Resource 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 calculus with an explicit control of erasure and duplication of substitutions, enjoying a sound and complete correspondence with the intuitionistic fragment of Linear Logic's proof-nets. We show the operational behaviour of the calculus and some of its fundamental properties such as confluence, preservation of strong normalisation, strong normalisation of simply-typed terms, step by step simulation of β-reduction and full composition.

1 Introduction

The *Curry-Howard* paradigm, according to which the terms/types/reduction of a term calculus 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 calculus and Natural Deduction for Intuitionistic Logic as logical system. 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 by implementing (higher-order) substitution as the interaction between (and the propagation of) erasure, duplication and linear substitution operators. On the other hand *Linear Logic* [Gir87] decomposes the intuitionistic logical connectives into more elementary connectives, such as the linear implication and the exponentials, thus providing a more refined and controlled use of resources (formulae) than that of Intuitionistic Logic.

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

Several works [DCK97,DCKP00,Abr93,vO01,FM99] 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.

In this paper, we present a calculus called $\lambda\mathsf{lxr}$ with erasure, duplication and linear substitution operators, which can be seen as a λ-calculus with *explicit substitutions*. Its simply-typed version can be considered, via the Curry-Howard paradigm, as a functional computational counterpart to the intuitionistic fragment of proof-nets. The major features of this calculus are

- Simple syntax and intuitive operational 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 explicit substitutions;
- Nice properties such as confluence, preservation of strong normalisation, strong normalisation for Curry-style simply-typed terms, and step by step simulation of β-reduction.

Explicit control of resources and Proof-nets

Much work on explicit substitutions has been done in the last 15 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 flaw in the design of these calculi with explicit substitutions in that they are supposed to implement their underlying calculus without losing its good properties such as strong normalisation of simply-typed terms. 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 exploits very little of the notion of explicit substitutions because they can be neither composed nor even delayed.

In order to cope with this problem David and Guillaume [DG01] defined a calculus with *labels*, called λ_{ws}, which allows *controlled* composition of explicit substitutions without losing PSN. These labels can be also seen as special annotations induced by a logical *weakening* rule. But the λ_{ws}-calculus has a complicated syntax and its named version [DCKP00] is even less readable. 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 simply-typed λ_{ws} into the proof-nets of Linear Logic. This translation gives at the same time a proof of strong normalisation for simply-typed λ_{ws}-terms. Moreover, the translation reveals a natural semantics for composition of explicit substitutions, and also suggests that explicit erasure and duplication operators 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.

The operators of $\lambda\mathsf{lxr}$ have thus a nice logical interpretation in the typed case: the linear substitution operator is *cut*, the duplication operator is *contraction*, the erasure operator is *weakening*. From the point of view of implementation, this can be read as the fact that substitution, 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 in calculi such as λ_{ws} and $\lambda\mathsf{lxr}$ that we illustrate via an example. Let us denote by $\mathcal{W}_{\_}(\_)$ the weakening operator, so that a $\lambda\mathsf{lxr}$-term whose variable x is used to weaken the term t is written $\mathcal{W}_x(t)$, that is, we explicitly annotate that the variable x does not appear free in the term t. Then, when evaluating the application of a term $\lambda x.\mathcal{W}_x(t)$ to another term u, a linear substitution operator $\langle x \backslash u\rangle$ is created and the computation will continue with $\mathcal{W}_x(t)\langle x \backslash u\rangle$. Then, the weakening operator will be used to prevent the substitution $\langle x \backslash u\rangle$ from going into the term t, 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 λlxr-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 λlxr-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 step. Thus for example, when simulating the β-reduction steps $(\lambda x.\lambda y.x)\ u\ z \longrightarrow^*_\beta u$, 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 $\mathcal{W}_z(\overline{u})$ (where $\overline{u}$ is the interpretation of u in λlxr), thus preparing the situation for an efficient garbage collection on z.

The weakening operator can thus be seen as a tool for handling *garbage collection*. For instance, it is worth noticing that the labels of the λ_{ws}-calculus cannot be pulled out to the top-level as in λlxr. Also, free variables may be lost during λ_{ws}-computation. Thus, garbage collection within λ_{ws} does not offer the advantages existing in λlxr.

Composition

From a rewriting point of view this calculus can be viewed as the first term calculus 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*. Thus, λlxr gives a human-readable formalism between the abstract λ-calculus and the graph presentations given for example by sharing graphs [Gue99]. By simulation of β-reduction step by step we mean that every β-reduction step in λ-calculus induces a non-empty λlxr-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 other words, full composition means that the explicit substitution operator of the calculus implements exactly a notion of meta-substitution defined on the same calculus, that is, on terms having weakening, contraction and linear substitution operators. In particular, in a term $t\langle y \backslash u\rangle\langle x \backslash v\rangle$, the external substitution is not blocked by the internal one and can be further evaluated without ever requiring any preliminary evaluation of $t\langle y \backslash u\rangle$. In other words, the application of the substitution $\langle x \backslash v\rangle$ to the term t can be evaluated independently from that of $\langle y \backslash u\rangle$. 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 operators have already been proposed.

Herbelin [Her94] proposes a term calculus 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. But he gives no equalities between terms reflecting the irrelevance of some syntactic details appearing in sequent poofs. Many other term calculi based on sequents rather than proof-nets have been proposed for Linear Logic, as for example [GdR00,BBdH93,RR97,Wad93,OH06].

An axiomatisation of sharing graphs by means of higher-order term syntax is proposed by Hasegawa [Has99] who investigates categorical models of the term calculi thus obtained. The proof-nets that we use in this paper go beyond his general treatment in that they have a particular semantics with extra equations and reduction rules. Our point in this paper is to relate them to a resource-aware λ-calculus, while the case studies in [Has99] are those of Ariola and Klop's cyclic lambda calculi and Milner's action calculi.

A related approach was independently developed by V. van Oostrom (available in course notes written in Dutch [vO01]), where operators for contraction and weakening are added to the λ-calculus to define a fine control of duplication and erasing. We show here how the same operators allow a fine control of composition when using linear substitution operators, although the proofs of some fundamental properties, such as PSN and confluence, become harder. An 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 contraction, weakening and linear substitution 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 establish the main operational properties of $\lambda\mathsf{lxr}$. Finally we conclude and give some ideas for further work.

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

2.1 The linear syntax of $\boldsymbol{\lambda}$lxr

We present in this section the syntax of the untyped λlxr-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 [3] and can be just viewed as an extension of that of $\lambda\mathtt{x}$ [BR95]. We assume that we have a set of *variables*, denoted $x, y, z, \ldots$, that is in bijection with the natural numbers and is thus equipped with a *total order*.

$$t ::= x \mid \lambda x.t \mid t\ t \mid t\langle x \backslash t\rangle \mid \mathcal{W}_x(t) \mid \mathcal{C}_x^{y|z}(t)$$

Terms are thus trees, so we shall only use parentheses to remove any ambiguity when writing them as strings.

The term $\lambda x.t$ is called an *abstraction*, $t\ u$ an *application*, and $t\langle x \backslash u\rangle$ a *closure*. The term constructors $\mathcal{W}_\_(\_)$, $\mathcal{C}_\_^{\_|\_}(\_)$ and $\_\langle \_ \backslash \_\rangle$ are respectively called *weakening*, *contraction* and *linear substitution operators*.

Definition 1 (Term Size). *We define the* size *of a term t, written $|t|$, as follows:* $|x| = 1$, $|\lambda x.t| = |\mathcal{W}_x(t)| = |\mathcal{C}_x^{y|z}(t)| = |t| + 1$, *and* $|tv| = |t\langle x \backslash v\rangle| = |t| + |v| + 1$.

Definition 2. *We write $\mathcal{FV}(t)$ to denote the set of* free *variables of a term t defined by induction on $|t|$ as follows:*

$$\begin{array}{llll}
\mathcal{FV}(x) & := \{x\} & \mathcal{FV}(t\langle x \backslash u\rangle) & := (\mathcal{FV}(t) \setminus \{x\}) \cup \mathcal{FV}(u) \\
\mathcal{FV}(\lambda x.t) & := \mathcal{FV}(t) \setminus \{x\} & \mathcal{FV}(t\ u) & := \mathcal{FV}(t) \cup \mathcal{FV}(u) \\
\mathcal{FV}(\mathcal{W}_x(t)) & := \mathcal{FV}(t) \cup \{x\} & \mathcal{FV}(\mathcal{C}_x^{y|z}(t)) & := (\mathcal{FV}(t) \setminus \{y, z\}) \cup \{x\}
\end{array}$$

The terms $\lambda x.t$ and $t\langle x \backslash u\rangle$ thus bind x in t. The term $\mathcal{C}_x^{y|z}(t)$ binds y and z in t. From this notion of binding we obtain the correspoding notion of α-equivalence.

Note that the syntax could equivalently be given as a HRS [Nip91], with a type $\mathcal{V}$ for variables and a type $\mathcal{T}$ for (raw)-terms, and six typed constants corresponding to the six term constructors above:

$$\begin{array}{llll}
\mathsf{var}: & \mathcal{V} \to \mathcal{T} & \mathsf{sub}: & (\mathcal{V} \to \mathcal{T}) \to (\mathcal{T} \to \mathcal{T}) \\
\mathsf{abs}: & (\mathcal{V} \to \mathcal{T}) \to \mathcal{T} & \mathsf{app}: & \mathcal{T} \to (\mathcal{T} \to \mathcal{T}) \\
\mathsf{weak}: & \mathcal{V} \to (\mathcal{T} \to \mathcal{T}) & \mathsf{cont}: & \mathcal{V} \to ((\mathcal{V} \to (\mathcal{V} \to \mathcal{T})) \to \mathcal{T})
\end{array}$$

For instance the λlxr-term $\mathcal{C}_x^{y|z}(t_1\ t_2\ t_3)\langle x \backslash \lambda x'.x'\rangle$ would be represented as the HRS-term $\mathsf{sub}(x.\mathsf{cont}(x, y.z.\mathsf{app}(\mathsf{app}(t_1, t_2), t_3)), \mathsf{abs}(x'.\mathsf{var}(x')))$.

The consequent notions of free and bound variables and α-equivalence coincide with ours.

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

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

that does have a free occurrence (and hence only one). This condition can be formally expressed as follows.

