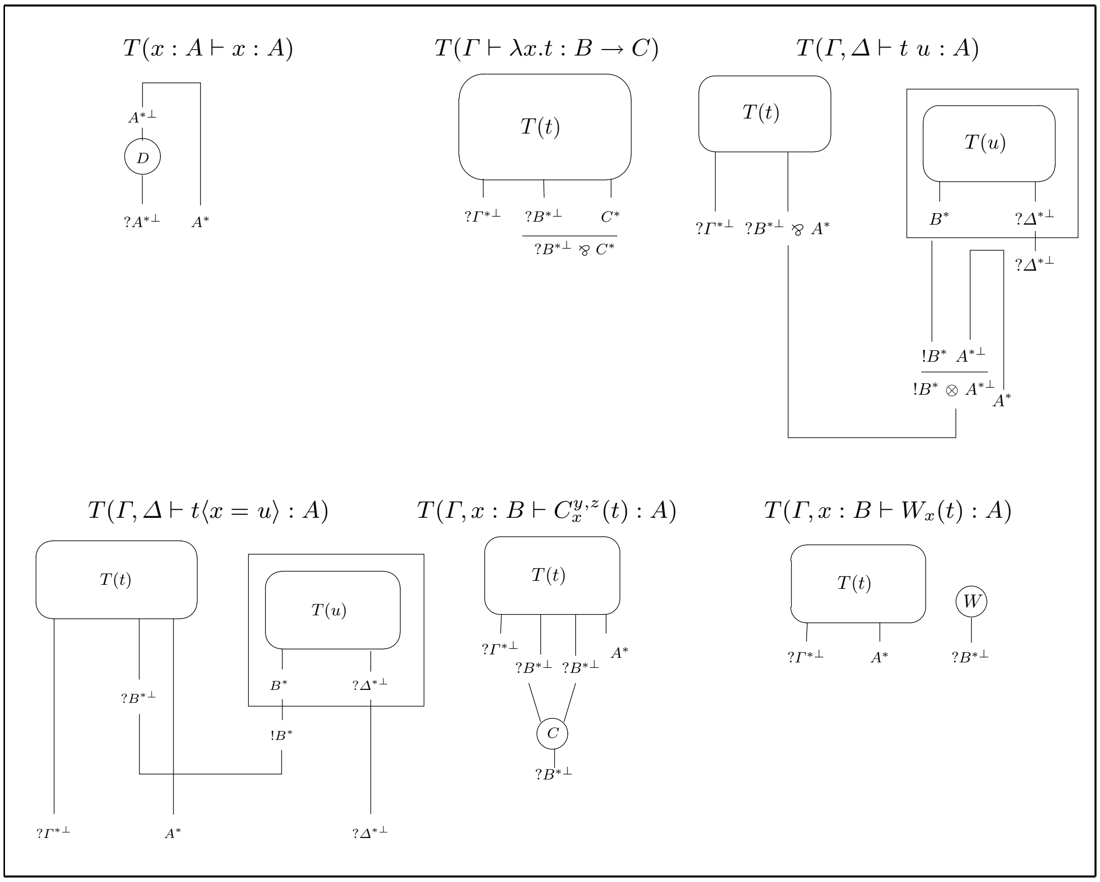
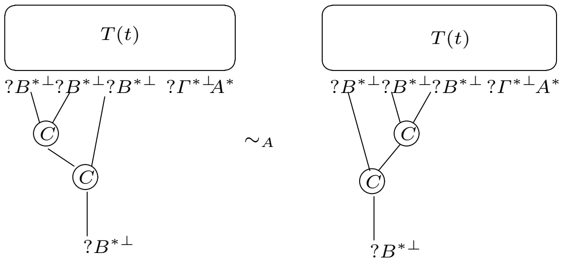
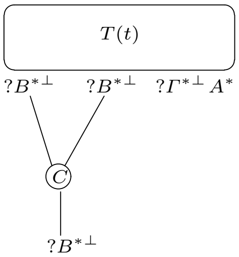


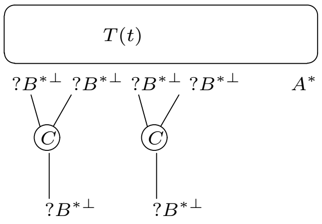
Figure9. Encoding typed λlxr-terms into MELL proof-nets



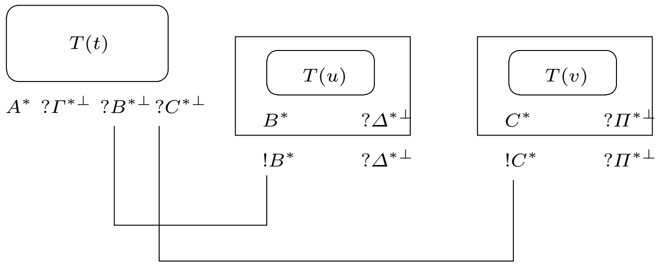
– For $C_x^{y,z}(t) \equiv_{C1_c} C_x^{z,y}(t)$ we have exactly the same interpretation for both terms:



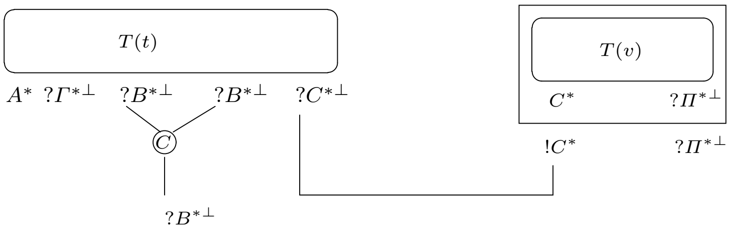
- For $C^{y',z'}_{x'}(C^{y,z}_{x}(t)) \equiv_{C2_c} C^{y,z}_{x}(C^{y',z'}_{x'}(t))$ we have exactly the same interpretation for both terms:



- For $t \equiv_{C_w} t'$ we also have the same interpretation.
- For $t\langle x = u\rangle\langle y = v\rangle \equiv_S t\langle y = v\rangle\langle x = u\rangle$, if $y \notin FV(u)$ and $x \notin FV(v)$, let $\Delta := FV(u)$ and $\Pi := FV(v)$. We have exactly the same interpretation for both terms:

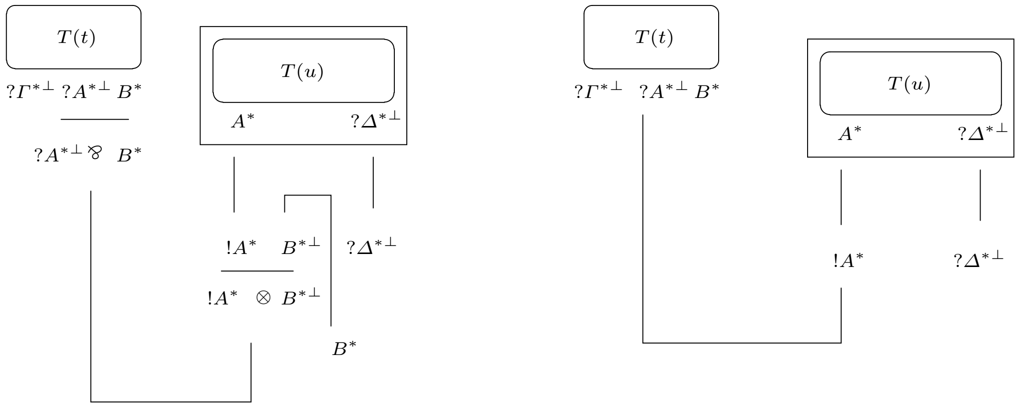


- For $C^{y,z}_{w}(t)\langle x = v\rangle \equiv_{Cont2} C^{y,z}_{w}(t\langle x = v\rangle)$, we have exactly the same interpretation.

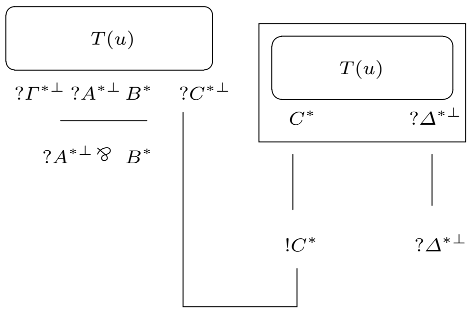


For the second and third property, we proceed by induction on terms. We only show here the cases of root reductions.

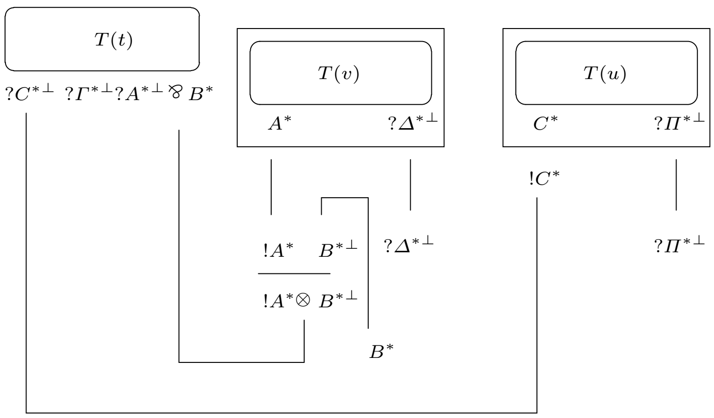
- For $t_1 = (\lambda x.t)\ u \longrightarrow_B t\langle x = u\rangle = t_2$, let $\Gamma := FV(\lambda x.t)$ and $\Delta := FV(u)$. We have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{\parr-\otimes, Ax-cut} T(t_2)$.



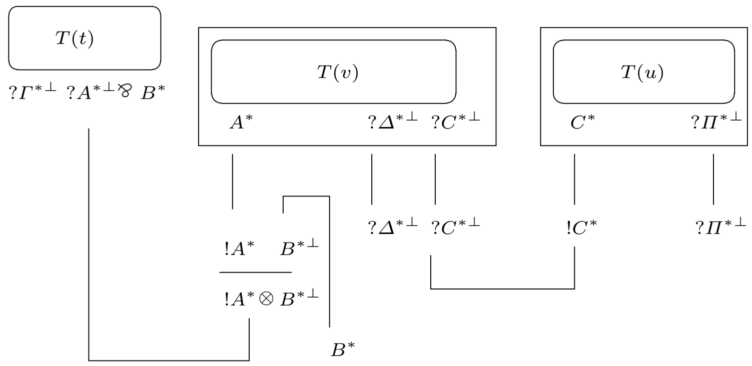
– For $t_1 = (\lambda y.t)\langle x = u\rangle \longrightarrow_{Abs} \lambda y.t\langle x = u\rangle = t_2$, let $\Gamma, x := FV(\lambda y.t)$ and $\Delta := FV(u)$. We have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

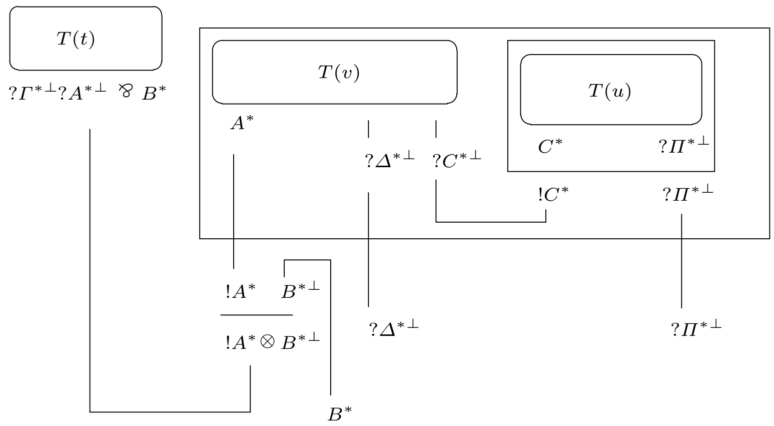


– For $t_1 = (t\ v)\langle x = u\rangle \longrightarrow_{App1} (t\langle x = u\rangle\ v) = t_2$ with $x \in FV(t)$, exactly the same interpretation $T(t_1)$ and $T(t_2)$.

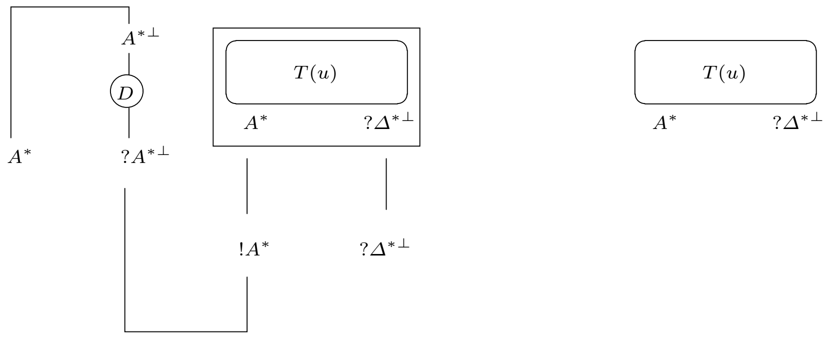


– For $t_1 = (t\ v)\langle x = u\rangle \longrightarrow_{App2} (t\ v\langle x = u\rangle) = t_2$ with $x \in FV(v)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{b-b} T(t_2)$.

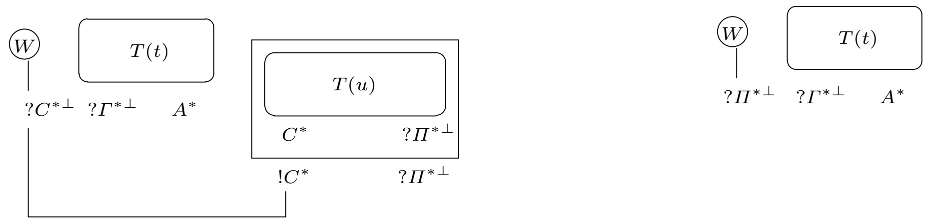




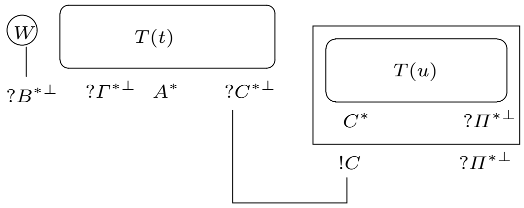
– For $t_1 = x\langle x = u\rangle \longrightarrow_{Var} \ u = t_2$, let $\Delta := FV(u)$. We have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{d-b,Ax-cut} T(t_2)$.



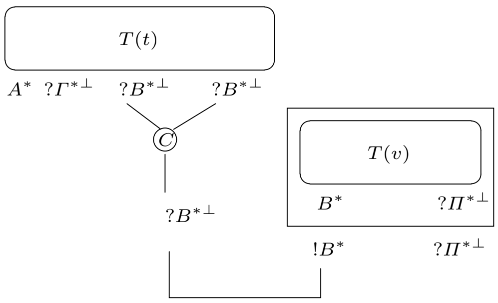
– For $t_1 = W_x(t)\langle x = u\rangle \longrightarrow_{Weak1} \ W_\Pi(t) = t_2$, where $\Pi := FV(u)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{w-b} T(t_2)$.