Definition 3 (Linear terms). *A term t is said to be* linear *if t* linear *can be derived from the following rules:*

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

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

$$\frac{t\ \mathsf{linear} \quad x \notin \mathcal{FV}(t)}{\mathcal{W}_x(t)\ \mathsf{linear}} \qquad \frac{t\ \mathsf{linear} \quad x, y \in \mathcal{FV}(t) \quad x \neq y \quad z \notin \mathcal{FV}(t) \setminus \{x, y\}}{\mathcal{C}_z^{x|y}(t)\ \mathsf{linear}}$$

For instance, the terms $\mathcal{W}_x(x)$ and $\lambda x.x\ x$ are not linear. However, the latter can be represented in the $\lambda\mathsf{lxr}$-calculus by the linear term $\lambda x.\mathcal{C}_x^{y|z}(y\ z)$. More generally, every λ-term can be represented by a linear $\lambda\mathsf{lxr}$-term (cf. Section 4). Note that being linear is a property of α-equivalent classes, i.e. given two α-equivalent terms, either both are linear or both are not.

Notice 1 *Using α-equivalence we can now consider Barendregt's convention that no variable is free and bound in a term, without loss of generality.*

2.2 Congruence and meta-notations

As mentioned in the introduction, β-reduction can be decomposed into cut-elimination steps in proof-nets, when the latter are considered modulo some equations [DCG99]. Similarly, some equations handling the operators of $\lambda\mathsf{lxr}$ equip our calculus with a fine-grained notion of rewriting modulo, which allows the decomposition of β-reduction as well, but also in the untyped case (cf. Section 4). Presented in Figure 1, these equations are also, in the typed case, at the center of the correspondence with the proof-nets modulo (cf. Section 3).

The reader may notice that the equation $\mathsf{A_c}$ needs the side condition $(x \neq y, v)$, while $\mathsf{P_{cs}}$ needs $(x \neq y)$; but they can always be satisfied by Barendregt's convention.

The equations $\mathsf{A_c}$ and $\mathsf{C_c}$ express the internal associativity and commutativity of contraction, when seen as a binary operation merging two "wires" labelled with its two bound variables into one labelled by its free variable. The equations $\mathsf{P_c}, \mathsf{P_w}, \mathsf{P_s}$ express the permutability of independent contractions, weakenings, and substitutions, respectively. The point of the equation $\mathsf{P_{cs}}$, expressing the permutability between independent contraction and substitution, is discussed in the next sub-section.

$$
\begin{array}{lll}
\mathcal{C}_w^{x|v}(\mathcal{C}_x^{z|y}(t)) & \equiv_{\mathsf{A_c}} & \mathcal{C}_w^{z|x}(\mathcal{C}_x^{y|v}(t)) \\
\mathcal{C}_x^{y|z}(t) & \equiv_{\mathsf{C_c}} & \mathcal{C}_x^{z|y}(t) \\
\mathcal{C}_{x'}^{y'|z'}(\mathcal{C}_x^{y|z}(t)) & \equiv_{\mathsf{P_c}} & \mathcal{C}_x^{y|z}(\mathcal{C}_{x'}^{y'|z'}(t)) \text{ if } x \neq y', z' \ \&\ x' \neq y, z \\
\mathcal{W}_x(\mathcal{W}_y(t)) & \equiv_{\mathsf{P_w}} & \mathcal{W}_y(\mathcal{W}_x(t)) \\
t\langle x\backslash u\rangle\langle y\backslash v\rangle & \equiv_{\mathsf{P_s}} & t\langle y\backslash v\rangle\langle x\backslash u\rangle \quad \text{if } y \notin \mathcal{FV}(u) \ \&\ x \notin \mathcal{FV}(v) \\
\mathcal{C}_w^{y|z}(t)\langle x\backslash u\rangle & \equiv_{\mathsf{P_{cs}}} & \mathcal{C}_w^{y|z}(t\langle x\backslash u\rangle) \quad \text{if } x \neq w \ \&\ y, z \notin \mathcal{FV}(u)
\end{array}
$$

Figure 1. Congruence equations for λlxr-terms

We define the relation $\equiv$ as the smallest congruence on terms (i.e. a symmetric, reflexive, transitive, context-closed relation [Ter03]) that contains the equations of Figure 1. It can easily be proved that $\equiv$ 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 this size that the congruence rules generate finitely many possibilities to pick up a representative of the class.

We use Φ, Υ, Σ, Ψ, Ξ, $\Omega, \ldots$ to denote finite *lists* of variables (with no repetition). The notation Φ, Ψ denote the concatenation of Φ and Ψ, and we always suppose in that case that no variable appears in both Φ and Ψ. The following renaming operation will be also used when necessary to ensure linearity.

Definition 4 (Renaming Operation). *If $\Phi = x_1, \ldots, x_n$ and $\Psi = y_1, \ldots, y_n$ are two lists, we define the* renaming operation *of Φ by Ψ on a term t, written $\mathcal{R}_\Psi^\Phi(t)$, as the capture-avoiding simultaneous substitution of y_i for every* free *occurrence of x_i in t where $i \in 1 \ldots n$.*

Thus for instance $\mathcal{R}_{x',y'}^{x,y}(\mathcal{C}_w^{y|z}(x\ (y\ z))) = \mathcal{C}_w^{y|z}(x'\ (y\ z))$ (y is not replaced since the occurence is bound). We remark that for any permutation π of $1 \ldots n$, we have $\mathcal{R}_\Psi^\Phi(t) = \mathcal{R}_{\pi(\Psi)}^{\pi(\Phi)}(t)$.

We use the notation $\mathcal{W}_{x_1,\ldots,x_n}(t)$ for $\mathcal{W}_{x_1}(\ldots \mathcal{W}_{x_n}(t))$ and $\mathcal{C}_{x_1,\ldots,x_n}^{y_1,\ldots,y_n|z_1,\ldots,z_n}(t)$ for $\mathcal{C}_{x_1}^{y_1|z_1}(\ldots \mathcal{C}_{x_n}^{y_n|z_n}(t))$, where $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$ are all disctinct variables. In the case of the empty list, we define $\mathcal{W}_\emptyset(t) = t$ and $\mathcal{C}_\emptyset^{\emptyset|\emptyset}(t) = t$.

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

Notice 2 *Sometimes we use a set of variables, e.g. $\mathcal{S}$, in places where lists are expected, as in $\mathcal{W}_\mathcal{S}(u)$, $\mathcal{C}_\mathcal{S}^{\Phi|\Psi}(t)$, $\mathcal{R}_\Phi^\mathcal{S}(t)$ or $\Phi := \mathcal{S}$. The intended list is obtained by ordering $\mathcal{S}$ according to the total order that we have on the set of variables. These notations introduce no ambiguity and are much more legible.*

2.3 Operational semantics

Reduction rules and Reduction relations The reduction relation of the calculus is the relation generated by the reduction rules in Figure 2 modulo the congruence relation in Figure 1, as described below in detail.

B	$(\lambda x.t)\ u$	$\to$	$t\langle x\backslash u\rangle$	
System x				
Abs	$(\lambda y.t)\langle x\backslash u\rangle$	$\to$	$\lambda y.t\langle x\backslash u\rangle$	
App1	$(t\ v)\langle x\backslash u\rangle$	$\to$	$t\langle x\backslash u\rangle\ v$	$x \notin \mathcal{FV}(v)$
App2	$(t\ v)\langle x\backslash u\rangle$	$\to$	$t\ v\langle x\backslash u\rangle$	$x \notin \mathcal{FV}(t)$
Var	$x\langle x\backslash u\rangle$	$\to$	u	
Weak1	$\mathcal{W}_x(t)\langle x\backslash u\rangle$	$\to$	$\mathcal{W}_{\mathcal{FV}(u)}(t)$	
Weak2	$\mathcal{W}_y(t)\langle x\backslash u\rangle$	$\to$	$\mathcal{W}_y(t\langle x\backslash u\rangle)$	$x \neq y$
Cont	$\mathcal{C}_x^{y\|z}(t)\langle x\backslash u\rangle$	$\to$	$\mathcal{C}^{\Psi\|\Upsilon}_{\mathcal{FV}(u)}(t\langle y\backslash\mathcal{R}_{\Psi}^{\mathcal{FV}(u)}(u)\rangle\langle z\backslash\mathcal{R}_{\Upsilon}^{\mathcal{FV}(u)}(u)\rangle)$	where Ψ, Υ are *fresh*
Comp	$t\langle y\backslash v\rangle\langle x\backslash u\rangle$	$\to$	$t\langle y\backslash v\langle x\backslash u\rangle\rangle$	$x \notin \mathcal{FV}(t)\setminus\{y\}$
System r				
WAbs	$\lambda x.\mathcal{W}_y(t)$	$\to$	$\mathcal{W}_y(\lambda x.t)$	$x \neq y$
WApp1	$\mathcal{W}_y(u)\ v$	$\to$	$\mathcal{W}_y(uv)$	
WApp2	$u\ \mathcal{W}_y(v)$	$\to$	$\mathcal{W}_y(uv)$	
WSubs	$t\langle x\backslash\mathcal{W}_y(u)\rangle$	$\to$	$\mathcal{W}_y(t\langle x\backslash u\rangle)$	
Merge	$\mathcal{C}_w^{y\|z}(\mathcal{W}_y(t))$	$\to$	$\mathcal{R}_w^z(t)$	
Cross	$\mathcal{C}_w^{y\|z}(\mathcal{W}_x(t))$	$\to$	$\mathcal{W}_x(\mathcal{C}_w^{y\|z}(t))$	$x \neq y,\ \ x \neq z$
CAbs	$\mathcal{C}_w^{y\|z}(\lambda x.t)$	$\to$	$\lambda x.\mathcal{C}_w^{y\|z}(t)$	
CApp1	$\mathcal{C}_w^{y\|z}(t\ u)$	$\to$	$\mathcal{C}_w^{y\|z}(t)\ u$	$y, z \notin \mathcal{FV}(u)$
CApp2	$\mathcal{C}_w^{y\|z}(t\ u)$	$\to$	$t\ \mathcal{C}_w^{y\|z}(u)$	$y, z \notin \mathcal{FV}(t)$
CSubs	$\mathcal{C}_w^{y\|z}(t\langle x\backslash u\rangle)$	$\to$	$t\langle x\backslash\mathcal{C}_w^{y\|z}(u)\rangle$	$y, z \notin \mathcal{FV}(t)\setminus\{x\}$

Figure 2. Reduction rules for $\lambda\mathsf{lxr}$-terms

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 $B\mathsf{xr}$). The *basic* reduction relation $\longrightarrow_{i_b}$ is the contextual closure of the relation formed by instances of the rules in the set i. Moreover, since we have a congruence relation on terms, we denote by $\longrightarrow_i$ the reduction relation modulo this congruence [Ter03], that is, $t \longrightarrow_i t'$ if and only if there exist two terms u and u' such that $t \equiv u \longrightarrow_{i_b} u' \equiv t'$. Hence, the most general reduction relation of our calculus is $\longrightarrow_{B\mathsf{xr}}$ (i.e. generated by the rules of $B\mathsf{xr}$), often written $\longrightarrow_{\lambda\mathsf{lxr}}$ (i.e. pertaining to the calculus $\lambda\mathsf{lxr}$).

For any reduction relation $\to_j$, we denote by $\to_j^n$ the n^{th} composition of $\to_j$, we denote by $\to_j^+$ the transitive closure (union of all $\to_j^n$ for $n \geq 1$), we denote by $\to_j^*$ the reflexive and transitive closure (union of all $\to_j^n$ for $n \geq 0$) and by $\leftrightarrow_j^*$ the reflexive, transitive and symmetric closure.

General properties In order to avoid variable capture, the rules Abs and CAbs need the side-conditions ($y \notin \mathcal{FV}(u)$) and ($x \neq y, z$) respectively, which are satisfied if Barendregt's convention is respected, so they can always be satisfied by α-equivalence, so their nature is different from that of the other side-conditions appearing on the right-hand side of Figure 2. The rules should be understood in the prospect of applying them to linear terms. Indeed, linearity is preserved by the reduction relation, which satisfies the following properties:

Lemma 1 (Preservation Properties). *Let t be a linear term and $t \to_{\lambda\mathsf{lxr}} t'$.*

1. *The set of free variables is preserved, i.e. $\mathcal{FV}(t) = \mathcal{FV}(t')$.*
2. *Linearity is preserved, i.e. t' is linear.*

Proof. By using the fact that the congruence preserves free variables and linearity, the two properties have to be satisfied by the basic reduction relation. This can be checked by a straightforward simultaneous induction on the reduction step and case analysis.

For instance in rule Cont, it is the introduction of the lists of fresh variables Ψ and Υ that ensures the linearity of 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 $\lambda\mathsf{lxr}$ 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 bound variables of a term may either increase (cf. rule Cont) or decrease (cf. rules $\mathsf{Var}, \mathsf{Merge}, \mathsf{Weak1}, \ldots$).

The fact that linearity is preserved by congruence and reduction (Lemma 1-2) is a minimal requirement of the system.

Notice 3 *From now on we only consider linear terms.*

Role of the rules The B-rule is a key rule of $\lambda\mathsf{lxr}$ in that it reduces what is considered in the λ-calculus as a β-redex, and creates a linear substitution operator, as in $\lambda\mathsf{x}$ [BR95] but respecting the linearity constraints.

System x propagates and eliminates linear substitution operators, and duplication and erasure are controlled by the presence of contraction and weakening (rules Cont and $\mathsf{Weak1}$ respectively). Contraction and weakening can thus be seen as *resource* operators also called respectively *duplication* and *erasure* operators. Note that this only makes sense if the linearity constraints are satisfied; in this case a construct such as $y\langle x \backslash t\rangle$ is forbidden.

Lemma 2. *t is a x-normal form if and only if t has no closure.*

Proof. We first remark that if t has no closure, then clearly no x-rule can be applied. Conversely, for each linear substitution operator applied to a non-closure term there is a reduction rule.

An important property is that reducing terms by system xr implements an appropriate notion of meta-substitution on all $\lambda\mathsf{lxr}$-terms,whereas in $\lambda\mathsf{x}$ the notion of meta-substitution thus implemented applies *only* to closure-free ones. Thus, for example, $x\langle x'\backslash y\ \lambda z.z\rangle\langle y\backslash\lambda z.z\rangle$ does not reduces to $x\langle x'\backslash(\lambda z.z)\ \lambda z.z\rangle$ in $\lambda\mathsf{x}$ but it does in $\lambda\mathsf{lxr}$ thanks to our the notion of composition. We thus say that $\lambda\mathsf{lxr}$ enjoys the *full composition* property, as explicit substitution operators of the calculus implement exactly a notion of meta-substitution defined on the same calculus, that is, on terms having weakening, contraction and linear substitution operators.

Note that when linearity constraints are not considered, four cases may occur when composing two explicit substitutions as in $t\langle y\backslash v\rangle\langle x\backslash u\rangle$: 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) $x \notin \mathcal{FV}(t) \cup \mathcal{FV}(v)$.

In calculi like λ_{ws} [DG01] only cases (1) and (3) are considered by the reduction rules, thus only yielding partial composition. Because of the linearity constraints of $\lambda\mathsf{lxr}$, 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 ($\mathsf{Weak1}$) and (Cont), respectively. Case (3) is treated by rule (Comp), and case (2) by the congruence rule $\mathsf{P_s}$. More precisely, the congruence rule can be applied to swap the substitutions, thus allowing the evaluation of the external substitution $\langle x\backslash u\rangle$ without forcing the internal one to be evaluated first. Indeed, all cases (1)-(4) are treated in $\lambda\mathsf{lxr}$, thus yielding a full notion of composition of substitutions.

The linearity constraints are *essential* for composition: if they are not taken into account, the composition rule Comp causes failure of the PSN and strong normalisation properties [BG99]. Hence, it is because of the presence of weakenings and contractions, combined with the linearity constraints, that the notion of composition in $\lambda\mathsf{lxr}$ is full. Thus, $\lambda\mathsf{lxr}$ turns out to be the first term calculus with linear substitution operators having full composition and preserving β-strong normalisation (Corollary 3).

Respectively viewed as duplication and erasure operators, contraction and weakening play a very special role with respect to optimisation issues. In a term, the further down a contraction $\mathcal{C}_x^{y|z}(\_)$ lies, the later a linear substitution operator on x will be duplicated in its propagation process by system x. Symmetrically, the further up a weakening $\mathcal{W}_x(\_)$ lies, the sooner a substitution on x, called a *void* substitution, will be erased. For instance, if $y, z \in \mathcal{FV}(t_2)$, we have $\mathcal{C}_x^{y|z}(t_1\ t_2\ t_3)\langle x\backslash\lambda x'.x'\rangle \longrightarrow_{\mathsf{x}}^5 t_1\ t_2\langle y\backslash\lambda x'.x'\rangle\langle z\backslash\lambda x'.x'\rangle\ t_3$ but

$(t_1\ \mathcal{C}^{y|z}_x(t_2)\ \ t_3)\langle x\backslash\lambda x'.x'\rangle\!\longrightarrow^3_{\mathsf{x}}\ t_1\ t_2\langle y\backslash\lambda x'.x'\rangle\langle z\backslash\lambda x'.x'\rangle\ \ t_3$, so $t_1\ \mathcal{C}^{y|z}_x(t_2)\ \ t_3$ is in a sense more optimised than $\mathcal{C}^{y|z}_x(t_1\ t_2\quad t_3)$. Symmetrically, we have $(t_1\ \mathcal{W}_x(t_2)\ t_3)\langle x\backslash\lambda x'.x'\rangle\!\longrightarrow^3_{\mathsf{x}}\ t_1\ t_2\ t_3$ but $\mathcal{W}_x(t_1\ t_2\ t_3)\langle x\backslash\lambda x'.x'\rangle\!\longrightarrow^1_{\mathsf{x}}\ t_1\ t_2\ t_3$, so $\mathcal{W}_x(t_1\ t_2\ t_3)$ is in a sense more optimised than $t_1\ \mathcal{W}_x(t_2)\ t_3$.

System r optimises terms by pushing down contractions and pulling up weakenings, so that they reach *canonical places* in $\lambda\mathsf{lxr}$-terms (also using $\mathsf{Weak2}$ and the left to right direction of the equation $\mathsf{P_{cs}}$). Such a place for a contraction $\mathcal{C}^{y|z}_x(\_)$ is just above an application or a closure, with y and z in distinct sides (i.e. $\mathcal{C}^{y|z}_x(t\ u)$ or $\mathcal{C}^{y|z}_x(t\langle x'\backslash u\rangle)$ with $y \in \mathcal{FV}(t)$ and $z \in \mathcal{FV}(u)$ or *vice versa*). The canonical place for a weakening $\mathcal{W}_x(\_)$ is either at the top-level of a term or just below a binder on x (i.e. $\lambda x.\mathcal{W}_x(t)$ or $\mathcal{W}_x(t)\langle x\backslash u\rangle$).

For closure-free terms, these constructs are just $\mathcal{C}^{y|z}_x(t\ u)$ (with $y \in \mathcal{FV}(t)$ and $z \in \mathcal{FV}(u)$ or *vice versa*) and either $\lambda x.\mathcal{W}_x(t)$ or $\mathcal{W}_x(t)$ at the top-level. In that case, the rules CSubs and WSubs and the right to left direction of the equation $\mathsf{P_{cs}}$ are not needed to place contractions and weakenings in canonical places. Removing these rules and orienting $\mathsf{P_{cs}}$ from left to right as a rule of system x would yield a system for which most of the results of this paper would hold (but not optimising as much terms with closures); in particular, x would still eliminate linear substitution operators and implement the same notion of implicit substitution and β-reduction could still be simulated (cf. Theorem 7).

2.4 Termination of xr

It is clear that rule B will be used to simulate β-reduction. The rules of system xr handle the operators that we have introduced, and a minimal requirement for those rules is to induce a terminating system. We shall also see in Section 5 that xr is confluent.

The use of resource operators allows 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. This idea inspires the notion of *multiplicity* of the substituted variable:

Definition 5 (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. Supposing that $x \neq y, x \neq z, x \neq w$,*

$$
\begin{array}{llll}
\mathcal{M}_x(x) := 1 & \mathcal{M}_x(t\langle y\backslash u\rangle) := \mathcal{M}_x(t) & & \textit{if } x \in FV(t)\setminus\{y\}\\
\mathcal{M}_x(\lambda y.t) := \mathcal{M}_x(t) & \mathcal{M}_x(t\langle y\backslash u\rangle) := \mathcal{M}_y(t)\cdot(\mathcal{M}_x(u)+1) & & \textit{if } x \in FV(u)\\
\mathcal{M}_x(\mathcal{W}_x(t)) := 1 & \mathcal{M}_x((t\ u)) := \mathcal{M}_x(t) & & \textit{if } x \in FV(t)\\
\mathcal{M}_x(\mathcal{W}_y(t)) := \mathcal{M}_x(t) & \mathcal{M}_x((t\ u)) := \mathcal{M}_x(u) & & \textit{if } x \in FV(u)\\
 & \mathcal{M}_x(\mathcal{C}^{z|w}_x(t)) := \mathcal{M}_z(t)+\mathcal{M}_w(t)+1 & & \\
 & \mathcal{M}_x(\mathcal{C}^{z|w}_y(t)) := \mathcal{M}_x(t) & &
\end{array}
$$

Roughly, this notion corresponds to the number of occurrences of a variable in a λlxr-term when translated to its corresponding λ-term free from linearity constraints and resource operators (see Section 4 for details), but we add a twist to this concept (+1 in the second case for closure and the first case for contraction in the definition above), so that the following notion of *term complexity*, which weighs the complexity of a sub-term in a substitution with the multiplicity of the substituted variable, is decreased by reductions (Lemma 3).