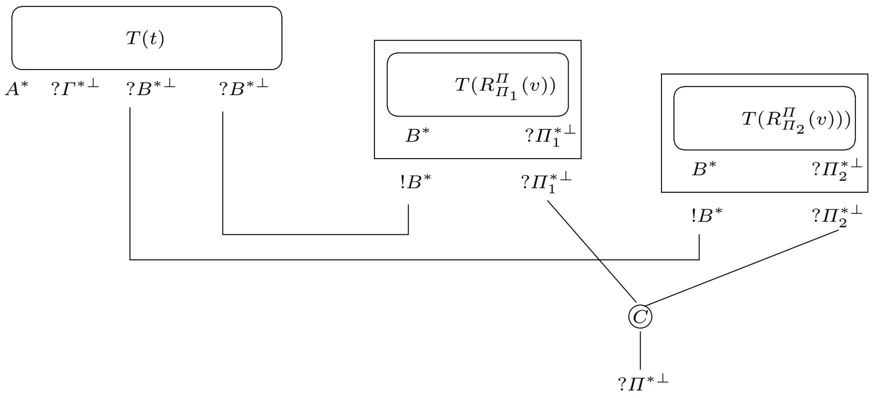


– For $t_1 = W_y(t)\langle x = u\rangle \longrightarrow_{Weak2} W_y(t\langle x = u\rangle) = t_2$, where $x \neq y$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

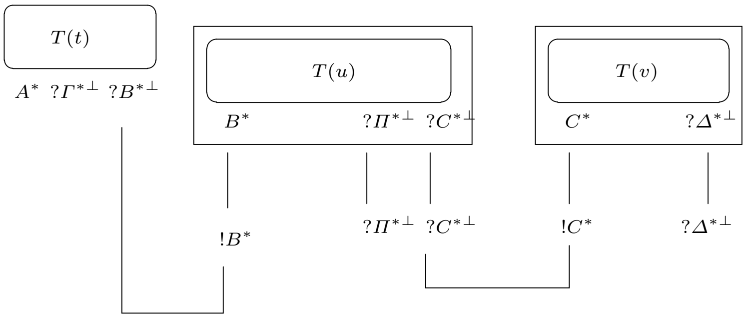


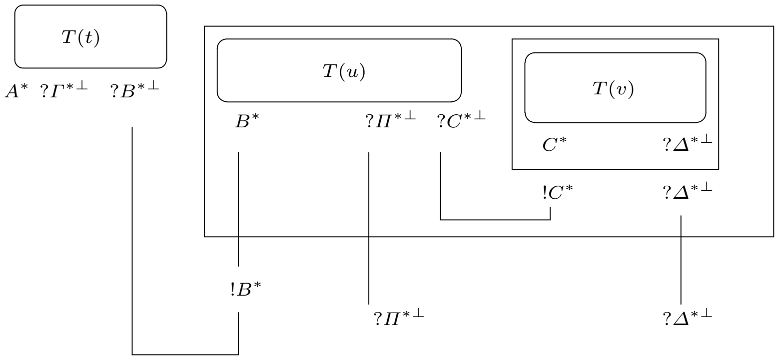
– For $t_1 = C_x^{y,z}(t)\langle x = v\rangle \longrightarrow_{Cont1} C_{\Pi}^{\Pi_1,\Pi_2}(t\langle y = R_{\Pi_1}^{\Pi}(v)\rangle\langle z = R_{\Pi_2}^{\Pi}(v)\rangle) = t_2$, where $\Pi := FV(v)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{c-b} T(t_2)$.



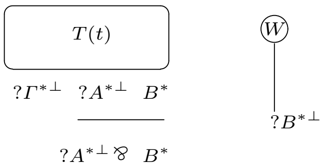


– For $t_1 = t\langle y = u\rangle\langle x = v\rangle \longrightarrow_{Comp} t\langle y = u\langle x = v\rangle\rangle = t_2$ where $x \in FV(u)$, let $(\Gamma, y) := FV(t)$, $(\Pi, x) := FV(u)$ and $\Delta := FV(v)$. We have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{b-b} T(t_2)$.

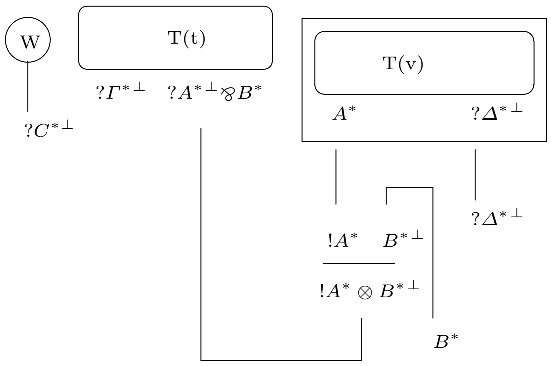




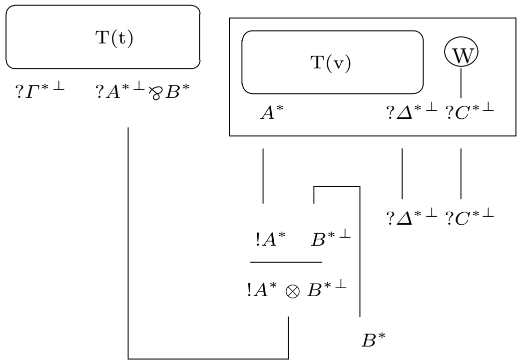
– For $t_1 = W_y(\lambda x.t) \longrightarrow_{WAbs} \lambda x.W_y(t) = t_2$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

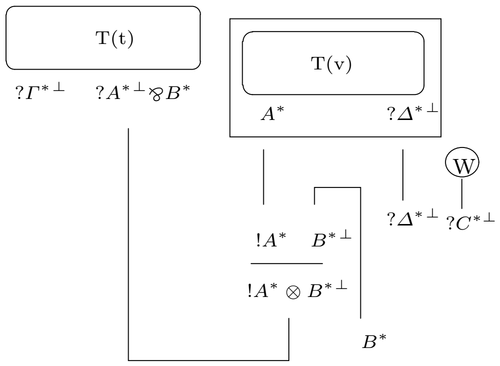


– For $t_1 = W_y(u)\ v \longrightarrow_{WApp1} W_y(uv) = t_2$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

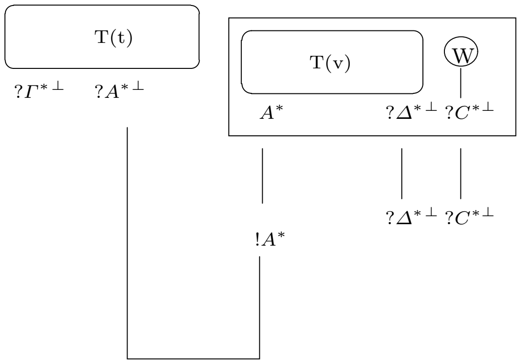


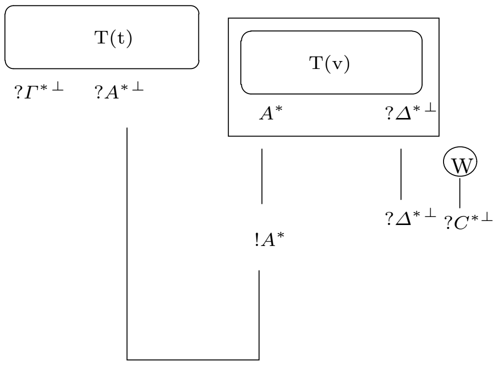
– For $t_1 = u\ W_y(v) \longrightarrow_{WApp2}\ W_y(uv) = t_2$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{pull}\ T(t_2)$.



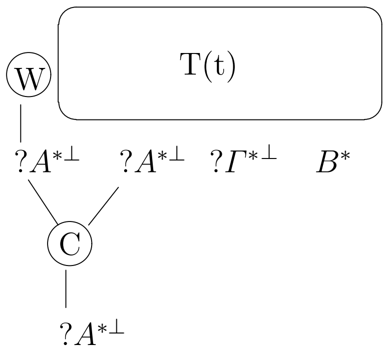


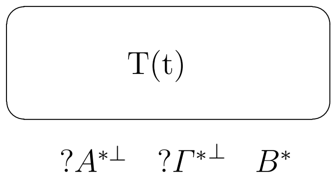
– For $t_1 = t\langle x = W_y(u)\rangle \longrightarrow_{WSubs}\ W_y(t\langle x = u\rangle) = t_2$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{pull}\ T(t_2)$.



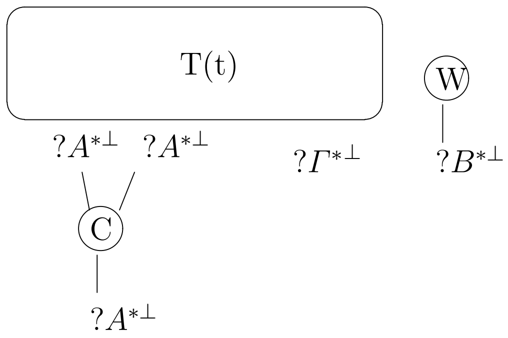


– For $t_1 = C_w^{y,z}(W_y(t)) \longrightarrow_{Merge}\ R_w^z(t) = t_2$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \longrightarrow_{elim}\ T(t_2)$.

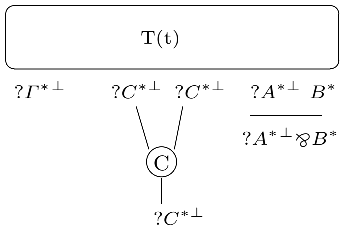




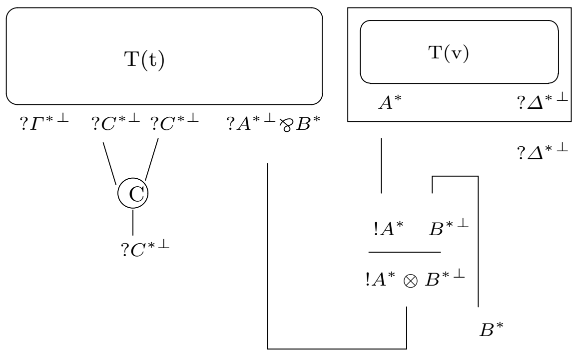
– For $t_1 = C_w^{y,z}(W_x(t)) \longrightarrow_{Cross}\ W_x(C_w^{y,z}(t)) = t_2$ when $x \neq y,\ \ x \neq z$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.



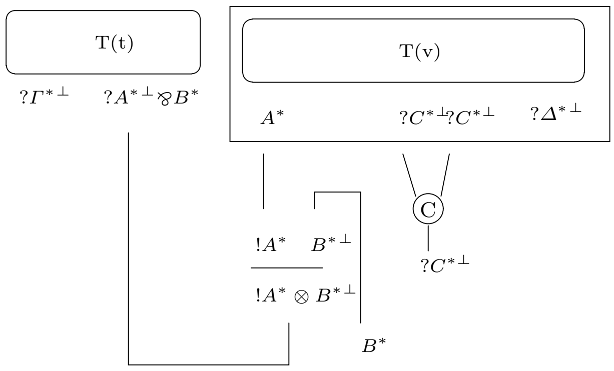
– For $t_1 = C_w^{y,z}(\lambda x.t) \longrightarrow_{CAbs} \lambda x.C_w^{y,z}(t) = t_2$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

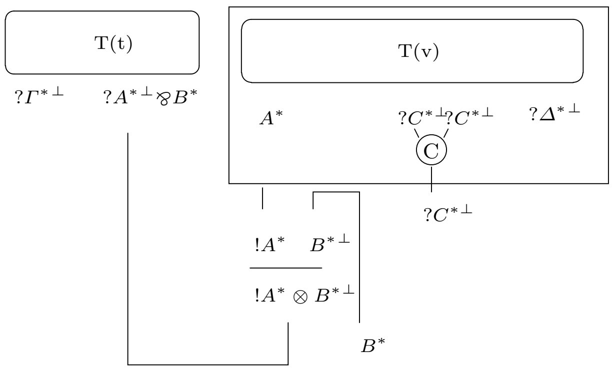


– For $t_1 = C_w^{y,z}(t\ u) \longrightarrow_{CApp1} C_w^{y,z}(t)\ u = t_2$ when $y, z \in FV(t)$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

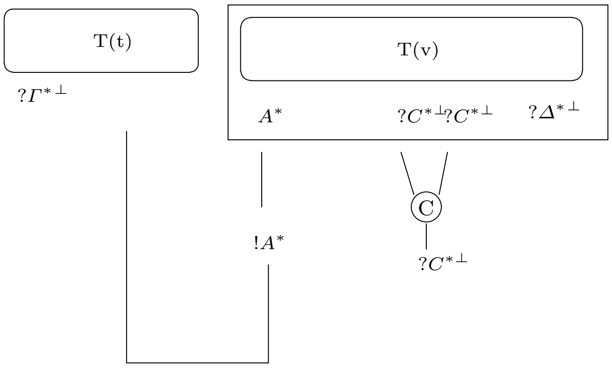


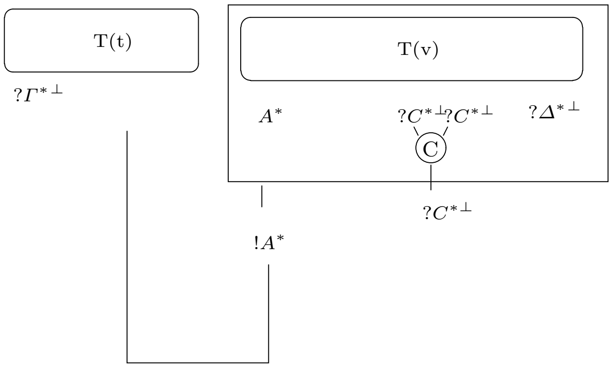
– For $t_1 = C_w^{y,z}(t\ u) \longrightarrow_{CApp2} t\ C_w^{y,z}(u) = t_2$ when $y, z \in FV(u)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \sim_B T(t_2)$.





– For $t_1 = C_w^{y,z}(t\langle x = u\rangle) \longrightarrow_{CSubs} t\langle x = C_w^{y,z}(u)\rangle$ when $y, z \in FV(u)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \sim_B T(t_2)$.





As a consequence we obtain one of the main important properties of λlxr:

Theorem 4 (Strong Normalisation).
The relation $\longrightarrow_{\lambda\mathsf{lxr}}$ is strongly normalising on well-typed λlxr-terms.

Proof. Suppose $\longrightarrow_{\lambda\mathsf{lxr}}$ is not strongly normalising. Then, since xr terminates by Theorem 1, an infinite λlxr-reduction sequence would have infinitely many B-steps. But this would lead by Lemma 5 to an infinite R/E-reduction sequence which is impossible by Theorem 3.

In order to show Theorem 4 another technique [Pol04] using Preservation of Strong Normalisation (c.f. Section 5) together with the strong normalisation of typed λ-calculus could be used. Direct proofs using for instance reducibility by perpetuality [Bon01,LLD+04] seem much more difficult to adapt to our case. We remark however that those techniques are no simpler than ours, as many auxiliary properties about the computational behaviour of the calculus need to be establish.

3.2 Towards completeness

The relevant computational steps of the cut-elimination procedure in Proof Nets are those modifying the box structure. Hence, we are interested in characterising those terms that are translated by $T(\_)$ into Proof Nets that have identical box structures. Let TB be the reduction relation on PN generated by the rules that do not modify the box structure, namely *pull*, *elim*, *Ax-cut* and $\parr$-$\otimes$, modulo the congruence $\sim_E$. Since the reduction rules of TB are not overlapping and termination holds for the whole system R/E which contains TB, then the following turns out to be straightforward:

Proposition 2. *The reduction relation TB is confluent and terminating. Hence, the normal form of a proof net r w.r.t this reduction relation, written $TB(r)$, exists and is unique up to the congruence $\sim_E$.*

Hence, "having the same box structure" can be expressed by the following equivalence:

Definition 7. *Given two proof nets r and r' we have $r \approx r'$ if and only if $TB(r) \sim_E TB(r')$.*

We would like to relate two terms having the same box structure when viewed as proof nets. For that we use the following notion:

Definition 8. *We define the congruence $\cong$ between $\lambda\mathsf{lxr}$-terms by adding to $\equiv$ the following rules turned into equalities:*

$$\{B, Abs, App1, Weak2, WAbs, WApp1, WApp2, Cross, Merge, CAbs, CApp1, CApp2\}$$

Remark that $WSubs$ and $CSubs$ are included in $\cong$, i.e. if $t =_{WSubs \cup CSubs} t'$, then $t \cong t'$.

The following property relates two $\cong$-convertible terms w.r.t. their semantic translations into Proof Nets and is one of the interesting results about $\lambda\mathsf{lxr}$.

Definition 9. *Given a derivable typing judgements $\Gamma \vdash t : A$, we define its translation $NT(\Gamma \vdash t : A)$ as $TB(T(\Gamma \vdash t : A))$. We will often write $NT(t)$ instead of $NT(\Gamma \vdash t : A)$ when Γ and A do not matter or are clear from the context. Remark that by definition we have $T(t_1) \approx T(t_2)$ iff $NT(t_1) \sim_E NT(t_2)$.*

The following property relates two $\cong$-convertible terms w.r.t their semantic translations into Proof Nets.

Theorem 5 (Soundness). *Given two well-typed $\lambda\mathsf{lxr}$-terms t_1, t_2, if $t_1 \cong t_2$, then $T(t_1) \approx T(t_2)$.*

Proof. Since the relations $\cong$ and $\sim_E$ are congruences it is sufficient to check the axioms.