Definition 6 (Term complexity). *We define the notion of term complexity by induction on terms as follows:*

$$\begin{array}{llll} \mathcal{S}(x) & := 1 & \mathcal{S}(t\langle x\backslash u\rangle) & := \mathcal{S}(t) + \mathcal{M}_x(t)\cdot\mathcal{S}(u) \\ \mathcal{S}(\lambda x.t) & := \mathcal{S}(t) & \mathcal{S}(t\ u) & := \mathcal{S}(t) + \mathcal{S}(u) \\ \mathcal{S}(\mathcal{W}_x(t)) & := \mathcal{S}(t) & \mathcal{S}(\mathcal{C}_x^{y|z}(t)) & := \mathcal{S}(t) \end{array}$$

Remark 1. The notions of multiplicity and term complexity are invariant under conversion by $\equiv$.

We have now to show that the term complexity does not increase during xr-reduction. In particular, the term complexity strictly decreases for some rules and it remains equal for others. This relies on the fact that the multiplicities cannot increase.

Lemma 3 (Decrease of multiplicities and term complexities).

- *If $t \to_{\mathsf{xr}} u$, then for all $w \in \mathcal{FV}(t)$, $\mathcal{M}_w(t) \geq \mathcal{M}_w(u)$.*
- *If $t \to_{\mathsf{xr}} u$, then $\mathcal{S}(t) \geq \mathcal{S}(u)$. Moreover, if $t \to_{\mathsf{Var,Weak1,Cont,Comp}} u$, then $\mathcal{S}(t) > \mathcal{S}(u)$.*

Proof. Both points are proved by case analysis and induction on t. The first one uses the fact that that $\mathcal{M}_x(t) \geq 1$ (provided $x \in FV(t)$), the second one relies on the first. See the appendix for details.

Note that this does not hold for rule B. For instance,
$t = (\lambda x.\mathcal{C}_x^{x_1|x_2}(x_1\ x_2))\ \lambda y.\mathcal{C}_y^{y_1|y_2}(y_1\ y_2) \to_B \mathcal{C}_x^{x_1|x_2}(x_1\ x_2)\langle x\backslash \lambda y.\mathcal{C}_y^{y_1|y_2}(y_1\ y_2)\rangle = u$
but $\mathcal{S}(t) = 4$ and $\mathcal{S}(u) = 8$.

We now use another measure to show the termination of the subsystem of xr containing only the rules that might not decrease the term complexity.

Definition 7. *We define an interpretation $\mathcal{I}(\_)$ from λlxr-terms to natural numbers as follows:*

$$\begin{array}{llll} \mathcal{I}(x) & := 2 & \mathcal{I}(t\langle x\backslash u\rangle) & := \mathcal{I}(t)\cdot(\mathcal{I}(u)+1) \\ \mathcal{I}(\lambda x.t) & := 2\cdot\mathcal{I}(t)+2 & \mathcal{I}(t\ u) & := 2\cdot(\mathcal{I}(t)+\mathcal{I}(u))+2 \\ \mathcal{I}(\mathcal{W}_x(t)) & := \mathcal{I}(t)+1 & \mathcal{I}(\mathcal{C}_x^{y|z}(t)) & := 2\cdot\mathcal{I}(t) \end{array}$$

Remark 2. The interpretation $\mathcal{I}(\_)$ is invariant under conversion by $\equiv$.

See the appendix for details.

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

Proof. By case analysis and induction on t, using the fact that for any term t, $\mathcal{I}(t) \geq 2$. See the appendix for details.

We can conclude this section with the following property

Theorem 1. *The system* xr *is terminating.*

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

2.5 Typing Rules

In this section we present the *simply-typed* λlxr-calculus. The typing system ensures strong normalisation (as in the λ-calculus) and also linearity. *Types* are defined by the following grammar, where σ ranges over a countable set of atomic types.

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

An *environment* Γ is a finite mapping from variables to types, so that it can be seen as a finite set of pairs $x : A$. We use the standard notion of *domain* of an environment Γ, written $\mathbf{d}(\Gamma)$. We write Γ, Δ to denote the disjoint and consistent union of the environments Γ and Δ. A *judgement* is an object of the form $\Gamma \vdash t : A$, where Γ is an environment, t is a λlxr-term, and A is a type.

The *typing rules* of the simply-typed λlxr-calculus are shown in Figure 3. *Derivations* of *typed* terms, a.k.a. *proofs*, are the trees built (as usual, see [TS00]) from these rules.

Remark 3. Note that a judgement $\Gamma \vdash t : A$ can be the root of at most one derivation (up to renaming, in sub-derivations, of the variables bound in t), which can be reconstructed using the structure of the term t (hence the notion of *proof-term*).

Remark that $\Gamma \vdash t : A$ implies that $\mathbf{d}(\Gamma) = \mathcal{FV}(t)$. Renaming is sound with respect to typing, as shown by the following result of admissibility, in the standard sense [TS00].

Lemma 5. *The following rule is admissible in the typing system of* λlxr *(we use dashed lines to emphasise their admissibility):*

$$\frac{\Gamma, \Delta \vdash t : A}{\mathcal{R}_{\Phi}^{\mathbf{d}(\Gamma)}(\Gamma), \Delta \vdash \mathcal{R}_{\Phi}^{\mathbf{d}(\Gamma)}(t) : A}$$

where $\mathcal{R}_{\Phi}^{\Psi}(\{x_1 : A_1, \ldots, x_n : A_n\}) := \{\mathcal{R}_{\Phi}^{\Psi}(x_1) : A_1, \ldots, \mathcal{R}_{\Phi}^{\Psi}(x_n) : A_n\}$.

The following rules are derivable in the typing system of λlxr *(we use double lines to emphasise their derivability.):*

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

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

$$\frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash \mathcal{C}_z^{x|y}(M) : B} \ (Cont) \qquad \frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash \mathcal{W}_x(t) : A} \ (Weak)$$

Figure 3. Typing Rules for λlxr-terms

$$\frac{\Delta \vdash t : A}{\Gamma, \Delta \vdash \mathcal{W}_{\mathbf{d}(\Gamma)}(t) : A} \qquad \frac{\mathcal{R}_{\Phi}^{\mathbf{d}(\Gamma)}(\Gamma), \mathcal{R}_{\Psi}^{\mathbf{d}(\Gamma)}(\Gamma), \Delta \vdash t : A}{\Gamma, \Delta \vdash \mathcal{C}_{\mathbf{d}(\Gamma)}^{\Phi|\Psi}(t) : A}$$

Proof. The admissibility of the first rule is proved by a routine induction on $|t|$. For the next two rules an induction on the cardinal of $\mathbf{d}(\Gamma)$ suffices.

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 $s \longrightarrow_{\lambda\mathsf{lxr}} s'$, then $\Gamma \vdash s' : A$.*

Proof. The proof of the first point is straightforward, based on checking that it holds for all equations defining $\equiv$. Using the first point leaves only the basic reduction to be checked in the second point. This is also straightforward and proved by induction on the reduction step and by case analysis. Using remark 3 we recompose from the hypothesis $\Gamma \vdash s : A$ the last steps of its derivation and rearrange the sub-derivations to conclude $\Gamma \vdash s' : A$ as follows:

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

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

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

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

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

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

– (App2): Similar to the previous case.

– (Var): We have $s = x\langle x\backslash u\rangle$ and $s' = u$.

$$\dfrac{\dfrac{}{x : A \vdash x : A} \qquad \Gamma \vdash u : A}{\Gamma \vdash x\langle x\backslash u\rangle : A} \qquad \Gamma \vdash u : A$$

– (Weak1): We have $s = \mathcal{W}_x(t)\langle x\backslash u\rangle$ and $s' = \mathcal{W}_{\mathcal{FV}(u)}(t)$.

$$\dfrac{\dfrac{\Gamma \vdash t : A}{\Gamma, x : B \vdash \mathcal{W}_x(t) : A} \qquad \Delta \vdash u : B}{\Gamma, \Delta \vdash \mathcal{W}_x(t)\langle x\backslash u\rangle : A} \qquad \dfrac{\Gamma \vdash t : A}{\Gamma, \Delta \vdash \mathcal{W}_{\mathcal{FV}(u)}(t) : A}$$

since $\mathbf{d}(\Delta) = \mathcal{FV}(u)$.

– (Weak2): We have $s = \mathcal{W}_y(t)\langle x\backslash u\rangle$ and $s' = \mathcal{W}_y(t\langle x\backslash u\rangle)$ with $x \neq y$.

$$\dfrac{\dfrac{\Gamma, x : B \vdash t : A}{\Gamma, y : C, x : B \vdash \mathcal{W}_y(t) : A} \qquad \Delta \vdash u : B}{\Gamma, y : C, \Delta \vdash \mathcal{W}_y(t)\langle x\backslash u\rangle : A} \qquad \dfrac{\dfrac{\Gamma, x : B \vdash t : A \qquad \Delta \vdash u : B}{\Gamma, \Delta \vdash t\langle x\backslash u\rangle : A}}{\Gamma, y : C, \Delta \vdash \mathcal{W}_y(t\langle x\backslash u\rangle) : A}$$

– (Cont): $s = \mathcal{C}_x^{y|z}(t)\langle x\backslash v\rangle$ and $s' = \mathcal{C}_{\mathcal{FV}(v)}^{\Phi|\Sigma}(t\langle y\backslash \mathcal{R}_\Phi^{\mathcal{FV}(v)}(v)\rangle\langle z\backslash \mathcal{R}_\Sigma^{\mathcal{FV}(v)}(v)\rangle)$.

$$\dfrac{\dfrac{\Gamma, y : B, z : B \vdash t : A}{\Gamma, x : B \vdash \mathcal{C}_x^{y|z}(t) : A} \qquad \Delta \vdash v : B}{\Gamma, \Delta \vdash \mathcal{C}_x^{y|z}(t)\langle x\backslash v\rangle : A}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, y : B, z : B \vdash t : A \qquad \dfrac{\Delta \vdash v : B}{\mathcal{R}_\Phi^{\mathbf{d}(\Delta)}(\Delta) \vdash \mathcal{R}_\Phi^{\mathcal{FV}(v)}(v) : B}}{\Gamma, z : B, \mathcal{R}_\Phi^{\mathbf{d}(\Delta)}(\Delta) \vdash t\langle y\backslash \mathcal{R}_\Phi^{\mathcal{FV}(v)}(v)\rangle : A} \qquad \dfrac{\Delta \vdash v : B}{\mathcal{R}_\Sigma^{\mathbf{d}(\Delta)}(\Delta) \vdash \mathcal{R}_\Sigma^{\mathcal{FV}(v)}(v) : B}}{\Gamma, \mathcal{R}_\Phi^{\mathbf{d}(\Delta)}(\Delta), \mathcal{R}_\Sigma^{\mathbf{d}(\Delta)}(\Delta) \vdash t\langle y\backslash \mathcal{R}_\Phi^{\mathcal{FV}(v)}(v)\rangle\langle z\backslash \mathcal{R}_\Sigma^{\mathcal{FV}(v)}(v)\rangle : A}}{\Gamma, \Delta \vdash \mathcal{C}_{\mathcal{FV}(v)}^{\Phi|\Sigma}(t\langle y\backslash \mathcal{R}_\Phi^{\mathcal{FV}(v)}(v)\rangle\langle z\backslash \mathcal{R}_\Sigma^{\mathcal{FV}(v)}(v)\rangle) : A}$$

since $\mathbf{d}(\Delta) = \mathcal{FV}(v)$.

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

$$\frac{\dfrac{\Gamma, y : C \vdash t : A \qquad \Delta, x : B \vdash u : C}{\Gamma, \Delta, x : B \vdash t\langle y\backslash u\rangle : A} \qquad \Pi \vdash v : B}{\Gamma, \Delta, \Pi \vdash t\langle y\backslash u\rangle\langle x\backslash v\rangle : A} \qquad \frac{\Gamma, y : C \vdash t : A \qquad \dfrac{\Delta, x : B \vdash u : C \qquad \Pi \vdash v : B}{\Delta, \Pi \vdash u\langle x\backslash v\rangle : C}}{\Gamma, \Delta, \Pi \vdash t\langle y\backslash u\langle x\backslash v\rangle\rangle : A}$$

– (WAbs): We have $s = \mathcal{W}_y(\lambda x.t)$ and $s' = \lambda x.\mathcal{W}_y(t)$.

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

– (WApp1): We have $s = \mathcal{W}_y(u)\ v$ and $s' = \mathcal{W}_y(uv)$.

$$\frac{\dfrac{\Gamma \vdash u : B \to C}{\Gamma, y : D \vdash \mathcal{W}_y(u) : B \to C} \qquad \Delta \vdash v : B}{\Gamma, y : D, \Delta \vdash \mathcal{W}_y(u)\ v : C} \qquad \frac{\dfrac{\Gamma \vdash u : B \to C \qquad \Delta \vdash v : B}{\Gamma, \Delta \vdash u\ v : C}}{\Gamma, y : D, \Delta \vdash \mathcal{W}_y(uv) : C}$$

– (WApp2): Similar to the previous case.
– (WSubs): We have $s = t\langle x\backslash\mathcal{W}_y(u)\rangle$ and $s' = \mathcal{W}_y(t\langle x\backslash u\rangle)$.

$$\frac{\dfrac{\Gamma \vdash u : B}{\Gamma, y : C \vdash \mathcal{W}_y(u) : B} \qquad \Delta, x : B \vdash t : A}{\Gamma, y : C, \Delta \vdash t\langle x\backslash\mathcal{W}_y(u)\rangle : A} \qquad \frac{\dfrac{\Gamma \vdash u : B \qquad \Delta, x : B \vdash t : A}{\Gamma, \Delta \vdash t\langle x\backslash u\rangle : A}}{\Gamma, y : C, \Delta \vdash \mathcal{W}_y(t\langle x\backslash u\rangle) : A}$$

– (Merge): $s = \mathcal{C}_w^{y|z}(\mathcal{W}_y(t))$ and $s' = \mathcal{R}_w^z(t)$.

$$\frac{\dfrac{\Gamma, z : C \vdash t : A}{\Gamma, y : C, z : C \vdash \mathcal{W}_y(t) : A}}{\Gamma, w : C \vdash \mathcal{C}_w^{y|z}(\mathcal{W}_y(t)) : A} \qquad \begin{array}{c}\Gamma, z : C \vdash t : A \\ \hdashline \Gamma, w : C \vdash \mathcal{R}_w^z(t) : A\end{array}$$

– (Cross): $s = \mathcal{C}_w^{y|z}(\mathcal{W}_x(t))$ and $s' = \mathcal{W}_x(\mathcal{C}_w^{y|z}(t))$.

$$\frac{\dfrac{\Gamma, y : C, z : C \vdash t : A}{\Gamma, y : C, z : C, x : B \vdash \mathcal{W}_x(t) : A}}{\Gamma, w : C, x : B \vdash \mathcal{C}_w^{y|z}(\mathcal{W}_x(t)) : A} \qquad \frac{\dfrac{\Gamma, y : C, z : C \vdash t : A}{\Gamma, w : C \vdash \mathcal{C}_w^{y|z}(t) : A}}{\Gamma, w : C, x : B \vdash \mathcal{W}_x(\mathcal{C}_w^{y|z}(t)) : A}$$

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

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

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

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

- (CApp2): Similar to the previous case.
- (CSubs): $s = \mathcal{C}^{y|z}_w(t\langle x\backslash u\rangle)$ and $s' = t\langle x\backslash \mathcal{C}^{y|z}_w(u)\rangle$.

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

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 simply-typed terms, which is achieved by translating simply-typed λlxr-terms to MELL proof-nets. MELL decomposes the intuitionistic logical connectives into more elementary connectives, such as the linear implication and the exponentials, thus providing a more refined use of resources than the one available in Intuitionistic Logic or Classical Logic. Proof-nets provide a geometric interpretation of proofs, thus keeping only the logical part of the structure of proofs and forgetting the structural details.

Theorem 5 and Theorem 6 show that the translation from λlxr to proof-nets is sound and 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] or [Laf95] for more details.

Let $\mathcal{A}$ be a set of *atomic formulae* containing positive atoms p and negative atoms $p^\perp$. 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 the linear version of the classical disjunction, whereas $?\mathcal{F}$ and $!\mathcal{F}$ are used to indicate where contraction or weakening can take place.

Linear negation of formulae is defined by

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

The set of proof-nets, that we denote by PN, is defined inductively in Figure 4 where we use rectangles having rounded corners to denote already defined nets used in the inductive constructions.

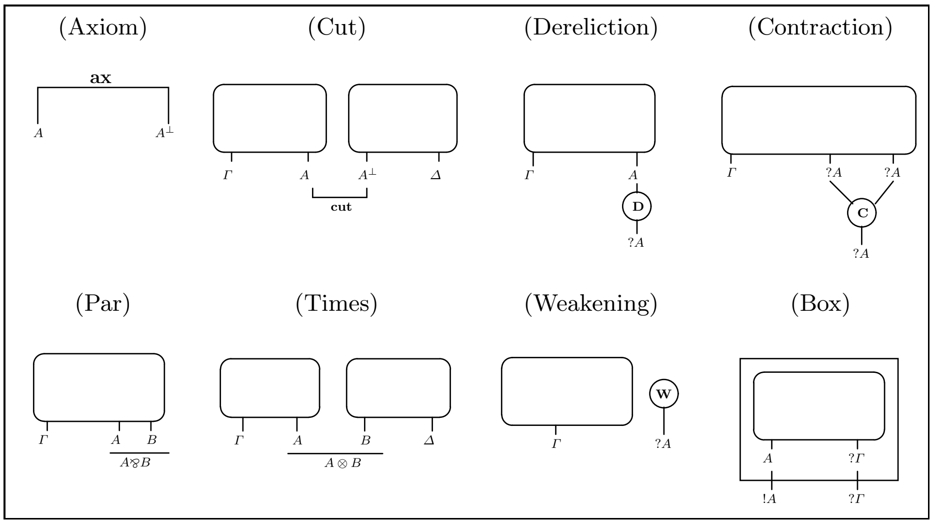
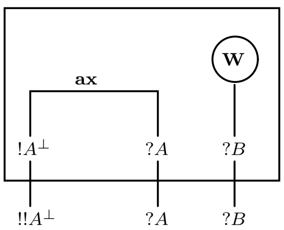
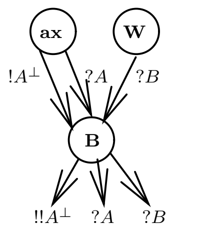


Figure 4. MELL Proof-nets

Similarly to term contexts [Ter03], proof-net *contexts* are proof-nets constructed by adding to Figure 4 the basic case of a special proof-net called *hole*.

A proof-net can be viewed as a finite acyclic oriented graph, where the nodes correspond to the alphabet $\{\mathbf{ax}, \mathbf{cut}, \otimes, \parr, \mathbf{C}, \mathbf{D}, \mathbf{W}, \mathbf{B}\}$. Each node has a number of $\mathtt{p}$ *(premise)* and $\mathtt{c}$ *(conclusion)* ports. Indeed, the **ax** node has two ports $\mathtt{c}$, the **cut** node two ports $\mathtt{p}$, the **D** node one port $\mathtt{p}$ and one port $\mathtt{c}$, the $\otimes$, $\parr$ and **C** nodes two ports $\mathtt{p}$ and one port $\mathtt{c}$, the **W** node one port $\mathtt{c}$ and the **B** node has n ports $\mathtt{p}$ and n ports $\mathtt{c}$ (for every $n \geq 1$). Each edge in the graph is decorated with a *type* and is connected to exactly one port $\mathtt{c}$ and at most one port $\mathtt{p}$, i.e. each edge has a *source* node but not necessarily a *goal* node [4]. The following picture shows an example of proof-net in standard graphical notation on the left, and its interpretation as oriented graph on the right.