- For $t_1 =_B t_2$, we have shown that $T(t_1) \longrightarrow^*_{\parr-\otimes, Ax-cut} T(t_2)$ (c.f. proof of Lemma 5) so that $NT(t_1) = NT(t_1)$ and thus $NT(t_1) \sim_E NT(t_2)$ trivially holds.

- For $t_1 =_{Abs,WAbs,CAbs,App1,WApp1,CApp1,CApp2,Weak2,Cross} t_2$, we have shown that $T(t_1) \sim_E T(t_2)$ (c.f. proof of Lemma 5) so that $NT(t_1) \sim_E NT(t_2)$ also holds.
- For $t_1 =_{WApp2} t_2$, we have shown that $T(t_1) \longrightarrow_{pull} T(t_2)$ (c.f. proof of Lemma 5) so that $NT(t_1) = NT(t_2)$ and thus $NT(t_1) \sim_E NT(t_2)$ trivially holds.
- For $t_1 =_{Merge} t_2$, we have shown that $T(t_1) \longrightarrow_{elim} T(t_2)$ (c.f. proof of Lemma 5) so that $NT(t_1) = NT(t_2)$ and thus $NT(t_1) \sim_E NT(t_2)$ trivially holds.
- For all the other cases we have already shown that $T(t_1) \sim_E T(t_2)$ (c.f. proof of Lemma 5) so that $NT(t_1) \sim_E NT(t_2)$ holds.

We proceed now to show the completeness result. We first establish a property of terms.

Definition 10 (Term Complexity). *We define the complexity of a term t, written $|t|$, as follows: $|x| = 1$, $|\lambda x.t| = |W_x(t)| = |C_x^{y,z}(t)| = |t| + 1$, and $|tv| = |t\langle x = v\rangle| = |t| + |v| + 1$.*

Lemma 6. *For every term t, there is a term $t' \cong t$ with $|t'| \leq |t|$ such that either $t' = \lambda x.t''$ or $t' = W_x(t'')$ or $t' = C_x^{y,z}(t'')$
or $t' = (xt_1 \dots t_n)\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$
or $t' = W_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ with $x_i \notin FV(t'')$ for all i
or $t' = C_x^{y,z}(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ with $x_i \notin FV(t'')$ for all i.*

Proof. By induction on $|t|$. In each of the following cases we obtain $|t'| \leq |t|$ by induction hypothesis and because the way we use $\cong$ to convert the terms can only decrease their complexity.

- If t is a variable, an abstraction, a weakening or a contraction, it is trivial.
- If $t = t'_1\langle x' = t'_2\rangle$, then we apply the induction hypothesis on t'_1:
  - If $t'_1 \cong \lambda x.t''$, then $t \cong (\lambda x.t'')\langle x' = t'_2\rangle \cong_{Abs} \lambda x.t''\langle x' = t'_2\rangle$.
  - If $t'_1 \cong W_y(t'')$, then $t \cong W_y(t'')\langle x' = t'_2\rangle \cong_{Weak2} W_y(t''\langle x' = t'_2\rangle)$.
  - If $t'_1 \cong C_x^{y,z}(t'')$, then $t \cong C_x^{y,z}(t'')\langle x' = t'_2\rangle \cong_{Cont2} C_x^{y,z}(t''\langle x' = t'_2\rangle)$.
  - If $t'_1 \cong (xt_1 \dots t_n)\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$, then $t \cong (xt_1 \dots t_n)\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle\langle x' = t'_2\rangle$.
  - If $t'_1 \cong W_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ with $x_i \notin FV(t'')$ for all i, then $t \cong W_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle\langle x' = t'_2\rangle$. Now either $x' \notin FV(t'')$ and we are done, or $x' \in FV(t'')$ and in that case $t \cong_{Weak2} W_x(t''\langle x' = t'_2\rangle)\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ since x_i is not free in t'_2 by α-conversion.
  - If $t'_1 \cong C_x^{y,z}(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ with $x_i \notin FV(t'')$ for all i, then $t \cong C_x^{y,z}(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle\langle x' = t'_2\rangle$. Now either $x' \notin FV(t'')$ and we are done, or $x' \in FV(t'')$ and in that case $t \cong_{Cont2} C_x^{y,z}(t''\langle x' = t'_2\rangle)\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ since x_i is not free in t'_2 by α-conversion.

– If $t = t'_1\ t'_2$, then we apply the induction hypothesis on t'_1.
- If $t'_1 \cong \lambda x.t''$, then $t \cong_B t''\langle x = t'_2\rangle$, on which we apply the induction hypothesis because the complexity is strictly smaller.
- If $t'_1 \cong W_y(t'')$, then $t \cong W_y(t'')\ t'_2 \cong_{WApp1} W_y(t''\ t'_2)$.
- If $t'_1 \cong C^{y,z}_x(t'')$, then $t \cong C^{y,z}_x(t'')\ t'_2 \cong_{CApp1} C^{y,z}_x(t''\ t'_2)$.
- If $t'_1 \cong (xt_1 \dots t_n)\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$, then $t \cong (xt_1 \dots t_n)\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle\ t'_2 \cong_{App1} (xt_1 \dots t_n t'_2)\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$.
- If $t'_1 \cong W_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ with $x_i \notin FV(t'')$ for all i, then $t \cong W_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle\ t'_2 \cong_{App1,WApp1} W_x(t''\ t'_2)\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$.
- If $t'_1 \cong C^{y,z}_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$ with $x_i \notin FV(t'')$ for all i, then $t \cong C^{y,z}_x(t'')\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle\ t'_2 \cong_{App1,CApp1} C^{y,z}_x(t''\ t'_2)\langle x = u\rangle\langle x_1 = u_1\rangle \dots \langle x_m = u_m\rangle$.

Now we establish some preliminary definitions concerning proof nets.

A proof net can be viewed as a finite acyclic oriented graph, where the nodes correspond to the following alphabet $\{ax, cut, \otimes, \parr, C, D, W, B\}$. Each node has a number of **p** *(premise)* and **c** *(conclusion)* ports. Indeed, the *ax* node has two ports **c**, the *cut* node two ports **p**, the D node one port **p** and one port **c**, the $\otimes$, $\parr$ and C nodes two ports **p** and one port **c**, the W node one port **c** and the B node has n ports **p** and n ports **c** (for every $n \geq 1$). Each edge in the graph is decorated with a *type* and is connected to exactly one port **c** and at most one port **p**, i.e. each edge has a *source* node but not necessarily a *goal* node [2].

We say that a node n is *final* in a proof-net iff there exists an edge connected to a **c** port of n which is not premise of another node. In other words, a node is final is some **c** port of n is "free". Remark that *cut* is never final. A type is said to be *distinguished* in a proof-net iff it is not a ? formula and it is the type of an edge connected to a **c** port of a final node. Every edge decorated with a ? formula which is connected to a **c** port of a final node n is labelled with an associated variable x. By abuse of language we will sometimes talk about the variable label of a final node having only one **c** port (as for example in the case of $W/C/\parr$-nodes).

Given a proof net r and a final node n of r we define $r \setminus n$ as the proof-net obtained from r by erasing the node n and its corresponding edges.

Remark that this notion is not well-adapted to proof nets modulo the congruence E so that we modify the intuitive notion of final node as follows:

A node W or C is *final-modulo* in a proof net r it is final in $pullB(r)$, where $pullB(r)$ is the proof net r where all the weakening and contraction nodes were pulled out boxes by using the *pull* reduction rule and the axiom B oriented as

[2] Formally, this is not an oriented graph but one can add an *empty* special node with one port **p** and none port **c** as destination of such edges.

a reduction rule from left to right. Finally, a node different from W and C is *final-modulo* if it is final.

A node n occurs at *depth* n in a proof-net r if n appears inside n different boxes of r.

Remark 2. Let t be a λlxr-term. Then the proof-net $T(\Gamma \vdash t : A)$ has a unique distinguished type which is A^*. As a consequence, $\parr$-nodes which are final-modulo in such proof-nets are unique and the edge connected to its conclusion port is decorated with the unique distinguished type of the proof-net.

Lemma 7. *Let t be a λlxr-term and let f be a final-modulo node of $NT(t)$.*

1. *If f is a W-node whose* **c** *edge is labelled with x, then there exist a term t' such that $t \cong W_x(t')$ and $|t| = |W_x(t')|$.*
2. *If f is a C-node whose* **c** *edge is labelled with x, then there exist a term t' such that $t \cong C_x^{y,z}(t')$ and $|t| = |C_x^{y,z}(t')|$.*
3. *If f is an $\parr$-node, then there exist a term t' and a variable x such that $t \cong \lambda x.t'$ and $|t| = |\lambda x.t'|$.*

Proof. By induction on $|t|$. We start by using Lemma 6 and thus assume t to be of a particular shape (without increasing its complexity).

- If $t \cong \lambda y.u$, then if f is a final-modulo $\parr$-node of $NT(t)$, statement 3 trivially holds. Otherwise, $NT(u)$ has the same final-modulo node f and statements 1 or 2 hold by i.h.
  In the first case, we have $u \cong W_x(u')$ and $|u| = |W_x(u')|$ by i.h.(1) so that $t \cong \lambda y.W_x(u')$ and $|t| = |\lambda y.W_x(u')|$. If $x \neq y$, then $\lambda y.W_x(u') \cong_{WAbs} W_x(\lambda y.u')$ and we are done. Otherwise, the W-node f is not final-modulo in $NT(t)$ which leads to a contradiction.
  In the second case, we have $u \cong C_x^{y,z}(u')$ and $|u| = |C_x^{y,z}(u')|$ by i.h.(2) so that $t \cong \lambda w.C_x^{y,z}(u')$ and $|t| = |\lambda w.C_x^{y,z}(u')|$. If $x \neq w$, then $\lambda w.C_x^{y,z}(u') \cong_{CAbs} C_x^{y,z}(\lambda x.u')$ and we are done. Otherwise, the C-node f is not final-modulo in $NT(t)$ which leads to a contradiction.
- If $t \cong W_y(u)$, then if f is a W-node labelled with $x = y$ the statement 1 trivially holds. Otherwise, $x \neq y$ and f is also final-modulo in $NT(u)$ and one of the statements hold by i.h.
  In the first case, we have $u \cong W_x(u')$ and $|u| = |W_x(u')|$ by i.h.(1), thus, $W_y(u) \cong W_y(W_x(u')) \cong_{C_w} W_x(W_y(u'))$ which concludes this case.
  In the second case, we have $u \cong C_x^{y,z}(u')$ and $|u| = |C_x^{y,z}(u')|$ by i.h.(2), thus $t \cong W_y(C_x^{w,z}(u'))$. Since w and z are bound variables we know that $y \neq w, z$. Therefore $W_y(C_x^{w,z}(u')) \cong_{Cross} C_x^{w,z}(W_y(u'))$ and we are done.
  In the third case, we have $u \cong \lambda x.u'$ and $|u| = |\lambda x.u'|$ by i.h.(3), thus $t = W_y(u) \cong W_y(\lambda x.u') \cong_{WAbs} \lambda x.W_y(u')$ and we are done.

- If $t \cong C^{y',z'}_{w}(u)$, then if f is a C-node labelled with $x = w$ the statement 2 trivially holds. Otherwise, f is already final-modulo in $NT(u)$ and one of the statements hold by i.h. We have to consider the following three different cases. In the first case, we have $u \cong W_x(u')$ and $|u| = |W_x(u')|$ by i.h.(1) so that $t \cong C^{y',z'}_{w}(W_x(u'))$. If $x \neq y', z'$, then $C^{y',z'}_{w}(W_x(u')) \cong_{Cross} W_x(C^{y',z'}_{w}(u'))$ and we are done. Otherwise, the W-node f is not final-modulo in $NT(t)$ which leads to a contradiction.
  In the second case, we have $u \cong C^{y,z}_{x}(u')$ and $|u| = |C^{y,z}_{x}(u')|$ by i.h.(2), thus $C^{y',z'}_{x'}(C^{y,z}_{x}(u')) \cong_{C1_c,C2_c} C^{y,z}_{x}(C^{y',z'}_{x'}(u'))$ and we are done.
  In the third case, we have by i.h.(3) $u \cong \lambda w.u'$ and $|u| = |\lambda w.u'|$, thus $t = C^{y,z}_{x}(u) \cong C^{y,z}_{x}(\lambda w.u') \cong_{CAbs} \lambda w.C^{y,z}_{x}(u')$ and we are done.
- Suppose $t \cong (xt_1 \ldots t_n)\langle x_{n+1} = t_{n+1}\rangle \ldots \langle x_m = t_m\rangle$.
  We know that if f is a final-modulo $W/C/\invamp$ node of $NT(t)$, then either it was already in $T(t)$ or it was created by a TB-reduction. But the only TB-reductions that can be applied on $T(t)$ are either the Ax-cut of the sequence of applications, which do not create new final-modulo nodes, or reductions inside a $Box(T(t_i))$.
  As a consequence, f cannot be a $\invamp$-node, and it is also final-modulo in some $NT(t_j)$.
  Hence, we apply the induction hypothesis on t_j to get $t_j \cong W_x(t'_j)$ (resp. $t_j \cong C^{y,z}_{x}(t'_j)$) with the same complexity as t_j. Let $t'_i = t_i$ for all $i \neq j$. If $j \leq n$, we use rules $WApp2, WApp1, Weak2$ (resp. $CApp2, CApp1, Cont2$) to get $t \cong W_x((xt'_1 \ldots t'_n)\langle x_{n+1} = t'_{n+1}\rangle \ldots \langle x_m = t'_m\rangle)$ (resp. $t \cong C^{y,z}_{x}((xt'_1 \ldots t'_n)\langle x_{n+1} = t'_{n+1}\rangle \ldots \langle x_m = t'_m\rangle)$). Otherwise, we use we use rules $WSubs, Weak2$ (resp. $CSubs, Cont2$).
- Suppose $t \cong W_v(u)\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle$ with $x_i \notin FV(u)$ for all i.
  Again, the only TB-reductions that can be applied on $T(t)$ are either inside $T(u)$ or inside $Box(T(u_i))$ for some i.
  As a consequence, f can only be a $\invamp$-node if it is final-modulo in $NT(u)$, otherwise it is final-modulo in some $NT(u_j)$.
  In the former case we use the induction hypothesis on u to get $u \cong \lambda x.u'$ (resp. $u \cong W_x(u')$ or $u \cong C^{y,z}_{x}(u')$). Then we use rules $WAbs, Abs$ (resp. $Weak2$ or $Cross, Cont2$) to get $t \cong \lambda x.(W_v(u')\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle)$ (resp. $t \cong W_x(W_v(u')\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle)$ or $t \cong C^{y,z}_{x}(W_v(u')\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle))$.
  In the latter case we use the induction hypothesis on u_j to get $u_j \cong W_x(u'_j)$ (resp. $u_j = C^{y,z}_{x}(u'_j)$). Let $u'_i = u_i$ for all $i \neq j$. We use rules $WSubs, Weak2$ (resp. $CSubs, Cont2$), and we get $t \cong W_x(W_v(u)\langle v = u'_0\rangle\langle x_1 = u'_1\rangle \ldots \langle x_m = u'_m\rangle)$ (resp. $t \cong C^{y,z}_{x}(W_v(u)\langle v = u'_0\rangle\langle x_1 = u'_1\rangle \ldots \langle x_m = u'_m\rangle))$.
- Suppose $t \cong C^{v_1,v_2}_{v}(u)\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle$ with $x_i \notin FV(u)$ for all i.
  Again, the only TB-reductions that can be applied on $T(t)$ are either inside $T(u)$ or inside a $Box(T(u_i))$.