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

Viewed as finite oriented graphs, the above example reads (inductively) as follows: the graph with one node **ax** which is the source of two edges decorated (respectively) by A and $A^{\perp}$ is a proof-net; the graph obtained by (1) taking two proof-nets n_1 and n_2, (2) adding a **cut** node and (3) adding two edges e_1 and e_2, decorated respectively with A and $A^{\perp}$, whose source is n_1 and n_2 respectively and whose goal is the **cut** node, is a proof-net.

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 5.

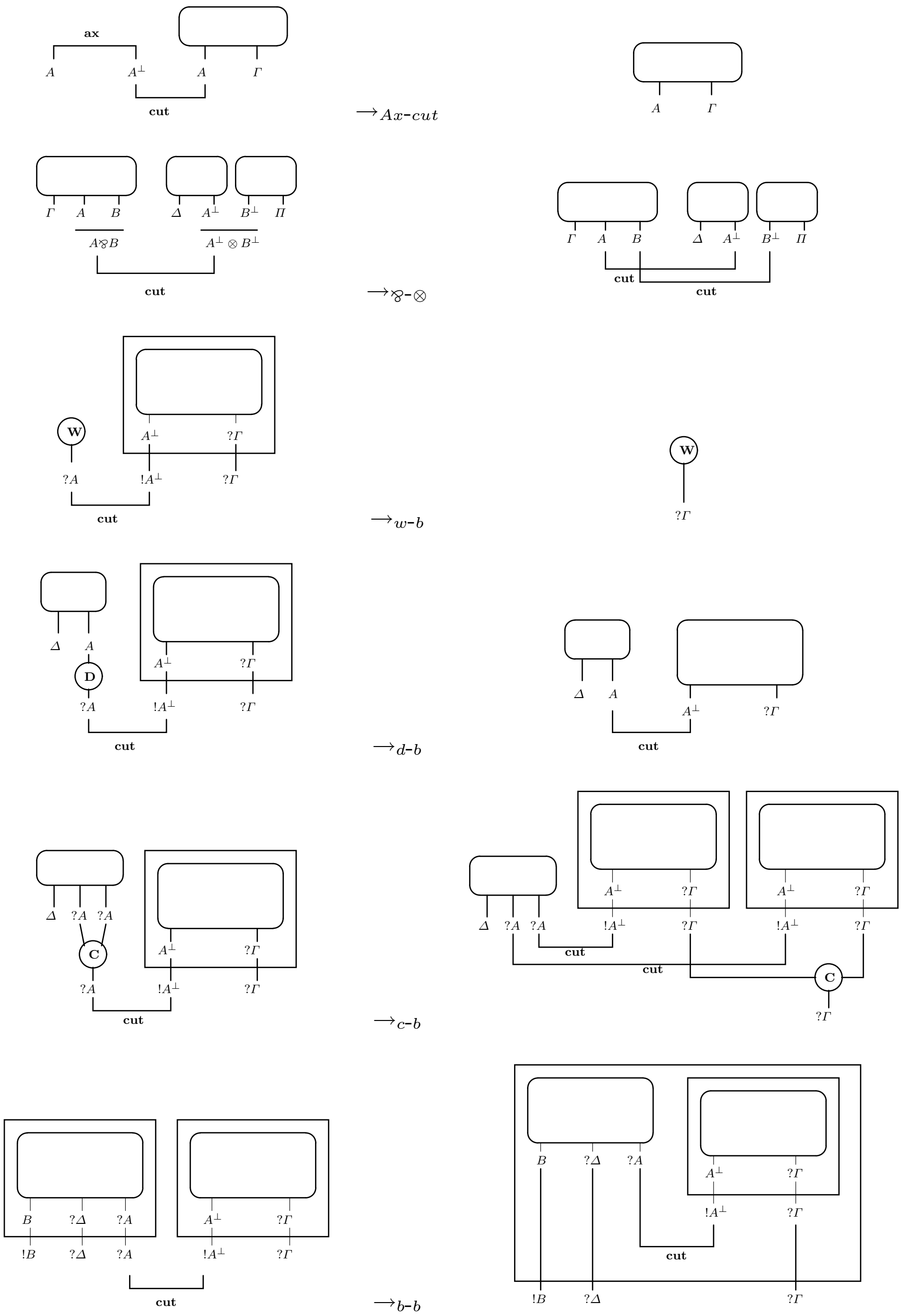


Figure 5. Cut elimination rules for MELL Proof-nets

Unfortunately, the original notion of reduction on PN is unsufficient to encode directly either the β rule of λ-calculus, or the rules dealing with propagation of substitution in explicit substitution calculi. For instance, this is prevented by the fact that the order in which contraction nodes are connected is still relevant in PN. One is thus led to define an equivalence relation on PN, as in [DCG99], where two equations $\sim_{\mathtt{A_c}}$ and $\sim_{\mathtt{P_{cb}}}$ are introduced (see Figure 6).

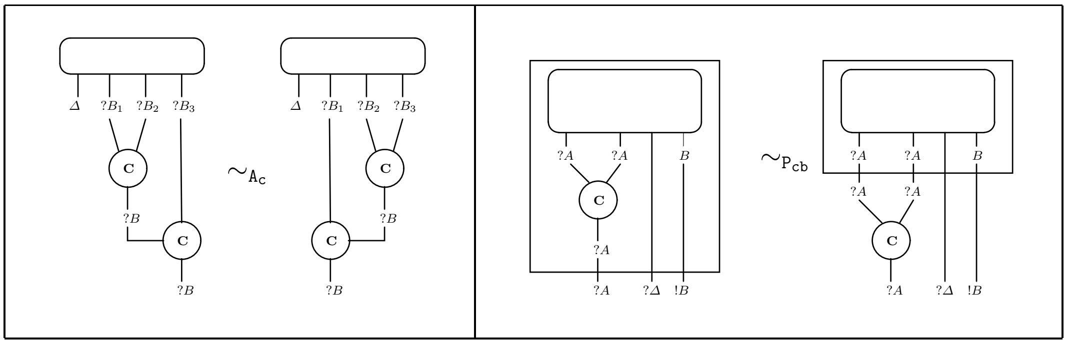


Figure 6. Equations for MELL proof-nets

Equivalence $\mathtt{A_c}$ turns contraction into an associative operator, and corresponds to forgetting the order in which the contraction rule is used. Equivalence $\mathtt{P_{cb}}$ 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 use the two extra reduction rules in Figure 7 : $\mathtt{U}$ is used to simplify weakening linked to contraction nodes and $\mathtt{V}$ allows weakening links to go outside boxes in order to bring them together at the top of the proof-nets.

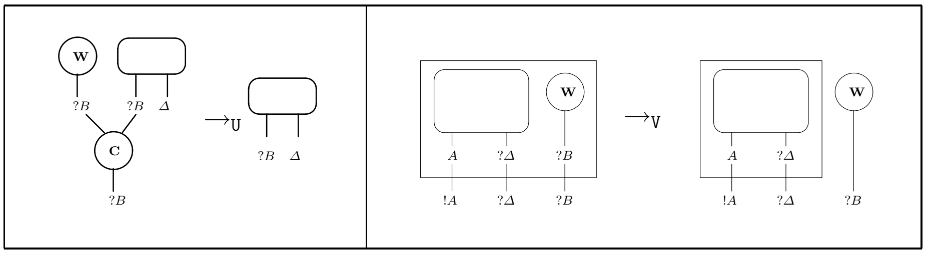


Figure 7. Extra reduction rules for MELL proof-nets

The reader may check that all these rules and equations preserve types as well as well-formedness of proof-nets.

Notation: Henceforth, we shall call R the set of rules *Ax-cut*, $\parr$-$\otimes$, *w-b*, *d-b*, *c-b*, *b-b*, $\mathtt{U}$ and $\mathtt{V}$ and E the set of equations $\mathtt{A_c}$ and $\mathtt{P_{cb}}$. We shall write $\sim_E$ for the congruence (reflexive, symmetric, transitive, closed by proof-net contexts) relation on proof-nets generated by the equations in E. We shall write R/E for the system of *reduction modulo an equivalence relation [Ter03]* made of the rules in R and the equations in E, given by $r \rightarrow_{R/E} s$ if and only if there exist r' and s' such that $r \sim_E r' \rightarrow_R s' \sim_E s$.

In order to prove one of the main properties of $\lambda\mathsf{lxr}$, namely strong normalisation, we shall use the following result:

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

Proof. This result is proved in [Pol04] for which we refer the interested reader for full details. For the sake of completeness, we explain the main steps of the proof here.

We note PN the system containing only rules *Ax-cut*, $⅋$-$\otimes$, *w-b*, *d-b*, *c-b*, *b-b* and W the system made of rules `U` and `V` (so that R is $PN \cup W$). The notation PN/E (resp. W/E) is used to denote PN (resp. W) reduction modulo E.

1. The reduction relation PN/E is terminating.

   Proof. The proof can be found in [DCG99].

2. The reduction relation W/E is terminating.

   Proof. We assign to each proof net p a pair $\langle p_1, p_2 \rangle$, where p_1 is the number of nodes in the proof-net, and p_2 is the the sum of the depths of all the weakening nodes in the net. The reduction relation W/E strictly decreases $\langle p_1, p_2 \rangle$ w.r.t the lexicographic order : `U` decreases p_1, `V` decreases p_2 while not modifying p_1, and the congruence $\sim_E$ keep $\langle p_1, p_2 \rangle$.

3. The relation W/E can be postponed w.r.t PN/E:
   if $t \rightarrow_{W/E} \rightarrow_{PN/E} t'$, then $t \rightarrow^+_{PN/E} \rightarrow^*_{R/E} t'$.

   Proof. Uses postponment of `U` (resp. `V`) w.r.t PN see [DCKP03] (resp. [Pol04]) and the fact that W and $\sim_E$ commute.

4. The relation R/E is terminating.

   Proof. By points 3 and 1 and 2.