As a consequence, f can only be a $\parr$-node if it is final-modulo in $NT(u)$, otherwise it is final-modulo in some $NT(u_j)$.
In the former case we use the induction hypothesis on u to get $u \cong \lambda x.u'$ (resp. $u \cong W_x(u')$ or $u \cong C_x^{y,z}(u')$). Then we use rules $CAbs, Abs$ (resp. $Cross, Weak2$ or $Cont2$) to get $t \cong \lambda x.(C_v^{v_1,v_2}(u')\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle)$ (resp. $t \cong W_x(C_v^{v_1,v_2}(u')\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle)$ or $t \cong C_x^{y,z}(C_v^{v_1,v_2}(u')\langle v = u_0\rangle\langle x_1 = u_1\rangle \ldots \langle x_m = u_m\rangle))$.
In the latter case we use the induction hypothesis on u_j to get $u_j \cong W_x(u'_j)$ (resp. $u_j = C_x^{y,z}(u'_j)$). Let $u'_i = u_i$ for all $i \neq j$. We use rules $WSubs, Weak2$ (resp. $CSubs, Cont2$), and we get $t \cong W_x(C_v^{v_1,v_2}(u)\langle v = u'_0\rangle\langle x_1 = u'_1\rangle \ldots \langle x_m = u'_m\rangle)$ (resp. $t \cong C_x^{y,z}(C_v^{v_1,v_2}(u)\langle v = u'_0\rangle\langle x_1 = u'_1\rangle \ldots \langle x_m = u'_m\rangle))$.

Lemma 8. *Let t be a λlxr-term. If $NT(t)$ has no final-modulo node C, W or $\parr$ but has some cut node at depth 0, then there exist terms v and u such that $t \cong (\lambda x.u)v$.*

Proof. By induction on the size of the term t.

- If $t = x$, then there is no *cut*-node at depth 0 in $NT(x)$.
- If $t = \lambda x.u$, then $NT(t)$ has a final-modulo $\parr$-node.
- If $t = W_y(u)$, then $NT(t)$ has a final-modulo W-node.
- If $t = C_x^{y,z}(u)$, then $NT(t)$ has a C final-modulo node.
- If t is an application or a closure, then $t \cong_B u_0u_1 \ldots u_n$ $(n \geq 1)$, where u_0 is neither an application nor a closure.
  - If $u_0 = y$, then there is no *cut*-node at depth 0 in $NT(t)$.
  - If $u_0 = W_y(w)$, then $NT(t)$ has a final-modulo W-node.
  - If $u_0 = C_w^{y,z}(t')$, then $NT(t)$ has a final-modulo C-node.
  - If $u_0 = (\lambda y.w)$, then we have

$$\begin{aligned} t &\cong_B (\lambda y.w)u_1u_2 \ldots u_n \\ &\cong_B w\langle y = u_1\rangle u_2 \ldots u_n \\ &\cong_{App1} (wu_2 \ldots u_n)\langle y = u_1\rangle \\ &\cong_B (\lambda y.wu_2 \ldots u_n)u_1 \end{aligned}$$

We can now state one of the main result of this paper.

Theorem 6 (Completeness). *Let t_1, t_2 be two λlxr-terms. If $T(t_1) \approx T(t_2)$, then $t_1 \cong t_2$.*

Proof. By induction on $NT(t_1)$.

- If $NT(t_1)$ has a final-modulo W-node called n, then $t_1 \cong W_x(t'_1)$ and $t_2 \cong W_x(t'_2)$ by Lemma 7. By definition of the translation $NT(t'_1) = NT(t_1) \setminus n$ and $NT(t'_2) = NT(t_2) \setminus n$. Thus $NT(t'_1) \sim_E NT(t'_2)$. By i.h. we have $t'_1 \cong t'_2$ so that $W_x(t'_1) \cong W_x(t'_2)$.

– If $NT(t_1)$ has a final-modulo C-node called n, then $t_1 \cong C^{y_1,z_1}_{x_1}(t'_1)$ and $t_2 \cong C^{y_2,z_2}_{x_2}(t'_2)$ by Lemma 7. By definition of the translation $NT(t'_1) = NT(t_1) \setminus n$ and $NT(t'_2) = NT(t_2) \setminus n$. Thus $NT(t'_1) \sim_E NT(t'_2)$. By i.h. we have $t'_1 \cong t'_2$ so that $C^{y_1,z_1}_{x_1}(t'_1) \cong C^{y_2,z_2}_{x_2}(t'_2)$.
– If $NT(t_1)$ has a final-modulo $\invamp$-node called n, then $t_1 \cong \lambda x.t'_1$ and $t_2 \cong \lambda x.t'_2$ by Lemma 7. By definition of the translation $NT(t'_1) = NT(t_1) \setminus n$ and $NT(t'_2) = NT(t_2) \setminus n$. Thus $NT(t'_1) \sim_E NT(t'_2)$. By i.h. we have $t'_1 \cong t'_2$ so that $\lambda x.t'_1 \cong \lambda x.t'_2$.
– If $NT(t_1)$ has no final-modulo $W/C/\invamp$-node, then t_1 is an application or a closure.
  If $NT(t_1)$ has no *cut*-node at depth 0, then $t_1 \cong xu_1 \dots u_n$ and $t_2 \cong yv_1 \dots v_n$. Since $NT(u_i) \sim_E NT(v_i)$, then by i.h. we have $u_i \cong v_i$ and thus $xu_1 \dots u_n \cong yv_1 \dots v_n$.
  If $NT(t_1)$ has some *cut*-node at depth 0, then by Lemma 8 we have $t_1 \cong (\lambda x.u_1)v_1$ and $t_2 \cong (\lambda x.u_2)v_2$. Since $NT(u_1)$ and $NT(u_2)$ are sub proof-nets of $NT(t_1)$ and $NT(t_2)$ respectively, then $u_1 \cong u_2$ by i.h. The same happens with $NT(v_1)$ and $NT(v_2)$ so that $v_1 \cong v_2$. We can then conclude $(\lambda x.u_1)v_1 \cong (\lambda x.u_2)v_2$.

A similar characterisation was given in [Lau03] for $\lambda\mu$-terms with respect to Polarized Proof Nets, where equality in the term syntax is an extension of the σ-equivalence on λ-terms defined in [Reg94]. Yet, the latter needs to consider specific permutations of β-redexs to achieve the characterisation, instead of simply turning some reduction rules into equivalence rules, which can be done in $\lambda\mathsf{lxr}$ only because the explicit operators of our calculus adequately reflect the structures of Proof Nets.

4 Recovering the λ-calculus

We show in this section the relation between $\lambda\mathsf{lxr}$-terms and λ-terms. More precisely, we show that the linearity constraints and the use of explicit resource operators in $\lambda\mathsf{lxr}$ are sufficient to decompose the β-reduction step into smaller steps. We will also show in this section the relation between typing in $\lambda\mathsf{lxr}$ and typing =changein λ-calculus.

For that we first recall in Figure 10 the typing rules for λ-calculus.

$$\frac{}{\Gamma, x : A \vdash_\lambda x : A} \qquad \frac{\Gamma, x : A \vdash_\lambda t : B}{\Gamma \vdash_\lambda \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash_\lambda t : A \to B \quad \Gamma \vdash_\lambda v : A}{\Gamma \vdash_\lambda t\ v : B}$$

Figure10. Typing Rules for λ-calculus

We shall always use the notation $\Gamma \vdash_\lambda t : A$ to denote typing judgements and typing derivability in λ-calculus in order to distinguish them from those of λlxr.

4.1 From λ-calculus to λlxr-calculus

We now describe how to encode a λ-term (possibly not linear) into a λlxr one. An important point is that

Definition 11. *The encoding of λ-terms is defined by induction as follows:*

$$
\begin{array}{lll}
\mathcal{A}(x) & := x & \\
\mathcal{A}(\lambda x.t) & := \lambda x.\mathcal{A}(t) & \textit{if } x \in \mathcal{FV}(t) \\
\mathcal{A}(\lambda x.t) & := \lambda x.W_x(\mathcal{A}(t)) & \textit{if } x \notin \mathcal{FV}(t) \\
\mathcal{A}(tu) & := C^{\Delta,\Pi}_{\Phi}(R^{\Phi}_{\Delta}(\mathcal{A}(t))\ R^{\Phi}_{\Pi}(\mathcal{A}(u))) & \textit{where } \Phi := \mathcal{FV}(t) \cap \mathcal{FV}(u)
\end{array}
$$

Note that $\mathcal{A}(tu) = \mathcal{A}(t)\mathcal{A}(u)$ in the particular case $\mathcal{FV}(t) \cap \mathcal{FV}(u) = \emptyset$. Also, using the fact that $W_\emptyset(t) = t$, we can write the translation of an abstraction by simply writing $\mathcal{A}(\lambda x.t) = \lambda x.W_{\{x\}\setminus\mathcal{FV}(t)}(\mathcal{A}(t))$.

In most of the following proofs, we shall use the following results:

Lemma 9 (Properties of $\mathcal{A}$).

1. $\mathcal{FV}(t) = \mathcal{FV}(\mathcal{A}(t))$.
2. $\mathcal{A}(R^{\Phi}_{\Delta}(t)) = R^{\Phi}_{\Delta}(\mathcal{A}(t))$

As a consequence, the encoding of a λ-term is a linear λlxr-term.

Example 1. Given $t = \lambda x.\lambda y.y(zz)$, we have $\mathcal{A}(t) = \lambda x.W_x(\lambda y.(y\ C^{z_1,z_2}_{z}(z_1\ z_2)))$.

We now want to simulate a β-reduction step in λlxr, so we start by proving that the reduction of an explicit substitution operator in xr does implement the notion of meta-substitution, generating the term substituted in the following sense:

Lemma 10. *For all λ-terms h and j such that $x \in \mathcal{FV}(h)$,*

$$C^{\Delta,\Pi}_{\Phi}(R^{\Phi}_{\Delta}(\mathcal{A}(h))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle) \longrightarrow^*_{\mathsf{xr}} \mathcal{A}(h\{x\backslash j\})$$

where $\Phi := (\mathcal{FV}(h) \setminus \{x\}) \cap \mathcal{FV}(j)$, provided that the former term is well-formed.

In the simple case where $\Phi = \emptyset$, the statement becomes:

$$\mathcal{A}(h)\langle x = \mathcal{A}(j)\rangle \longrightarrow^*_{\mathsf{xr}} \mathcal{A}(h\{x\backslash j\})$$

Proof. By induction on the size of h. We shall always suppose, by Barendregt's convention, that $x \notin \mathcal{FV}(j)$. Moreover, whenever we use the induction hypothesis throughout the proof, it will be applied to a term to which the former term reduces, so the hypothesis of well-formedness will be satisfied because well-formedness is preserved by reduction (Lemma 2, Property 1).

1. If h is a variable, then it must be x, so there is no contraction and

$$x\langle x = \mathcal{A}(j)\rangle \longrightarrow_{Var} \mathcal{A}(j) = \mathcal{A}(x\{x\backslash j\})$$

2. If $h = (t\ u)$, let
$\Sigma := \mathcal{FV}(j) \cap \mathcal{FV}(t) \cap \mathcal{FV}(u)$
$\Phi := \mathcal{FV}(j) \cap \mathcal{FV}(t) \setminus \Sigma$
$\Psi := \mathcal{FV}(j) \cap \mathcal{FV}(u) \setminus \Sigma$
$\Xi := \mathcal{FV}(t) \cap \mathcal{FV}(u) \setminus \Sigma$
$\Theta := \mathcal{FV}(j) \setminus \Phi$

Note in particular that $\Phi = \Sigma, \Lambda, \Psi$.
We now fix an order on Σ, Λ, Ψ, Ξ, Θ so that they can be treated as lists. We can in particular write $\mathcal{A}(h)$ as

$$C^{(\Sigma_3,\Xi_3),(\Sigma_4,\Xi_4)}_{\Sigma,\Xi}(R^{\Sigma,\Xi}_{\Sigma_3,\Xi_3}(\mathcal{A}(t))\ R^{\Sigma,\Xi}_{\Sigma_4,\Xi_4}(\mathcal{A}(u)))$$

We then have:

$$\begin{aligned} &C^{(\Sigma_1,\Lambda_1,\Psi_1),(\Sigma_2,\Lambda_2,\Psi_2)}_{\Sigma,\Lambda,\Psi}(R^{\Sigma,\Lambda,\Psi}_{\Sigma_1,\Lambda_1,\Psi_1}(\mathcal{A}(h))\langle x = R^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(j))\rangle) \\ = &C^{(\Sigma_1,\Lambda_1,\Psi_1),(\Sigma_2,\Lambda_2,\Psi_2)}_{\Sigma,\Lambda,\Psi}(C^{(\Sigma_3,\Xi_3),(\Sigma_4,\Xi_4)}_{\Sigma_1,\Xi}(t'\ u')\langle x = R^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(j))\rangle) \end{aligned}$$

where $t' = R^{\Lambda,\Sigma,\Xi}_{\Lambda_1,\Sigma_3,\Xi_3}(\mathcal{A}(t))$ and $u' = R^{\Psi,\Sigma,\Xi}_{\Psi_1,\Sigma_4,\Xi_4}(\mathcal{A}(u))$. We call this term H.

(a) If $x \in \mathcal{FV}(t) \cap \mathcal{FV}(u)$, then x is necessarily in Ξ (since $x \notin \mathcal{FV}(j)$), so that $\Xi := \Xi', x$. We note that $C^{(\Sigma_3,\Xi_3),(\Sigma_4,\Xi_4)}_{\Sigma_1,\Xi}()$ can be written as $C^{(\Sigma_3,\Xi'_3),(\Sigma_4,\Xi'_4)}_{\Sigma_1,\Xi'}(C^{x_3,x_4}_{x}())$ and thus, $\Xi'_3 := \Xi_3 \setminus \{x_3\}$ and $\Xi'_4 := \Xi_4 \setminus \{x_4\}$. The term H can be transformed by $Cont2$ and then by rule $Cont1$ to:

$$C^{(\Sigma_1,\Lambda_1,\Psi_1),(\Sigma_2,\Lambda_2,\Psi_2)}_{\Sigma,\Lambda,\Psi}(C^{(\Sigma_3,\Xi'_3),(\Sigma_4,\Xi'_4)}_{\Sigma_1,\Xi'}(C^{(\Theta_5,\Sigma_5,\Lambda_5,\Psi_5),(\Theta_6,\Sigma_6,\Lambda_6,\Psi_6)}_{\Theta,\Sigma_2,\Lambda_2,\Psi_2}(M)))$$

where M is

$$(t'\ u')\langle x_3 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(j))\rangle\langle x_4 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(j))\rangle$$

which can be reduced by $App1$ to

$$(t'\langle x_3 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(j))\rangle\ u')\langle x_4 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(j))\rangle$$

which can be reduced by $App2$ to

$$t'\langle x_3 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(j))\rangle\ u'\langle x_4 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(j))\rangle$$

which we call M'. Now we are going to rearrange the contractions:

$$
\begin{aligned}
& C^{(\Sigma_1,\Lambda_1,\Psi_1),(\Sigma_2,\Lambda_2,\Psi_2)}_{\Sigma,\Lambda,\Psi}(C^{(\Sigma_3,\Xi'_3),(\Sigma_4,\Xi'_4)}_{\Sigma_1,\Xi'}(C^{(\Theta_5,\Sigma_5,\Lambda_5,\Psi_5),(\Theta_6,\Sigma_6,\Lambda_6,\Psi_6)}_{\Theta,\Sigma_2,\Lambda_2,\Psi_2}(M'))) \\
\equiv\ & C^{\Theta_5,\Theta_6}_{\Theta}(C^{\Xi'_3,\Xi'_4}_{\Xi'}(C^{\Lambda_1,\Lambda_2}_{\Lambda}(C^{\Lambda_5,\Lambda_6}_{\Lambda_2}(C^{\Psi_1,\Psi_2}_{\Psi}(C^{\Psi_5,\Psi_6}_{\Psi_2}(N)))))) \\
& \qquad \text{where } N := C^{\Sigma_1,\Sigma_2}_{\Sigma}(C^{\Sigma_3,\Sigma_4}_{\Sigma_1}(C^{\Sigma_5,\Sigma_6}_{\Sigma_2}(M'))) \\
\equiv\ & C^{\Theta_5,\Theta_6}_{\Theta}(C^{\Xi'_3,\Xi'_4}_{\Xi'}(C^{\Lambda_2,\Lambda_6}_{\Lambda}(C^{\Lambda_1,\Lambda_5}_{\Lambda_2}(C^{\Psi_5,\Psi_2}_{\Psi}(C^{\Psi_1,\Psi_6}_{\Psi_2}(N)))))) \\
& \qquad \text{where } N := C^{\Sigma_1,\Sigma_2}_{\Sigma}(C^{\Sigma_3,\Sigma_5}_{\Sigma_1}(C^{\Sigma_4,\Sigma_6}_{\Sigma_2}(M'))) \\
\equiv\ & C^{(\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1),(\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2)}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(C^{(\Lambda_1,\Sigma_3),(\Lambda_5,\Sigma_5)}_{\Lambda_2,\Sigma_1}(C^{(\Psi_1,\Sigma_4),(\Psi_6,\Sigma_6)}_{\Psi_2,\Sigma_2}(M')))
\end{aligned}
$$

This term can be reduced by $CApp2$ and then by $CApp1$ to

$$H' := C^{(\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1),(\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2)}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(P\ Q)$$

where

$$
\begin{aligned}
P &:= C^{(\Lambda_1,\Sigma_3),(\Lambda_5,\Sigma_5)}_{\Lambda_2,\Sigma_1}(t'\langle x_3 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(j))\rangle) \\
&= R^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1}(C^{(\Lambda_1,\Sigma_3),(\Lambda_5,\Sigma_5)}_{\Lambda,\Sigma}(R^{\Lambda,\Sigma}_{\Lambda_1,\Sigma_3}(R^{x}_{x_3}(\mathcal{A}(t)))\langle x_3 = R^{\Sigma,\Lambda}_{\Sigma_5,\Lambda_5}(\mathcal{A}(j))\rangle))
\end{aligned}
$$

$$
\begin{aligned}
Q &:= C^{(\Psi_1,\Sigma_4),(\Psi_6,\Sigma_6)}_{\Psi_2,\Sigma_2}(u'\langle x_4 = R^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(j))\rangle) \\
&= R^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}(C^{(\Psi_1,\Sigma_4),(\Psi_6,\Sigma_6)}_{\Psi,\Sigma}(R^{\Psi,\Sigma}_{\Psi_1,\Sigma_4}(R^{x}_{x_4}(\mathcal{A}(u)))\langle x_4 = R^{\Sigma,\Psi}_{\Sigma_6,\Psi_6}(\mathcal{A}(j))\rangle))
\end{aligned}
$$

We can now apply the induction hypothesis to both subterms and we get:

$$
\begin{aligned}
P \longrightarrow^*_{\mathsf{xr}} P' &:= R^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1}(\mathcal{A}(t\{x\backslash j\})) \\
Q \longrightarrow^*_{\mathsf{xr}} Q' &:= R^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}(\mathcal{A}(u\{x\backslash j\}))
\end{aligned}
$$

So H' reduces to

$$C^{(\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1),(\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2)}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(P'\ Q')$$

which is $\mathcal{A}(t\{x\backslash j\}\ \ u\{x\backslash j\}) = \mathcal{A}((t\ u)\{x\backslash j\})$.

(b) If $x \in \mathcal{FV}(t)$ et $x \notin \mathcal{FV}(u)$, the term H can be transformed by $Cont2$ to:

$$C^{(\Sigma_1,\Lambda_1,\Psi_1),(\Sigma_2,\Lambda_2,\Psi_2)}_{\Sigma,\Lambda,\Psi}(C^{(\Sigma_3,\Xi_3),(\Sigma_4,\Xi_4)}_{\Sigma_1,\Xi}((t'\ u')\langle x = R^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(j))\rangle))$$

which can be reduced by $App1$ to

$$
\begin{aligned}
& C^{(\Sigma_1,\Lambda_1,\Psi_1),(\Sigma_2,\Lambda_2,\Psi_2)}_{\Sigma,\Lambda,\Psi}(C^{(\Sigma_3,\Xi_3),(\Sigma_4,\Xi_4)}_{\Sigma_1,\Xi}(t'\langle x = R^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(j))\rangle\ u')) \\
\equiv\ & C^{(\Sigma_1,\Psi_2,\Xi_3),(\Sigma_4,\Psi_1,\Xi_4)}_{\Sigma,\Psi,\Xi}(C^{(\Sigma_3,\Lambda_1),(\Sigma_2,\Lambda_2)}_{\Sigma_1,\Lambda}(t'\langle x = R^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(j))\rangle\ u'))
\end{aligned}
$$

which can be reduced by $CApp1$ to

$$
\begin{aligned}
&C^{(\Sigma_1,\Psi_2,\Xi_3),(\Sigma_4,\Psi_1,\Xi_4)}_{\Sigma,\Psi,\Xi}(C^{(\Sigma_3,\Lambda_1),(\Sigma_2,\Lambda_2)}_{\Sigma_1,\Lambda}(t'\langle x = R^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(j))\rangle)\ u')\\
=\ &C^{(\Sigma_1,\Psi_2,\Xi_3),(\Sigma_4,\Psi_1,\Xi_4)}_{\Sigma,\Psi,\Xi}(R^{\Sigma,\Psi,\Xi}_{\Sigma_1,\Psi_2,\Xi_3}(M)\ R^{\Sigma,\Psi,\Xi}_{\Sigma_4,\Psi_1,\Xi_4}(u))
\end{aligned}
$$

where $M := C^{(\Sigma_3,\Lambda_1),(\Sigma_2,\Lambda_2)}_{\Sigma,\Lambda}(R^{\Lambda,\Sigma}_{\Lambda_1,\Sigma_3}(\mathcal{A}(t))\langle x = R^{\Sigma,\Lambda}_{\Sigma_2,\Lambda_2}(\mathcal{A}(j))\rangle)$, which reduces, by induction hypothesis, to $\mathcal{A}(t\{x\backslash j\})$. Hence,

$$H \longrightarrow^*_{\mathsf{xr}} C^{(\Sigma_1,\Psi_2,\Xi_3),(\Sigma_4,\Psi_1,\Xi_4)}_{\Sigma,\Psi,\Xi}(R^{\Sigma,\Psi,\Xi}_{\Sigma_1,\Psi_2,\Xi_3}(\mathcal{A}(t\{x\backslash j\}))\ R^{\Sigma,\Psi,\Xi}_{\Sigma_4,\Psi_1,\Xi_4}(u))$$

which is exactly $\mathcal{A}(t\{x\backslash j\}\ \ u) = \mathcal{A}((t\ u)\{x\backslash j\})$.

(c) If $x \in \mathcal{FV}(t)$ et $x \notin \mathcal{FV}(u)$ the proof is exactly the same.

(d) If $x \notin \mathcal{FV}(t)$ et $x \notin \mathcal{FV}(u)$ then the first term is not well-formed.

3. If $h = \lambda u.v$ then by α-conversion we can suppose $u \neq x$ and $u \notin \mathcal{FV}(j)$.

(a) If $u \in \mathcal{FV}(v)$ then

$$
\begin{array}{ll}
& C^{\Delta,\Pi}_{\Phi}(R^{\Phi}_{\Delta}(\lambda u.\mathcal{A}(v))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle)\\
= & C^{\Delta,\Pi}_{\Phi}((\lambda u.R^{\Phi}_{\Delta}(\mathcal{A}(v)))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle)\\
\longrightarrow_{Abs} & C^{\Delta,\Pi}_{\Phi}(\lambda u.(R^{\Phi}_{\Delta}(\mathcal{A}(v))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle))\\
\longrightarrow_{CAbs} & \lambda u.C^{\Delta,\Pi}_{\Phi}(R^{\Phi}_{\Delta}(\mathcal{A}(v))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle)
\end{array}
$$

and we get the result by the induction hypothesis.

(b) If $u \notin \mathcal{FV}(v)$ then

$$
\begin{array}{ll}
& C^{\Delta,\Pi}_{\Phi}(R^{\Phi}_{\Delta}(\lambda u.W_u(\mathcal{A}(v)))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle)\\
= & C^{\Delta,\Pi}_{\Phi}((\lambda u.W_u(R^{\Phi}_{\Delta}(\mathcal{A}(v))))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle)\\
\longrightarrow_{Abs} & C^{\Delta,\Pi}_{\Phi}(\lambda u.(W_u(R^{\Phi}_{\Delta}(\mathcal{A}(v))))\langle x = R^{\Phi}_{\Pi}(\mathcal{A}(j))\rangle)\\
\longrightarrow_{Weak2} & C^{\Delta,\Pi}_{\Phi}(\lambda u.W_u(R^{\Phi}_{\Delta}(\mathcal{A}(v))\langle x = R^{\Gamma}_{\Pi}(\mathcal{A}(j))\rangle))\\
\longrightarrow_{CAbs} & \lambda u.C^{\Delta,\Pi}_{\Phi}(W_u(R^{\Gamma}_{\Delta}(\mathcal{A}(v))\langle x = R^{\Gamma}_{\Pi}(\mathcal{A}(j))\rangle))\\
\longrightarrow^*_{Cross} & \lambda u.W_u(C^{\Delta,\Pi}_{\Phi}(R^{\Gamma}_{\Delta}(\mathcal{A}(v))\langle x = R^{\Gamma}_{\Pi}(\mathcal{A}(j))\rangle))
\end{array}
$$

and we get the result by the induction hypothesis.

The *correctness* result concerning explicit substitutions obtained in the previous lemma enables us to prove a more general property concerning simulation of β-reduction in $\lambda\mathsf{lxr}$. Notice that a β-reduction step may not preserve the set of free variables whereas any reduction in $\lambda\mathsf{lxr}$ does. Indeed, we have $t = (\lambda x.y)\ z \longrightarrow_\beta\ y$, but

$$\mathcal{A}(t) = (\lambda x.W_x(y))\ z \longrightarrow^*_{\lambda\mathsf{lxr}} W_z(y) = W_z(\mathcal{A}(y))$$

As a consequence, the simulation property has to be stated very carefully.

Theorem 7 (Simulating β-reduction). *Let t be a λ-term such that $t \longrightarrow_\beta\ t'$. Then $\mathcal{A}(t) \longrightarrow^+_{\lambda\mathsf{lxr}} W_{\mathcal{FV}(t)\backslash\mathcal{FV}(t')}(\mathcal{A}(t'))$.*

Proof. We prove this by structural induction on terms. We only show here the root reduction cases.

1. The root case is the reduction $(\lambda x.M)N \longrightarrow_\beta \; M\{x\backslash N\}$. By α-conversion, $x \notin \mathcal{FV}(N)$.
   (a) If $x \notin \mathcal{FV}(M)$,
   let $\Phi := \mathcal{FV}(M) \cap \mathcal{FV}(N)$ and $\Xi := \mathcal{FV}(N) \setminus \mathcal{FV}(M)$.

$$\begin{array}{lll}\mathcal{A}((\lambda x.M)N) = & & C_\Phi^{\Delta,\Pi}(\lambda x.W_x(R_\Delta^\Phi(\mathcal{A}(M)))\ R_\Pi^\Phi(\mathcal{A}(N))) \\ & \longrightarrow_B & C_\Phi^{\Delta,\Pi}(W_x(R_\Delta^\Phi(\mathcal{A}(M)))\langle x = R_\Pi^\Phi(\mathcal{A}(N))\rangle) \\ & \longrightarrow_{Weak1} & C_\Phi^{\Delta,\Pi}(W_{\mathcal{FV}(R_\Pi^\Phi(N))}(R_\Delta^\Phi(\mathcal{A}(M)))) \\ & \equiv & C_\Phi^{\Delta,\Pi}(W_{\Pi,\Xi}(R_\Delta^\Phi(\mathcal{A}(M)))) \\ & \equiv & C_\Phi^{\Delta,\Pi}(W_\Pi(W_\Xi(R_\Delta^\Phi(\mathcal{A}(M))))) \\ & \longrightarrow^*_{Merge} & R_\Gamma^\Delta(W_\Xi(R_\Delta^\Phi(\mathcal{A}(M)))) \\ & \equiv & W_\Xi(\mathcal{A}(M))\end{array}$$

   Now it suffices to notice that
   $\Xi := \mathcal{FV}((\lambda x.M)\ N) \setminus \mathcal{FV}(M\{x\backslash N\})$
   (b) If $x \in \mathcal{FV}(M)$,
   let $\Phi := (\mathcal{FV}(M) \setminus \{x\}) \cap \mathcal{FV}(N)$.

$$\begin{array}{lll}\mathcal{A}((\lambda x.M)N) = & & C_\Phi^{\Delta,\Pi}(\lambda x.R_\Delta^\Phi(\mathcal{A}(M))\ R_\Pi^\Phi(\mathcal{A}(N))) \\ & \longrightarrow_B & C_\Phi^{\Delta,\Pi}(\mathcal{A}(R_\Delta^\Phi(M))\langle x = \mathcal{A}(R_\Pi^\Phi(N))\rangle) \\ & \longrightarrow^*_{\mathsf{xr}} & \mathcal{A}(M\{x\backslash N\}) \qquad \text{by Lemma 10}\end{array}$$

2. Now suppose $\lambda x.M \longrightarrow_\beta \; \lambda x.M'$,
   (a) If $x \in \mathcal{FV}(M)$

$$\begin{array}{lll}\mathcal{A}(\lambda x.M) = & & \lambda x.\mathcal{A}(M) \\ & \longrightarrow^+_{\lambda\mathsf{lxr}} & \lambda x.W_{\mathcal{FV}(M)\setminus\mathcal{FV}(M')}(\mathcal{A}(M')) \qquad \text{by induction} \\ & = & \lambda x.W_{\mathcal{FV}(\lambda x.M)\setminus\mathcal{FV}(M')}(W_{\{x\}\setminus\mathcal{FV}(M')}(\mathcal{A}(M'))) \\ & = & \lambda x.W_{\mathcal{FV}(\lambda x.M)\setminus\mathcal{FV}(\lambda x.M')}(W_{\{x\}\setminus\mathcal{FV}(M')}(\mathcal{A}(M'))) \\ & \longrightarrow^*_{WAbs} & W_{\mathcal{FV}(\lambda x.M)\setminus\mathcal{FV}(\lambda x.M')}(\lambda x.W_{\{x\}\setminus\mathcal{FV}(M')}(\mathcal{A}(M')))\end{array}$$

   (b) If $x \notin \mathcal{FV}(M)$

$$\begin{array}{lll}\mathcal{A}(\lambda x.M) = & & \lambda x.W_x(\mathcal{A}(M)) \\ & \longrightarrow^+_{\lambda\mathsf{lxr}} & \lambda x.W_x(W_{\mathcal{FV}(M)\setminus\mathcal{FV}(M')}(\mathcal{A}(M'))) \\ & = & \lambda x.W_x(W_{\mathcal{FV}(\lambda x.M)\setminus\mathcal{FV}(\lambda x.M')}(\mathcal{A}(M'))) \\ & \longrightarrow^*_{WAbs} & W_{\mathcal{FV}(\lambda x.M)\setminus\mathcal{FV}(\lambda x.M')}(\lambda x.W_x(\mathcal{A}(M')))\end{array}$$

3. Now suppose $M\ N \longrightarrow_\beta \; M'\ N$, let
   $\Phi := \mathcal{FV}(M) \cap \mathcal{FV}(N)$
   $\Sigma := \mathcal{FV}(M') \cap \mathcal{FV}(N)$
   $\Lambda := \mathcal{FV}(M') \setminus \Sigma$

$\Psi := (\mathcal{FV}(M) \cap \mathcal{FV}(N)) \setminus \mathcal{FV}(M')$
$\Xi := (\mathcal{FV}(M) \setminus \Psi) \setminus \mathcal{FV}(M')$
Note in particular that $\Phi = \Sigma, \Psi$.
Correspondingly, we split Δ and Π: $\Delta = \Sigma_l, \Psi_l$ and $\Pi = \Sigma_r, \Psi_r$.
We have:

$$\begin{array}{lll}
\mathcal{A}(M\ N) = & C_\Phi^{\Delta,\Pi}(R_\Delta^\Phi(\mathcal{A}(M))\ R_\Pi^\Phi(\mathcal{A}(N))) & \\
\longrightarrow^+{}_{\lambda\mathsf{lxr}} & C_\Phi^{\Delta,\Pi}(R_\Delta^\Phi(W_{\mathcal{FV}(M)\setminus\mathcal{FV}(M')}(\mathcal{A}(M')))\ R_\Pi^\Phi(\mathcal{A}(N))) & \\
 & \text{by induction hypothesis} & \\
= & C_\Phi^{\Delta,\Pi}(R_{\Sigma_l,\Psi_l}^{\Sigma,\Psi}(W_{\Psi,\Xi}(\mathcal{A}(M')))\ R_{\Sigma_r,\Psi_r}^{\Sigma,\Psi}(\mathcal{A}(N))) & \\
= & C_\Phi^{\Delta,\Pi}(W_\Xi(W_{\Psi_l}(R_{\Sigma_l}^\Sigma(\mathcal{A}(M'))))\ R_{\Sigma_r,\Psi_r}^{\Sigma,\Psi}(\mathcal{A}(N))) & \\
\longrightarrow^*{}_{WApp1} & C_\Phi^{\Delta,\Pi}(W_\Xi(W_{\Psi_l}(R_{\Sigma_l}^\Sigma(\mathcal{A}(M'))\ R_{\Sigma_r,\Psi_r}^{\Sigma,\Psi}(\mathcal{A}(N))))) & \\
\longrightarrow^*{}_{Cross} & W_\Xi(C_\Phi^{\Delta,\Pi}(W_{\Psi_l}(R_{\Sigma_l}^\Sigma(\mathcal{A}(M'))\ R_{\Sigma_r,\Psi_r}^{\Sigma,\Psi}(\mathcal{A}(N))))) & \\
= & W_\Xi(C_{\Sigma,\Psi}^{(\Sigma_l,\Psi_l),(\Sigma_r,\Psi_r)}(W_{\Psi_l}(R_{\Sigma_l}^\Sigma(\mathcal{A}(M'))\ R_{\Sigma_r,\Psi_r}^{\Sigma,\Psi}(\mathcal{A}(N))))) & \\
\longrightarrow^*{}_{Merge} & W_\Xi(C_\Sigma^{\Sigma_l,\Sigma_r}(R_\Psi^{\Psi_r}(R_{\Sigma_l}^\Sigma(\mathcal{A}(M'))\ R_{\Sigma_r,\Psi_r}^{\Sigma,\Psi}(\mathcal{A}(N))))) & \\
= & W_\Xi(C_\Sigma^{\Sigma_l,\Sigma_r}(R_{\Sigma_l}^\Sigma(\mathcal{A}(M'))\ R_{\Sigma_r}^\Sigma(\mathcal{A}(N)))) &
\end{array}$$

Then it suffices to notice that $\Xi = \mathcal{FV}(M\ N) \setminus \mathcal{FV}(M'\ N)$.