3.1 Interpreting terms into proof-nets

We now present the natural interpretation of typed $\lambda\mathsf{lxr}$-terms as proof-nets. For that, we use the standard translation of intuitionistic types [Gir87] given by :

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

Figure 8 defines the translation $T(\_)$ from derivable typing judgements of $\lambda\mathsf{lxr}$ 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 $T(t)$

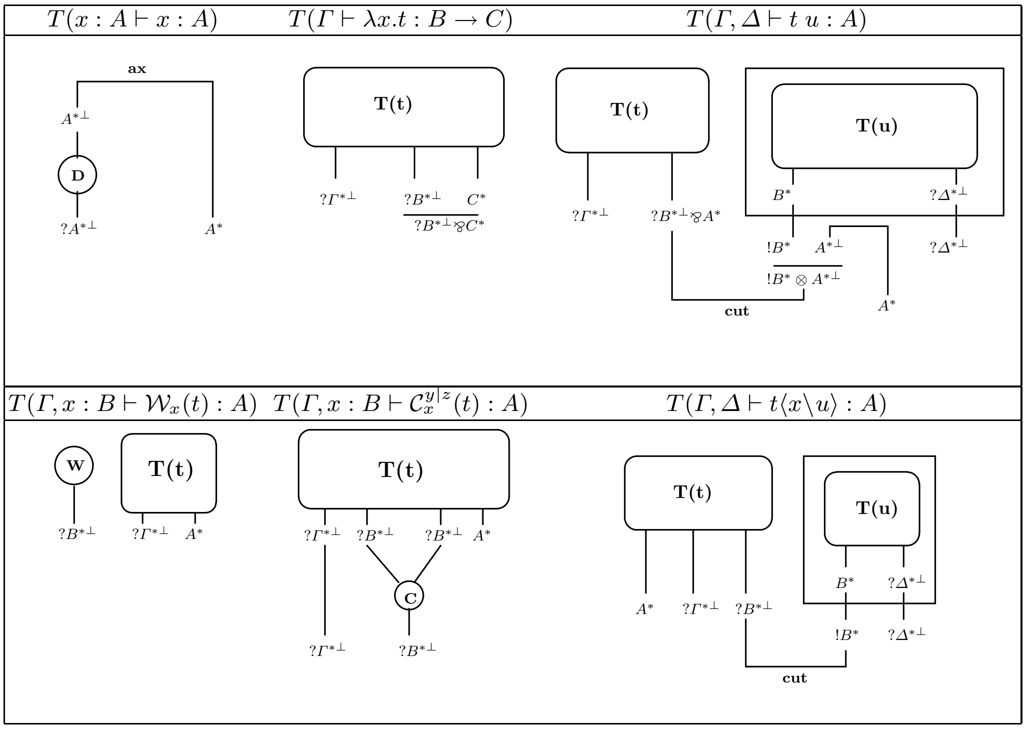


Figure 8. Encoding typed λlxr-terms into MELL proof-nets

instead of $T(\Gamma \vdash t : A)$ when Γ and A do not matter or are clear (for instance from Subject Reduction in λlxr, Theorem 2). The translation $T(\_)$ satisfies the following properties:

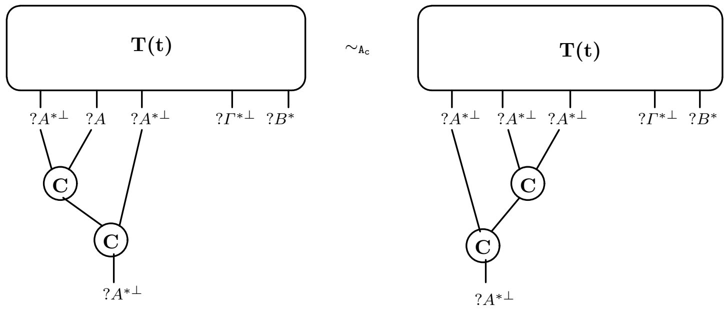
We now show how to simulate λlxr-reduction into R/E-reduction. This proof justifies the use of the additional equations $\mathtt{A_c}$ and $\mathtt{P_{cb}}$ as well as the additional reduction rules V and U. Indeed, we use the equation $\mathtt{A_c}$ on proof-nets to simulate the $\mathtt{A_c}$ equation of λlxr-terms, the $\mathtt{P_{cb}}$ equation to simulate rules CApp2 and CSubs, the rule V to simulate WApp2 and WSubs, and the U rule to simulate Merge.

Lemma 6 (Simulation of λlxr-reduction). *Let s be a simply-typed λlxr-term.*

- *If $s \equiv s'$, then $T(s) \sim_E T(s')$.*
- *If $s \to_B s'$, then $T(s) \to^2_{R/E} T(s')$.*
- *If $s \to_{\mathsf{xr}} s'$, then $T(s) \to^*_{R/E} T(s')$.*

Proof. For the first property we consider the base cases below, the rest of the induction being straightforward using the fact that the relations $\equiv$ and $\sim_E$ are congruences.

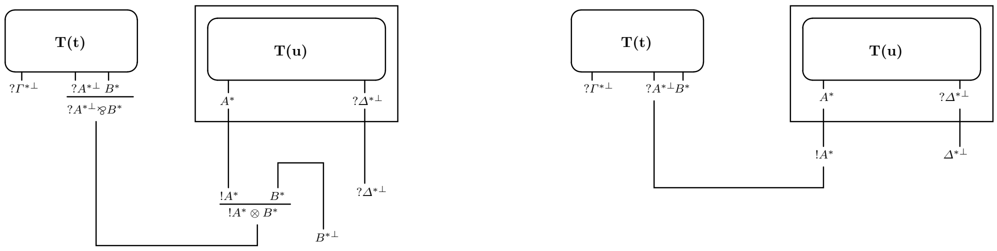
– For $\mathcal{C}^{x|v}_w(\mathcal{C}^{y|z}_x(t)) \equiv_{\mathsf{A_c}} \mathcal{C}^{x|y}_w(\mathcal{C}^{z|v}_x(t))$ we have the two following equivalent proof-nets.



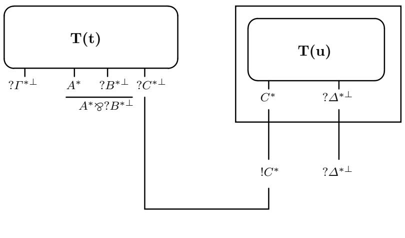
– For all the other cases we leave to the reader the (easy) verification that both interpretations are exactly equal.

For the second and third property, we proceed by induction on the reduction step. We only show here the cases of root reductions. We give for each case the rules/equations needed to verify the statement.

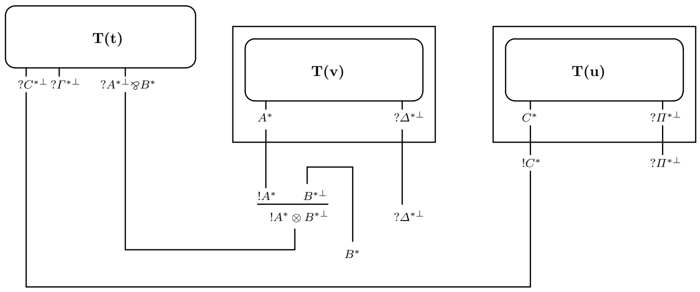
– For $t_1 = (\lambda x.t)\ u \to_B\ t\langle x\backslash u\rangle = t_2$, let $\Gamma := FV(\lambda x.t)$ and $\Delta := FV(u)$. We have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \to^2_{⅋\text{-}\otimes, Ax\text{-}cut}\ T(t_2)$ in exactly two steps.



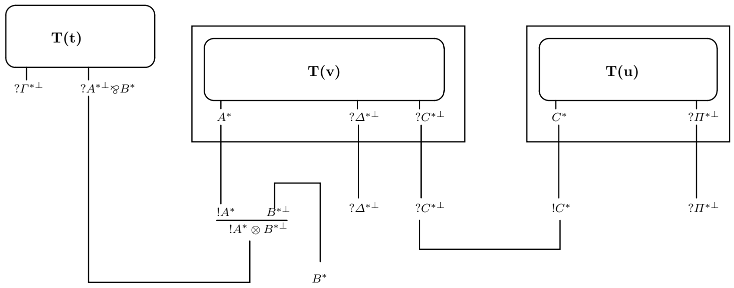
– For $t_1 = (\lambda y.t)\langle x\backslash u\rangle \to_{\mathsf{Abs}}\ \lambda y.t\langle x\backslash u\rangle = t_2$, let $\Gamma, x := FV(\lambda y.t)$ and $\Delta := FV(u)$. We have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

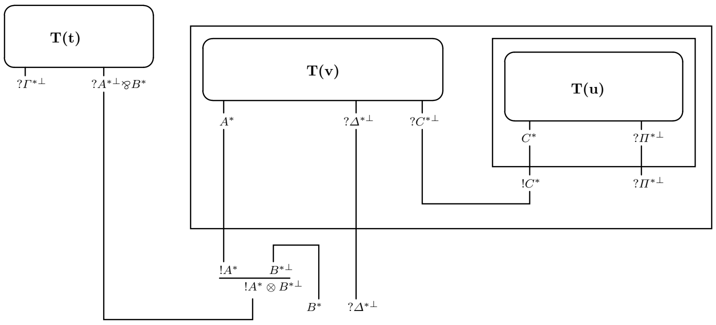


– For $t_1 = (t\ v)\langle x\backslash u\rangle \to_{\mathsf{App1}}\ (t\langle x\backslash u\rangle\ v) = t_2$ with $x \in FV(t)$, exactly the same interpretation $T(t_1)$ and $T(t_2)$.

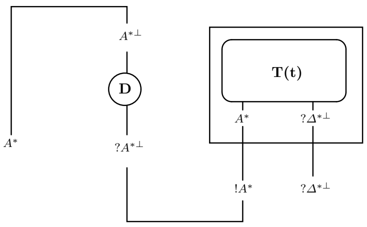


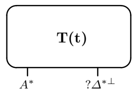
– For $t_1 = (t\ v)\langle x\backslash u\rangle \rightarrow_{\mathsf{App2}} (t\ v\langle x\backslash u\rangle) = t_2$ with $x \in FV(v)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow_{b\text{-}b} T(t_2)$.



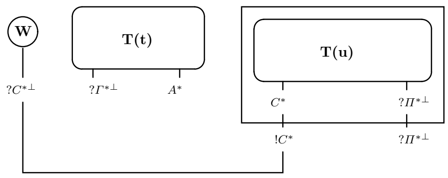


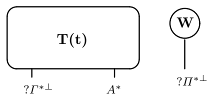
– For $t_1 = x\langle x\backslash u\rangle \rightarrow_{\mathsf{Var}} u = t_2$, let $\Delta := FV(u)$. We have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow^*_{d\text{-}b,Ax\text{-}cut} T(t_2)$.



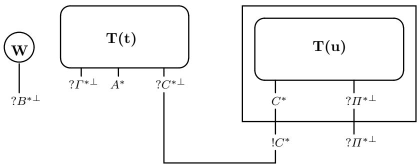


– For $t_1 = \mathcal{W}_x(t)\langle x\backslash u\rangle \rightarrow_{\mathsf{Weak1}} \mathcal{W}_\Pi(t) = t_2$, where $\Pi := FV(u)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow_{w\text{-}b} T(t_2)$.