4. The case $M\ N \longrightarrow_\beta\ M\ N'$ is exactly the same.

As for the types, a straightforward induction on typing derivations allows us to show the soundness of the translation $\mathcal{A}$:

Lemma 11 (Encoding $\mathcal{A}$ preserves types). *If t is a λ-term s.t. $\Gamma \vdash_\lambda t : A$, then $\Gamma \vdash W_{\Gamma\setminus\mathcal{FV}(t)}(\mathcal{A}(t)) : A$.*

4.2 From λlxr-calculus to λ-calculus

We now show how to encode a λlxr-term into a λ-term.

Definition 12. *Let t be a λlxr-term. We define the function $\mathcal{B}(t)$ by induction on the structure of t as follows:*

$$\begin{array}{ll}
\mathcal{B}(x) & = x \\
\mathcal{B}(\lambda x.t) & = \lambda x.\mathcal{B}(t) \\
\mathcal{B}(W_x(t)) & = \mathcal{B}(t) \\
\mathcal{B}(C_x^{y,z}(t)) & = \mathcal{B}(t)\{y\backslash x\}\{z\backslash x\} \\
\mathcal{B}(t\ u) & = \mathcal{B}(t)\mathcal{B}(u) \\
\mathcal{B}(t\langle x = u\rangle) & = \mathcal{B}(t)\{x\backslash\mathcal{B}(u)\}
\end{array}$$

The translation $\mathcal{B}$ enjoys the following properties.

Remark 3 (Properties of $\mathcal{B}$).

- $\mathcal{B}(R_\Delta^\Phi(t)) = R_\Delta^\Phi(\mathcal{B}(t))$
- $\mathcal{FV}(\mathcal{B}(t)) \subseteq \mathcal{FV}(t)$

The following result will allow us to establish a clean relation between λlxr and λ-calculi.

Lemma 12 (Simulating λlxr-reduction).

1. *If $M \equiv N$, then $\mathcal{B}(M) = \mathcal{B}(N)$.*
2. *If $M \longrightarrow_B N$, then $\mathcal{B}(M) \longrightarrow^*_\beta \mathcal{B}(N)$.*
3. *If $M \longrightarrow_{\mathsf{xr}} N$, then $\mathcal{B}(M) = \mathcal{B}(N)$.*

Proof. 1. This is obvious for the equivalence rule C_w. For the other ones we have to use the well-known [Bar84] substitution lemma of λ-calculus stating that for any λ-terms t, u, v,

$$t\{x\backslash u\}\{y\backslash v\} = t\{y\backslash v\}\{x\backslash u\{y\backslash v\}\}$$

2. A B-reduction step at the root of M corresponds exactly to a β-reduction step at the root of $\mathcal{B}(M)$. For the closure under contexts, all cases are trivial except for:
   - the contraction, for which we use the fact that if $\mathcal{B}(t) \longrightarrow^*_\beta \mathcal{B}(t')$ then $\mathcal{B}(t)\{y\backslash x\}\{z\backslash x\} \longrightarrow^*_\beta \mathcal{B}(t')\{y\backslash x\}\{z\backslash x\}$.
   - the substitution, for which we use the two following facts:
     If $\mathcal{B}(t) \longrightarrow^*_\beta \mathcal{B}(t')$ then $\mathcal{B}(t)\{x\backslash \mathcal{B}(u)\} \longrightarrow^*_\beta \mathcal{B}(t')\{x\backslash \mathcal{B}(u)\}$.
     If $\mathcal{B}(u) \longrightarrow^*_\beta \mathcal{B}(u')$ then $\mathcal{B}(t)\{x\backslash \mathcal{B}(u)\} \longrightarrow^*_\beta \mathcal{B}(t)\{x\backslash \mathcal{B}(u')\}$.
3. We only discuss the cases where the reduction takes place at the root, all the other ones being trivial.
   - If the rule applied is $WAbs, WApp1, WApp2, WSubs, Cross, Weak2$, then the property is trivial.
   - If the rule applied is $Abs, App1, App2, Var, CAbs, CApp1, CApp2$, then the property follows from the definition of implicit substitution.
   - If the rule applied is $Comp$, then x is not free in t since the left-hand side is well formed, so by Remark 3 x is neither free in $\mathcal{B}(t)$. It suffices to use the substitution lemma as before.
   - If the rule applied is $Weak1$, then x is not free in t since the left-hand side is well formed, so by Remark 3 x is neither free in $\mathcal{B}(t)$. Hence, we get on the left-hand side $\mathcal{B}(t)\{x\backslash \mathcal{B}(u)\}$ which is exactly $\mathcal{B}(t)$.
   - If the rule applied is $Merge$, then, as before, y is not free in $\mathcal{B}(t)$ so that it suffices to notice that $\mathcal{B}(t)\{z\backslash w\} = \mathcal{B}(R^z_w(t))$ by Remark 3.
   - If the rule applies is $CSubs$, then it is sufficient to apply the substitution lemma of λ-calculus.
   - If the rule applied is $Cont1$, then, as before, x is not free in $\mathcal{B}(t)$ so that $\mathcal{B}(M) = \mathcal{B}(t)\{y\backslash \mathcal{B}(u)\}\{z\backslash \mathcal{B}(u)\}$ by the substitution lemma. For the right-hand side we have

$$\mathcal{B}(N) = \mathcal{B}(t)\{y\backslash \mathcal{B}(R^{\Gamma}_{\Delta}(u))\}\{z\backslash \mathcal{B}(R^{\Gamma}_{\Pi}(u))\}\{\Delta\backslash\Gamma\}\{\Pi\backslash\Gamma\}$$

which, using Remark 3, is equal to

$$\mathcal{B}(t)\{y\backslash R^{\Gamma}_{\Delta}(\mathcal{B}(u))\}\{z\backslash R^{\Gamma}_{\Pi}(\mathcal{B}(u))\}\{\Delta\backslash\Gamma\}\{\Pi\backslash\Gamma\}$$

which is

$$R^{\Pi}_{\Gamma}(R^{\Delta}_{\Gamma}(\mathcal{B}(t)\{y\backslash R^{\Gamma}_{\Delta}(\mathcal{B}(u))\}\{z\backslash R^{\Gamma}_{\Pi}(\mathcal{B}(u))\}))$$

which is equal to the left-hand side.

Corollary 1. *If $M \longrightarrow_{\lambda\mathsf{lxr}} N$, then $\mathcal{B}(M) \longrightarrow^*_{\beta} \mathcal{B}(N)$.*

Remark 4 (Composition of Encodings). Since congruent terms are mapped to the same λ-term, it makes sense to consider $\mathcal{B}(\mathcal{A}(\_))$, which is in fact the identity: $t =_{\alpha} \mathcal{B}(\mathcal{A}(t))$ (straightforward induction on t).

A straightforward induction on typing derivations allows us to show:

Lemma 13 ($\mathcal{B}$ preserves types). *If t is a $\lambda\mathsf{lxr}$-term such that $\Gamma \vdash t : A$, then $\Gamma \vdash_{\lambda} \mathcal{B}(t) : A$.*

5 Operational Properties

In Sections 2, 3 and 4 we have already established the properties of subject reduction, strong normalisation of well-typed $\lambda\mathsf{lxr}$-terms and simulation of β-reduction step by step. But a calculus which is defined in order to implement λ-calculus is also expected to satisfy confluence and preservation of strong normalisation (PSN). We prove both properties in this section.

5.1 Preservation of Strong Normalisation

PSN is in some sense a *test* property when a new calculus with explicit operators is proposed. As mentioned in the introduction, Melliès [Mel95] has shown that calculi such as $\lambda\sigma$ [ACCL91] and $\lambda\sigma_{\Uparrow}$ [HL89] do not have PSN, that is, there are β-strongly normalisable terms in λ-calculus which are not strongly normalisable when evaluated with the reduction rules of $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$.

The original notion of PSN [BBLRD96] makes sense in any calculus extending the syntax of the λ-calculus. Since we work with linear terms, it is not the case of $\lambda\mathsf{lxr}$, so the notion of PSN has to be properly reformulated in our context as follows: every strongly normalisable λ-term *is encoded* into a strongly normalisable $\lambda\mathsf{lxr}$-term, the encoding being in our case $\mathcal{A}(\_)$ (Definition 11).

We establish PSN of $\lambda\mathsf{lxr}$ by simulating reductions in $\lambda\mathsf{lxr}$ by reductions in the λI-calculus of Church-Klop [Klo80] with its associated reduction relations β and π. We refer the reader to [Sor97,Xi97] for a survey on different techniques based on the λI-calculus to infer normalisation properties. The proof technique that we use here is presented in [Len05], which establishes general results, but also fully treats the case of $\lambda\mathsf{lxr}$ as an example. We give here the part that is specific to $\lambda\mathsf{lxr}$ and refer the reader to [Len05] for the proofs of the general results that we use. The technique can be summarised as follows:

1. Define a relation $\mathcal{I}$ between linear λlxr-terms and λI-terms (Definition 14).
2. Show that $t \,\mathcal{I}\, T$ and $t \longrightarrow_{\mathsf{xr}} t'$ imply $t' \,\mathcal{I}\, T$, as well as $t \,\mathcal{I}\, T$ and $t \longrightarrow_B t'$ imply $\exists T'$ such that $t' \,\mathcal{I}\, T'$ and $T \longrightarrow^+_{\beta\pi} T'$ (Theorem 8).
3. Deduce that if $t \,\mathcal{I}\, T$ and $T \in SN_{\beta\pi}$, then $t \in SN_{\lambda\mathsf{lxr}}$ (Corollary 2).
4. Use an encoding $i() : \lambda \mapsto \lambda I$ (Definition 15) such that if $t \in SN_\beta$ then $i(t) \in SN_{\beta\pi}$ (Theorem 9).
5. Show that $\mathcal{A}(t) \,\mathcal{I}\, i(t)$ (Theorem 10), where $\mathcal{A}(t)$ is the encoding given in Section 4, and conclude PSN (Corollary 3).

We now proceed to develop the above points needed to conclude PSN as explained above.

Definition 13. *The set Λ_I of terms of the λI-calculus [Chu41] is defined by the following grammar:*

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

where every abstraction $\lambda x.M$ satisfies $x \in FV(M)$.

As usual we write $\overrightarrow{M_i}$ for the application $M_1 \dots M_n$. We also use $[N, \langle M \rangle]$ or $[N, M_1, M_2, \dots, M_n]$ to denote the term $[\dots[[N, M_1], M_2], \dots, M_n]$.

The following property is straightforward by induction on terms.

Lemma 14 (Substitutions). *For all Λ_I-terms t, u, v, we have $t\{x\backslash u\} \in \Lambda_I$ and $t\{x\backslash u\}\{y\backslash v\} = t\{y\backslash v\}\{x\backslash u\{y\backslash v\}\}$ provided there is no variable capture.*

Definition 14. *The relation $\mathcal{I}$ between well-formed λlxr-terms and Λ_I is given by the following rules:*

$$\frac{}{x \,\mathcal{I}\, x} \qquad \frac{t \,\mathcal{I}\, T}{\lambda x.t \,\mathcal{I}\, \lambda x.T} \qquad \frac{t \,\mathcal{I}\, T \quad u \,\mathcal{I}\, U}{tu \,\mathcal{I}\, TU} \qquad \frac{t \,\mathcal{I}\, T}{t \,\mathcal{I}\, [T, N]}\ N \in \Lambda_I$$

$$\frac{t \,\mathcal{I}\, T \quad u \,\mathcal{I}\, U}{t\langle x = u\rangle \,\mathcal{I}\, T\{x\backslash U\}} \qquad \frac{t \,\mathcal{I}\, T}{C_x^{y,z}(t) \,\mathcal{I}\, T\{y\backslash x\}\{z\backslash x\}} \qquad \frac{t \,\mathcal{I}\, T}{W_x(t) \,\mathcal{I}\, T}\ x \in FV(T)$$

The relation $\mathcal{I}$ enjoys the following properties.

Lemma 15. *If $t \,\mathcal{I}\, M$, then*

1. $FV(t) \subseteq FV(M)$
2. $M \in \Lambda_I$
3. $x \notin FV(t)$ *and* $N \in \Lambda_I$ *implies* $t \,\mathcal{I}\, M\{x\backslash N\}$
4. $t \equiv t'$ *implies* $t' \,\mathcal{I}\, M$
5. $R^\Gamma_\Delta(t) \,\mathcal{I}\, R^\Gamma_\Delta(M)$

Proof. Property (1) is a straightforward induction on the proof tree as well as Property (2) which also uses Lemma 14. Properties (3) and (5) are also proved by induction on the tree, using Lemma 14. For Property (4):

- If $t\langle x = u\rangle\langle y = v\rangle \ \mathcal{I}\ M$ with $y \notin FV(u)$, then $M = [[T\{x\backslash U\}, \langle T\rangle]\{y\backslash V\}, \langle U\rangle]$ with $t\ \mathcal{I}\ T$, $u\ \mathcal{I}\ U$ and $v\ \mathcal{I}\ V$. We can assume that $x \notin FV(T_1)\cup\ldots\cup FV(T_m)\cup FV(V)$, so that $M = [[T, \langle T\rangle]\{x\backslash U\}\{y\backslash V\}, \langle U\rangle] = [[T, \langle T\rangle]\{y\backslash V\}\{x\backslash U\{y\backslash V\}\}, \langle U\rangle]$. As a consequence $t\langle y = v\rangle\langle x = u\rangle\ \mathcal{I}\ M$, since by (3) we get $u\ \mathcal{I}\ U\{y\backslash V\}$.
- The associativity and commutativity of contraction are very similar.
- If $W_x(W_y(t))\ \mathcal{I}\ M$ then $M = [[T, \langle T\rangle], \langle U\rangle]$ with $t\ \mathcal{I}\ T$, $y \in FV(T)$ and $x \in FV([T, \langle T\rangle])$. Then $W_y(W_x(t))\ \mathcal{I}\ M$.
- If $C_w^{y,z}(p)\langle x = u\rangle\ \mathcal{I}\ M$, then $M = [[P\{y\backslash w\}\{z\backslash w\}, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$, $p\ \mathcal{I}\ P$ and $u\ \mathcal{I}\ U$. We then conclude by the fact that $C_w^{y,z}(p\langle x = u\rangle)\ \mathcal{I}\ M$ where $M = [P\{x\backslash U\}\{y\backslash w\}\{z\backslash w\}, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$.

Theorem 8 (Simulation in Λ_I).

1. *If $t\ \mathcal{I}\ T$ and $t \longrightarrow_{\mathsf{xr}} t'$, then $t'\ \mathcal{I}\ T$.*
2. *If $t\ \mathcal{I}\ T$ and $t \longrightarrow_B t'$, then there is $T' \in \Lambda_I$ such that $t'\ \mathcal{I}\ T'$ and $T \longrightarrow^+_{\beta\pi} T'$.*

Proof. We first analyse every possible reduction case at the root.

- B: $(\lambda x.p)\ u \longrightarrow\ p\langle x = u\rangle$.
  Then $T = [[\lambda x.P, \langle P\rangle]U, \langle U\rangle]$ with $p\ \mathcal{I}\ P$ and $u\ \mathcal{I}\ U$. We then obtain the reduction sequence $T \longrightarrow^*_\pi [(\lambda x.P)U, \langle P\rangle, \langle U\rangle] \longrightarrow_\beta [P\{x\backslash U\}, \langle P\rangle, \langle U\rangle] = T'$.
- Abs: $(\lambda y.p)\langle x = u\rangle \longrightarrow\ \lambda y.p\langle x = u\rangle$. Then $T = [[\lambda y.P, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$ with $p\ \mathcal{I}\ P$ and $u\ \mathcal{I}\ U$. We have $T = [\lambda y.(P\{x\backslash U\}), \langle P\{x\backslash U\}\rangle, \langle U\rangle]$.
- $App1$,$App2$: Similar to the previous case.
- Var: $x\langle x = u\rangle \longrightarrow\ u$. Then $T = [[x, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$ with $u\ \mathcal{I}\ U$. We have $T = [U, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$.
- $Weak1$: $W_x(p)\langle x = u\rangle \longrightarrow\ W_{FV(u)}(p)$.
  Then $T = [[P, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$ with $p\ \mathcal{I}\ P$, $u\ \mathcal{I}\ U$, and $x \in FV(P)$. We have $T = [P\{x\backslash U\}, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$. Since $x \notin FV(p)$, then by Lemma 15-3 $p\ \mathcal{I}\ P\{x\backslash U\}$, and since $x \in FV(P)$, $FV(U) \subseteq FV(P\{x\backslash U\})$. By Lemma 15-1 $FV(u) \subseteq FV(U)$ so that $FV(u) \subseteq FV(P\{x\backslash U\})$ concludes the proof.
- $Weak2$: $W_y(p)\langle x = u\rangle \longrightarrow\ W_y(p\langle x = u\rangle)$.
  Then $T = [[P, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$ with $p\ \mathcal{I}\ P$, $u\ \mathcal{I}\ U$, and $y \in FV(P)$. We have $T = [P\{x\backslash U\}, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$ and we still have $y \in FV(P\{x\backslash U\})$.
- $Cont1$: $C_x^{y,z}(p)\langle x = u\rangle \longrightarrow\ C_\Gamma^{\Delta,\Pi}(p\langle y = R_\Delta^\Gamma(u)\rangle\langle z = R_\Pi^\Gamma(u)\rangle)$.
  Then $T = [[P\{y\backslash x\}\{z\backslash x\}, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$ with $p\ \mathcal{I}\ P$ and $u\ \mathcal{I}\ U$. We obtain the following equality $T = [P\{y\backslash U\}\{z\backslash U\}, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$ which can be expressed as

$$T = [P\{y\backslash U'\}\{z\backslash U''\}\{\Delta\backslash\Gamma\}\{\Pi\backslash\Gamma\}, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$$

where $U' = U\{\Gamma\backslash\Delta\}$ and $U'' = U\{\Gamma\backslash\Pi\}$. We obtain $R^{\Gamma}_{\Delta}(u)\ \mathcal{I}\ U'$ and $R^{\Gamma}_{\Pi}(u)\ \mathcal{I}\ U''$ by Lemma 15-5.

- *Comp*: $p\langle y = v\rangle\langle x = u\rangle \longrightarrow\ p\langle y = v\langle x = u\rangle\rangle$ where $x \in FV(v)$. Then $T = [[P\{y\backslash Q\}, \langle P\rangle]\{x\backslash U\}, \langle U\rangle]$ with $t\ \mathcal{I}\ P$, $v\ \mathcal{I}\ Q$, and $u\ \mathcal{I}\ U$.
  We have $T = [P\{x\backslash U\}\{y\backslash Q\{x\backslash U\}\}, \langle P\{x\backslash U\}\rangle, \langle U\rangle]$. Notice that we obtain $t\ \mathcal{I}\ P\{x\backslash U\}$ by Lemma 15-3.
- *WAbs*, *WApp*1, *WApp*2, *WSubs*, *Cross* are straightforward because the condition $x \in FV(P)$ that is checked by $W_x()$ is just changed into a side-condition $x \in FV(Q)$ (checked one step later), where $x \in FV(P)$ implies $x \in FV(Q)$.
- *Merge*: $C^{y,z}_{w}(W_y(p)) \longrightarrow\ R^{z}_{w}(p)$.
  Then $T = [[P, \langle P\rangle]\{y\backslash w\}\{z\backslash w\}, \langle U\rangle]$ with $t\ \mathcal{I}\ P$ and $y \in FV(P)$. We then have the equality $T = [[P\{z\backslash w\}, \langle P\{z\backslash w\}\rangle]\{y\backslash w\}, \langle U\rangle]$ and we conclude by Lemma 15-3.
- *CAbs*: $C^{y,z}_{w}(\lambda x.t) \longrightarrow\ \lambda x.C^{y,z}_{w}(p)$.
  Then $T = [[\lambda x.P, \langle P\rangle]\{y\backslash w\}\{z\backslash w\}, \langle U\rangle]$ with $t\ \mathcal{I}\ P$.
  We have $T = [\lambda x.(P\{y\backslash w\}\{z\backslash w\}), \langle P\{y\backslash w\}\{z\backslash w\}\rangle, \langle U\rangle]$.
- *CApp*1, *CApp*2: Similar to the previous case.
- *CSubs*: We have $C^{y,z}_{w}(p\langle x = u\rangle)\ \mathcal{I}\ [[P\{x\backslash U\}, \langle P\rangle]\{y\backslash w\}\{z\backslash w\}, \langle U\rangle]$ which is equal to $T = [[P\{y\backslash w\}\{z\backslash w\}\{x\backslash U\{y\backslash w\}\{z\backslash w\}\}, \langle P\rangle\{y\backslash w\}\{z\backslash w\}], \langle U\rangle]$ by Lemma 14. We have $p\langle x = C^{y,z}_{w}(u)\rangle\ \mathcal{I}\ T$ by Lemma 15-3, which concludes this case.

Now for the closure under context, we use the fact that if $P \longrightarrow_{\beta\pi}\ P'$ then $P\{x\backslash U\} \longrightarrow_{\beta\pi}\ P'\{x\backslash U\}$, and if moreover $x \in FV(P)$ then $P\{x\backslash U\}\longrightarrow^{+}{}_{\beta\pi}\ P\{x\backslash U'\}$. The latter is useful for the closure: if $p\langle x = t\rangle\ \mathcal{I}\ Q$ and $t \longrightarrow_{B}\ t'$, then $Q = [P\{x\backslash T\}, \langle U\rangle]$ with $p\ \mathcal{I}\ P$, $u\ \mathcal{I}\ U$ and by induction hypothesis we get $T\longrightarrow^{+}{}_{\beta\pi}\ T'$ such that $t'\ \mathcal{I}\ T'$. Since $x \in FV(p)$, $x \in FV(P)$ by Lemma 15-2, and hence $Q\longrightarrow^{+}{}_{\beta\pi}\ [P\{x\backslash T'\}, \langle U\rangle]$.

Corollary 2. *If $t\ \mathcal{I}\ T$ and $T \in SN_{\beta\pi}$, then $t \in SN_{\lambda\mathsf{lxr}}$.*

Proof. Suppose that $t \notin SN_{\lambda\mathsf{lxr}}$. Then there is an infinite $\lambda\mathsf{lxr}$-reduction sequence σ starting at t. Since xr is terminating (Lemma 1), then there are infinite B-steps in the sequence σ, so that it is of the form

$$\sigma : t\longrightarrow^{*}{}_{\mathsf{xr}}\ \longrightarrow_{B}\ t_1 \cdots \longrightarrow^{*}{}_{\mathsf{xr}}\ \longrightarrow_{B}\ t_i\longrightarrow^{*}{}_{\mathsf{xr}}\ \longrightarrow_{B}\ \cdots$$

By Theorem 8 we can construct another infinite $\beta\pi$-reduction sequence

$$T\longrightarrow^{+}{}_{\beta\pi}\ T_1\longrightarrow^{+}{}_{\beta\pi}\ \cdots\longrightarrow^{+}{}_{\beta\pi}\ T_i\longrightarrow^{+}{}_{\beta\pi}\ \cdots$$

where $t_i\ \mathcal{I}\ T_i$. This contradicts the hypothesis $T \in SN_{\beta\pi}$.

Definition 15 ([Len05]). *We encode the λ-calculus into Λ_I as follows:*

$$\begin{array}{lll} i(x) & = x & \\ i(\lambda x.t) & = \lambda x.i(t) & x \in FV(t) \\ i(\lambda x.t) & = \lambda x.[i(t), x] & x \notin FV(t) \\ i(t\ u) & = i(t)\ i(u) & \end{array}$$

Theorem 9 ([Len05]). *For any λ-term t, if $t \in SN_\beta$, then $i(t) \in SN_{\beta\pi}$.*

Theorem 10. *For any λ-term u, $\mathcal{A}(u)\ \mathcal{I}\ i(u)$.*

Proof. By induction on u:

- $x\ \mathcal{I}\ x$ trivially holds.
- If $u = \lambda x.t$, then $\mathcal{A}(t)\ \mathcal{I}\ i(t)$ holds by induction hypothesis. Therefore, we obtain $\lambda x.\mathcal{A}(t)\ \mathcal{I}\ \lambda x.i(t)$ and $\lambda x.W_x(\mathcal{A}(t))\ \mathcal{I}\ \lambda x.[i(t), x]$.
- If $u = (t\ u)$, then $\mathcal{A}(t)\ \mathcal{I}\ i(t)$ and $\mathcal{A}(u)\ \mathcal{I}\ i(u)$ hold by induction hypothesis and $R^\Gamma_\Pi(\mathcal{A}(t))\ \mathcal{I}\ R^\Gamma_\Pi(i(t))$ and $R^\Gamma_\Pi(\mathcal{A}(u))\ \mathcal{I}\ R^\Gamma_\Pi(i(u))$ by Lemma 15-5. Since $R^\Gamma_\Pi(i(t))\{\Pi\backslash\Gamma\} = i(t)$ (and the same for $i(u)$), we can then conclude $C^{\Delta,\Pi}_\Gamma(R^\Gamma_\Delta(\mathcal{A}(t))\ R^\Gamma_\Pi(\mathcal{A}(u)))\ \mathcal{I}\ i(t)\ i(u)$.

Corollary 3 (PSN). *For any λ-term t, if $t \in SN_\beta$, then $\mathcal{A}(t) \in SN_{\lambda\mathsf{lxr}}$.*

Proof. If $t \in SN_\beta$, then $i(t) \in SN_{\beta\pi}$ by Theorem 9. As $\mathcal{A}(t)\ \mathcal{I}\ i(t)$ by Theorem 10, then we conclude $\mathcal{A}(t) \in SN_{\lambda\mathsf{lxr}}$ by Corollary 2.

5.2 Confluence

We now use both simulations presented in Section 4 to derive the confluence property for systems xr and $\lambda\mathsf{lxr}$ via a generalisation of the Interpretation Method [Har87]. We start by stating a result that relates xr-normal forms to the composition of encodings $\mathcal{A}$ and $\mathcal{B}$.

Theorem 11. *If t is an xr-normal form, then $t \equiv W_{\mathcal{FV}(t)\backslash\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.*

Proof. The proof may proceed by induction since a subterm of an xr-normal form is an xr-normal form:

- If $t = x$, then $x = \mathcal{A}(\mathcal{B}(x))$ and $\mathcal{FV}(t) \setminus \mathcal{FV}(\mathcal{B}(t)) = \emptyset$
- If $t = \lambda x.M$, then we know $t \equiv \lambda x.W_{\mathcal{FV}(M)\backslash\mathcal{FV}(\mathcal{B}(M))}(\mathcal{A}(\mathcal{B}(M)))$ by i.h. But t is an xr-normal form, so $\mathcal{FV}(M) \setminus \mathcal{FV}(\mathcal{B}(M)) \subseteq \{x\}$ (otherwise it can be reduced by $WAbs$).
  Hence $\mathcal{FV}(M) \setminus \mathcal{FV}(\mathcal{B}(M)) = \{x\} \setminus \mathcal{FV}(\mathcal{B}(M))$ and $\mathcal{FV}(t) = \mathcal{FV}(\mathcal{B}(t))$, so that $t \equiv \lambda x.W_{\{x\}\backslash\mathcal{FV}(\mathcal{B}(M))}(\mathcal{A}(\mathcal{B}(M))) \equiv \mathcal{A}(\mathcal{B}(t))$.

– If $t = u\ v$, $t \equiv W_{\mathcal{FV}(u)\setminus\mathcal{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u)))\ W_{\mathcal{FV}(v)\setminus\mathcal{FV}(\mathcal{B}(v))}(\mathcal{A}(\mathcal{B}(v)))$ by i.h. But t is a xr-normal form, so

$$\mathcal{FV}(u) \setminus \mathcal{FV}(\mathcal{B}(u)) = \mathcal{FV}(v) \setminus \mathcal{FV}(\mathcal{B}(v)) = \emptyset$$

(otherwise it could be reduced by $WApp1$ or $WApp2$). Hence, $\mathcal{FV}(t) = \mathcal{FV}(\mathcal{B}(t))$ and $t \equiv \mathcal{A}(\mathcal{B}(u))\ \mathcal{A}(\mathcal{B}(v)) \equiv \mathcal{A}(\mathcal{B}(t))$ since u and v have no variable in common.

– The case $t = u\langle x = v\rangle$ is not possible by Lemma 3.

– If $t = W_x(M)$, $t \equiv W_x(W_{\mathcal{FV}(M)\setminus\mathcal{FV}(\mathcal{B}(M))}(\mathcal{A}(\mathcal{B}(M))))$ by i.h. This last term is equal to $W_{\mathcal{FV}(t)\setminus\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$ since $x \in \mathcal{FV}(t)$ but $x \notin \mathcal{FV}(\mathcal{B}(t))$.

– If $t = C_x^{y,z}(M)$, $t \equiv C_x^{y,z}(W_{\mathcal{FV}(M)\setminus\mathcal{FV}(\mathcal{B}(M))}(\mathcal{A}(\mathcal{B}(M))))$ by i.h.
We first remark that y and z are free in M since t is well-formed, and also x is not free in M, hence neither is it free in $\mathcal{B}(M)$.
Secondly, since t is an xr-normal form, we have $\mathcal{FV}(M) \setminus \mathcal{FV}(\mathcal{B}(M)) = \emptyset$ (otherwise t could be reduced by $Cross$ or $Merge$). Hence, y and z are free in $\mathcal{B}(M)$ and $t \equiv C_x^{y,z}(\mathcal{A}(\mathcal{B}(M)))$.
But $\mathcal{B}(t) = \mathcal{B}(M)\{y\backslash x\}\{z\backslash x\}$, so x is free in $\mathcal{B}(t)$. We conclude $\mathcal{FV}(t) = \mathcal{FV}(\mathcal{B}(t))$.
Third, notice that $\mathcal{B}(M)$ can be neither a variable (otherwise t would not be well-formed) nor an abstraction (otherwise t could be reduced by $CAbs$), so $\mathcal{B}(M) = u\ v$, and $\mathcal{A}(\mathcal{B}(M)) = C_\Phi^{\Delta,\Pi}(R_\Delta^\Phi(\mathcal{A}(u))\ R_\Pi^\Phi(\mathcal{A}(v)))$ with $\Phi = \mathcal{FV}(u) \cap \mathcal{FV}(v)$. Hence, $t \equiv C_x^{y,z}(C_\Phi^{\Delta,\Pi}(R_\Delta^\Phi(\mathcal{A}(u))\ R_\Pi^\Phi(\mathcal{A}(v))))$.
Now it would suffice that $y \in \mathcal{FV}(u) \setminus \mathcal{FV}(v)$ and $z \in \mathcal{FV}(v) \setminus \mathcal{FV}(u)$ to prove that this term is in fact

$$\mathcal{A}(u\{y\backslash x\}\ v\{z\backslash x\}) = \mathcal{A}(\mathcal{B}(M)\{y\backslash x\}\{z\backslash x\}) = \mathcal{A}(\mathcal{B}(t))$$

We are going to prove that this is the case (or the symmetrical case when y and z are swapped): we know that they are free in $u\ v$.
Suppose that one of them, say y, is both in u and in v. Then $y \in \Phi$, so

$$t \equiv C_x^{y,z}(C_{\Phi',y}^{(\Delta',y'),(\Pi',y'')}(R_\Delta^\Phi(\mathcal{A}(u))\ R_\Pi^\Phi(\mathcal{A}(v))))$$

which we can rearrange into

$$t \equiv C_x^{y,y''}(C_{\Phi',y}^{(\Delta',y'),(\Pi',z)}(R_\Delta^\Phi(\mathcal{A}(u))\ R_\Pi^\Phi(\mathcal{A}(v))))$$

if $z \in \mathcal{FV}(u)$, or $t \equiv C_x^{y,y'}(C_{\Phi',y}^{(\Delta',z),(\Pi',y'')}(R_\Delta^\Phi(\mathcal{A}(u))\ R_\Pi^\Phi(\mathcal{A}(v))))$ if $z \in \mathcal{FV}(v)$.
In the first case, t can be reduced by $CApp1$ (on $C_y^{y',z}()$), and in the second by $CApp2$ (on $C_y^{z,y''}()$). In both cases, it contradicts the fact that t is a xr-normal form. Hence, $y \notin \Phi$ (and similarly $z \notin \Phi$).
Now suppose that both y and z are on the same side, say in u. Then t can be reduced by $CApp1$ on $C_x^{y,z}()$. Similarly, they cannot be both in v (t could be reduced by $CApp2$). Hence one of them is only in u, and the other is only in v, as required.

Lemma 16. *The system* xr *is confluent and terminating, and the* xr*-normal form of* t *is* $W_{\mathcal{FV}(t)\setminus\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

Proof. By Theorem 1 the system xr is terminating so that we can take any xr-normal form t' of t such that $t \longrightarrow^*_{\mathsf{xr}} t'$. We then have $\mathcal{FV}(t) = \mathcal{FV}(t')$ by Lemma 1 and $\mathcal{B}(t) = \mathcal{B}(t')$ by Theorem 1. Since t' is an xr-normal form, then $t' \equiv W_{\mathcal{FV}(t')\setminus\mathcal{FV}(\mathcal{B}(t'))}(\mathcal{A}(\mathcal{B}(t')))$ by Theorem 11. Hence $t' \equiv W_{\mathcal{FV}(t)\setminus\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

To show confluence let us suppose $t \longrightarrow^*_{\mathsf{xr}} t_1$ and $t \longrightarrow^*_{\mathsf{xr}} t_2$. Let us take xr-normal forms t'_1 and t'_2 such that $t_i \longrightarrow^*_{\mathsf{xr}} t'_i$. By the previous remark both t'_1 and t'_2 are congruent to $W_{\mathcal{FV}(t)\setminus\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$ which concludes the proof.

Theorem 12. *The system* $\lambda\mathsf{lxr}$ *is confluent.*

Proof. Suppose $N \longrightarrow^*_{\lambda\mathsf{lxr}} N_1$ and $N \longrightarrow^*_{\lambda\mathsf{lxr}} N_2$. We get $\mathcal{B}(N) \longrightarrow^*_{\beta} \mathcal{B}(N_1)$ and $\mathcal{B}(N) \longrightarrow^*_{\beta} \mathcal{B}(N_2)$ using Theorem 1. Now, confluence of $\longrightarrow_{\beta}$ gives us a λ-term M such that $\mathcal{B}(N_1) \longrightarrow^*_{\beta} M$ and $\mathcal{B}(N_2) \longrightarrow^*_{\beta} M$. Thus, by Theorem 7, there exist two lists of variables Φ_1, Φ_2 such that $\mathcal{A}(\mathcal{B}(N_1)) \longrightarrow^*_{\lambda\mathsf{lxr}} W_{\Phi_1}(\mathcal{A}(M))$ and $\mathcal{A}(\mathcal{B}(N_2)) \longrightarrow^*_{\lambda\mathsf{lxr}} W_{\Phi_2}(\mathcal{A}(M))$.

We can now reduce N_1 and N_2 to their respective xr-normal form so that Lemma 16 provides Φ'_1 and Φ'_2 such that

$$N_1 \longrightarrow^*_{\mathsf{xr}} W_{\Phi'_1}(\mathcal{A}(\mathcal{B}(N_1))) \longrightarrow^*_{\lambda\mathsf{lxr}} W_{\Delta_1}(\mathcal{A}(M))$$

$$N_2 \longrightarrow^*_{\mathsf{xr}} W_{\Phi'_2}(\mathcal{A}(\mathcal{B}(N_2))) \longrightarrow^*_{\lambda\mathsf{lxr}} W_{\Delta_2}(\mathcal{A}(M))$$

for $\Delta_1 = \Phi'_1, \Phi_1$ and $\Delta_2 = \Phi'_2, \Phi_2$.

The relation $\longrightarrow_{\lambda\mathsf{lxr}}$ preserves the set of free variables by Lemma 1, so that

$$\begin{array}{l}\mathcal{FV}(W_{\Delta_1}(\mathcal{A}(M))) = \mathcal{FV}(N_1) = \mathcal{FV}(N) \\ \mathcal{FV}(W_{\Delta_2}(\mathcal{A}(M))) = \mathcal{FV}(N_2) = \mathcal{FV}(N)\end{array}$$

We can conclude that Δ_1 is a permutation of Δ_2, hence N_1 and N_2 both reduce to the same term up to $\equiv$, as required.

6 Conclusion and further work

This paper extends the explicit substitution paradigm by showing how the Proof Nets of Linear Logic can be suitable as a logical model of a language with explicit operators for erasure, duplication and substitution.

Our term language is expressed by a simple syntax, and enjoys natural operational semantics via a well-established notion of reduction modulo a set of equations. Soundness and completeness of $\lambda\mathsf{lxr}$ are shown with respect to its Proof Nets model.

In contrast to other calculi in the literature, $\lambda\mathsf{lxr}$ has full composition and enjoys PSN. Moreover, $\lambda\mathsf{lxr}$ enjoys all the nice properties that one expects such as confluence, strong normalisation of well-typed terms and step by step simulation of β-reduction. All these properties are shown by considering the complex notion of reduction modulo an equivalence which we have associated to λlxr-terms.

We claim that weakening operators are well-adapted to implement efficient garbage collection. Indeed, free variables are never lost and weakening operators are pulled out to the top-level during computation.

Our soundness and completeness proofs illustrate that the only rules with computational relevance in λlxr are $\{App2, Comp, Var, Weak1, Cont1\}$, just as the interesting rules in Proof Nets are only those concerning boxes. More precisely, $App2$ and $Comp$ in λlxr correspond to b-b in PN, Var to d-b, $Weak1$ to w-b and $Cont1$ to c-b.

It is worth mentioning the calculus obtained by turning the equation $Cont2$ into a reduction rule (from left to right) and by eliminating reduction rules $WSubs$ and $CSubs$ enjoys exactly the same properties as the calculus presented in this paper, namely Theorems 2,4,7,1,12, and Corollary 3. However, they seem to be necessary for the confluence on open terms (ongoing work).

We think that many interesting points raised in this work deserve further development. The first one concerns the study of reduction strategies well-adapted to handle explicit operators of substitution, erasure and duplication. This may take into account the notion of weak reduction used to implement functional programming [LM99].

Proof techniques used in the literature to show PSN of calculi with explicit substitutions (zoom-in [ABR00], minimality [BBLRD96], semantic RPO [BG99], PSN by standardisation [KOvO01], or intersection types [DL01]) are not all easy to adapt/extend to λlxr and other formalisms. We believe that the proof technique used here is really flexible.

But using the PSN result, we believe that we can characterise very neatly the strongly normalising terms of λlxr as the terms typable with intersection types, as it it the case in λ-calculus as well as in the explicit substitution calculus λx [LLD+04].

First-order term syntax for λlxr via de Bruijn indices, or other special notation to avoid α-conversion as for example explicit scoping [HvO03] or also director strings [SFM03], would make implementation easier and bring the term language even closer to the Proof Nets model which has no notion of binding.

Acknowledgements

We are very grateful to R. Dyckhoff, J. Forest, B. Guillaume, O. Laurent, P. Lescanne, R. Matthes, J. Mc Kinna and V. van Oostrom for valuable comments and suggestions.

References

[Abr93] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[ABR00] A. Arbiser, E. Bonelli, and A. Ríos. Perpetuality in a lambda calculus with explicit substitutions and composition. Workshop Argentino de Informática Teórica (WAIT), JAIIO, 2000.

[ACCL91] M. Abadi, L. Cardelli, P. L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[AG98] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[Bar84] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984. Revised Edition.

[BBdH93] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. A term calculus for intuitionistic linear logic. In J. F. Groote and M. Bezem, editors, *Proceedings of the 1st International Conference of Typed Lambda Calculus and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90. Springer-Verlag, May 1993.

[BBLRD96] 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):699–722, 1996.

[BG99] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211(1-2):375–395, 1999.

[Bon01] E. Bonelli. Perpetuality in a named lambda calculus with explicit substitutions. *Mathematical Structures in Computer Science*, 11(1):47–90, 2001.

[BR95] R. Bloo and K. Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. In *Computing Science in the Netherlands*, pages 62–72. Netherlands Computer Science Research Foundation, 1995.

[Chu41] A. Church. *The Calculi of Lambda Conversion*. Princeton University Press, 1941.

[CK99] S. Cerrito and D. Kesner. Pattern matching as cut elimination. In G. Longo, editor, *14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 98–108. IEEE Computer Society Press, July 1999.

[Coq] The Coq Proof Assistant. `http://coq.inria.fr/`.

[DCG99] R. Di Cosmo and S. Guerrini. Strong normalization of proof nets modulo structural congruences. In P. Narendran and M. Rusinowitch, editors, *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 75–89. Springer-Verlag, July 1999.

[DCK97] R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *12th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 35–46. IEEE Computer Society Press, July 1997.

[DCKP00] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. In J. Tiuryn, editor, *Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1784 of *Lecture Notes in Computer Science*, pages 63–81. Springer-Verlag, Mar. 2000.

[DCKP03] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409–450, 2003.

[DG01] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11:169–206, 2001.

[DHK95] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, 1995.

[DJS95] V. Danos, J.-B. Joinet, and H. Schellinx. Sequent calculi for second order logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*. Cambridge University Press, 1995.

[DL01] D. Dougherty and P. Lescanne. Reductions, intersection types and explicit substitutions. In S. Abramsky, editor, *Proceedings of the 5th International Conference of Typed Lambda Calculus and Applications*, volume 2044 of *Lecture Notes in Computer Science*, pages 121–135. Springer-Verlag, May 2001.

[FM99] M. Fernández and I. Mackie. Closed reductions in the lambda calculus. In J. Flum and M. Rodríguez-Artalejo, editors, *Proceedings of the 13th Annual Conference of the European Association for Computer Science Logic (CSL)*, volume 1683 of *Lecture Notes in Computer Science*. Springer-Verlag, Sept. 1999.

[For02] J. Forest. A weak calculus with explicit operators for pattern matching and substitution. In S. Tison, editor, *13th International Conference on Rewriting Techniques and Applications*, volume 2378 of *Lecture Notes in Computer Science*, pages 174–191. Springer-Verlag, July 2002.

[GdR00] N. Ghani, V. de Paiva, and E. Ritter. Linear explicit substitutions. *Logic Journal of the Interest Group of Pure and Applied Logic*, 8(1):7–31, 2000.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[GL98] J. Goubault-Larrecq. A proof of weak termination of typed lambda sigma-calculi. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Proceedings of the International Workshop Types for Proofs and Programs*, volume 1512 of *Lecture Notes in Computer Science*, pages 134–151. Springer-Verlag, Dec. 1998.

[Har87] T. Hardin. *Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs.* Thèse de doctorat, Université de Paris VII, 1987.

[Her94] H. Herbelin. A λ-calculus structure isomorphic to sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of the 8th Annual Conference of the European Association for Computer Science Logic (CSL)*, volume 933 of *Lecture Notes in Computer Science*. Springer-Verlag, Sept. 1994.

[HL89] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[HMP96] T. Hardin, L. Maranget, and B. Pagano. Functional back-ends within the lambda-sigma calculus. In R. K. Dybvig, editor, *Proceedings of the ACM International Conference on Functional Programming*. ACM Press, May 1996.

[HOL] The HOL system. `http://www.dcs.gla.ac.uk/~tfm/fmt/hol.html`.

[HvO03] D. Hendriks and V. van Oostrom. Adbmal. In C. Kirchner, editor, *19th Conference on Automated Deduction (CADE)*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 136–150. Springer-Verlag, July 2003.

[Klo80] J.-W. Klop. *Combinatory Reduction Systems.* PhD thesis, Mathematical Centre Tracts 127, CWI, Amsterdam, 1980.

[KOvO01] Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Uniform Normalization Beyond Orthogonality. In A. Middeldorp, editor, *12th International Conference on Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, pages 122–136. Springer-Verlag, May 2001.

[KR95] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In D. Swierstra and M. Hermenegildo, editors, *Proceedings of the 7th International Symposium on Proceedings of the International Symposium on Programming Language Implementation and Logic Programming*, volume 982 of *Lecture Notes in Computer Science*, pages 45–62. Springer-Verlag, Sept. 1995.

[Laf90] Y. Lafont. Interaction nets. In *17th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 95–108. ACM, 1990.

[Lau03] O. Laurent. Polarized proof-nets and lambda-mu calculus. *Theoretical Computer Science*, 1(290):161–188, 2003.

[Len05] S. Lengrand. Induction principles as the foundation of the theory of normalisation: Concepts and techniques. Technical report, PPS laboratory, Université Paris 7, Mar. 2005.

[LLD+04] S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

[LM99] J.-J. Lévy and L. Maranget. Explicit substitutions and programming languages. In R. R. C. Pandu Rangan, Venkatesh Raman, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 181–200. Springer-Verlag, Dec. 1999.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the 2nd International Conference of*

Typed Lambda Calculus and Applications, volume 902 of *Lecture Notes in Computer Science*, pages 328–334. Springer-Verlag, Apr. 1995.

[Par92] M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, July 1992.

[Pol04] E. Polonovski. *Substitutions explicites, logique et normalisation*. Thèse de doctorat, Université Paris 7, 2004.

[Reg94] L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 2(126):281–292, 1994.

[RR97] S. Ronchi della Rocca and L. Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 59(3), 1997.

[SFM03] F.-R. Sinot, M. Fernández, and I. Mackie. Efficient reductions with director strings. In R. Nieuwenhuis, editor, *14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 46–60. Springer-Verlag, June 2003.

[Sor97] M. H. Sorensen. Strong normalization from weak normalization in typed lambda-calculi. *IandC*, 37:35–71, 1997.

[vO01] V. van Oostrom. Net-calculus. Course Notes available on `http://www.phil.uu.nl/~oostrom/typcomp/00-01/net.ps`, 2001.

[VW01] R. Vestergaard and J. Wells. Cut rules and explicit substitutions. *Mathematical Structures in Computer Science*, 11(1):131–168, 2001.

[Wad93] P. Wadler. A syntax for linear logic. In S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, editors, *The 9th International Conference on the Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*, pages 513–529. Springer-Verlag, Apr. 1993.

[Xi97] H. Xi. Weak and strong beta normalisations in typed lambda-calculi. In P. de Groote, editor, *Proceedings of the 3th International Conference of Typed Lambda Calculus and Applications*, volume 1210 of *Lecture Notes in Computer Science*, pages 390–404. Springer-Verlag, Apr. 1997.