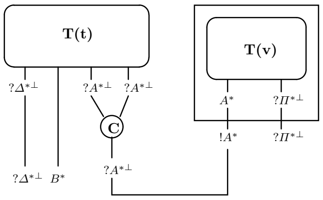


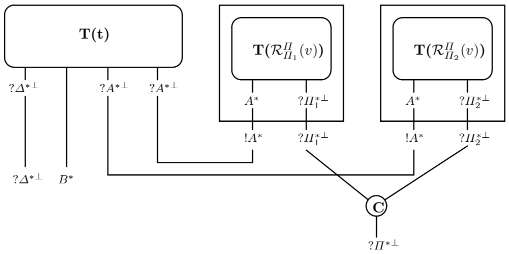


– For $t_1 = \mathcal{W}_y(t)\langle x\backslash u\rangle \rightarrow_{\mathsf{Weak2}} \mathcal{W}_y(t\langle x\backslash u\rangle) = t_2$, where $x \neq y$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.

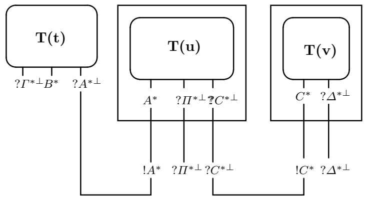


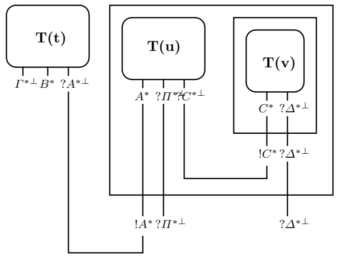
– For $t_1 = \mathcal{C}_x^{y|z}(t)\langle x\backslash v\rangle \rightarrow_{\mathsf{Cont}} \mathcal{C}_{\Pi}^{\Pi_1|\Pi_2}(t\langle y\backslash \mathcal{R}_{\Pi_1}^{\Pi}(v)\rangle\langle z\backslash \mathcal{R}_{\Pi_2}^{\Pi}(v)\rangle) = t_2$, where $\Pi := FV(v)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow_{\mathsf{c\text{-}b}} T(t_2)$.



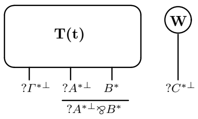


– For $t_1 = t\langle y\backslash u\rangle\langle x\backslash v\rangle \rightarrow_{\mathsf{Comp}} t\langle y\backslash u\langle x\backslash v\rangle\rangle = t_2$ where $x \in FV(u)$, let $(\Gamma, y) := FV(t)$, $(\Pi, x) := FV(u)$ and $\Delta := FV(v)$. We have the following interpretations $T(t_1)$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

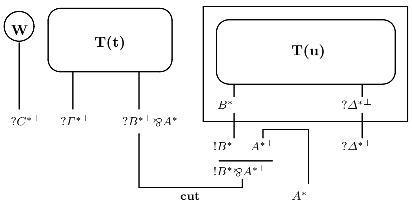




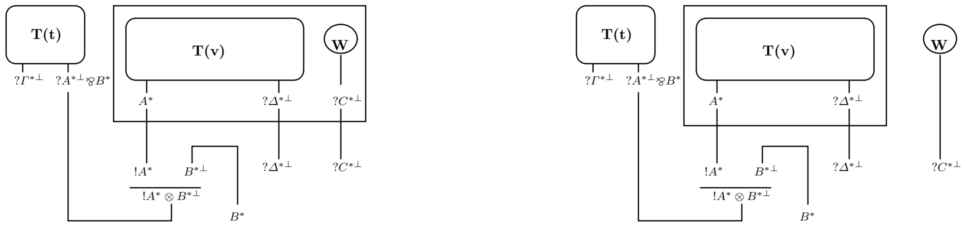
– For $t_1 = \lambda x.\mathcal{W}_y(t) \rightarrow_{\mathsf{WAbs}} \mathcal{W}_y(\lambda x.t) = t_2$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.



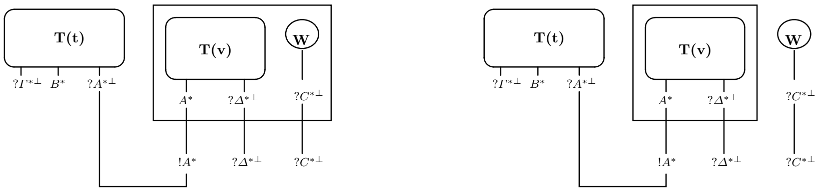
– For $t_1 = \mathcal{W}_y(t)\ u \rightarrow_{\mathsf{WApp1}} \mathcal{W}_y(tu) = t_2$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.



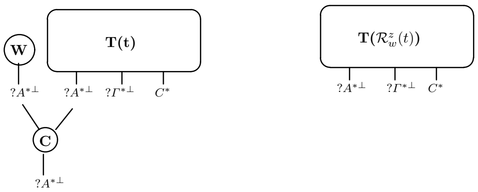
– For $t_1 = t\ \mathcal{W}_y(v) \rightarrow_{\mathsf{WApp2}} \mathcal{W}_y(tv) = t_2$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow_{\mathsf{v}} T(t_2)$.



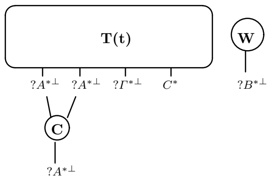
– For $t_1 = t\langle x \backslash \mathcal{W}_y(v)\rangle \rightarrow_{\mathsf{WSubs}} \mathcal{W}_y(t\langle x\backslash v\rangle) = t_2$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow_{\mathtt{V}} T(t_2)$.



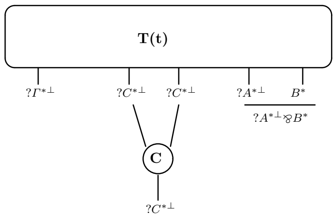
– For $t_1 = \mathcal{C}_w^{y|z}(\mathcal{W}_y(t)) \rightarrow_{\mathsf{Merge}} \mathcal{R}_w^z(t) = t_2$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \rightarrow_{\mathtt{U}} T(t_2)$.



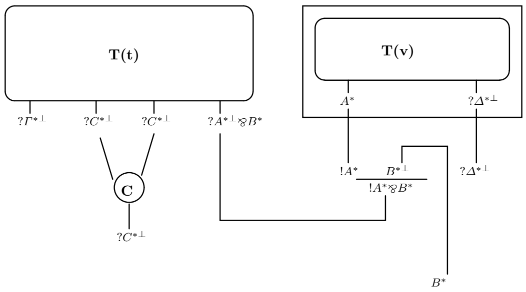
– For $t_1 = \mathcal{C}_w^{y|z}(\mathcal{W}_x(t)) \rightarrow_{\mathsf{Cross}} \mathcal{W}_x(\mathcal{C}_w^{y|z}(t)) = t_2$ when $x \neq y, \quad x \neq z$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.



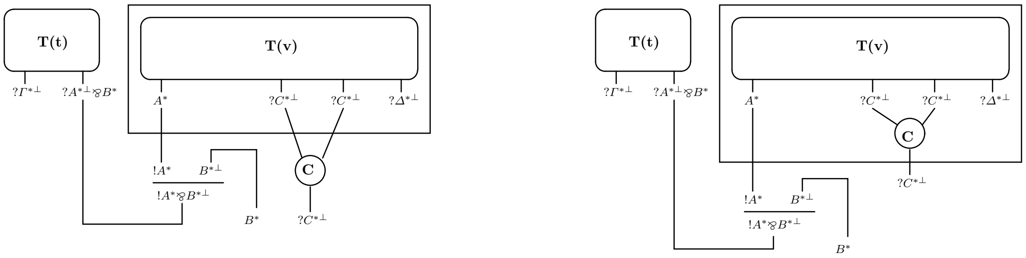
– For $t_1 = \mathcal{C}_w^{y|z}(\lambda x.t) \rightarrow_{\mathsf{CAbs}} \lambda x.\mathcal{C}_w^{y|z}(t) = t_2$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.



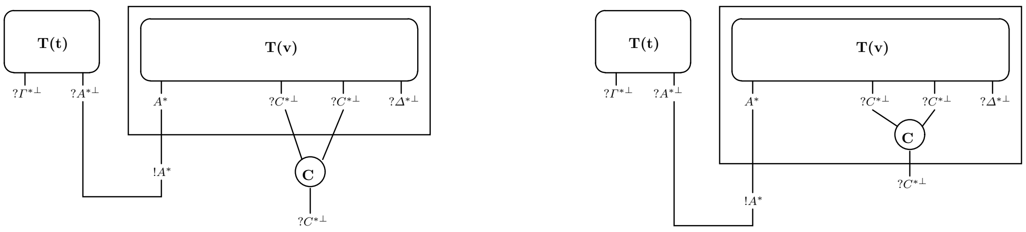
– For $t_1 = \mathcal{C}_w^{y|z}(t\ u) \rightarrow_{\mathsf{CApp1}} \mathcal{C}_w^{y|z}(t)\ u = t_2$ when $y, z \in FV(t)$, we have exactly the same interpretation $T(t_1)$ and $T(t_2)$.



– For $t_1 = \mathcal{C}^{y|z}_w(t\ u) \rightarrow_{\mathsf{CApp2}} t\ \mathcal{C}^{y|z}_w(u) = t_2$ when $y, z \in FV(u)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \sim_{\mathsf{P_{cb}}} T(t_2)$.



– For $t_1 = \mathcal{C}^{y|z}_w(t\langle x\backslash u\rangle) \rightarrow_{\mathsf{CSubs}} t\langle x\backslash \mathcal{C}^{y|z}_w(u)\rangle$ when $y, z \in FV(u)$, we have the following interpretations $T(t_1)$ and $T(t_2)$ and we can verify that $T(t_1) \sim_{\mathsf{P_{cb}}} T(t_2)$.



As a consequence we obtain one of the main important properties of $\lambda\mathsf{lxr}$:

Theorem 4 (Strong Normalisation).
The relation $\rightarrow_{\lambda\mathsf{lxr}}$ *is strongly normalising on simply-typed* $\lambda\mathsf{lxr}$*-terms.*

Proof. Suppose $\rightarrow_{\lambda\mathsf{lxr}}$ is not strongly normalising. Then, since xr terminates by Theorem 1, an infinite $\lambda\mathsf{lxr}$-reduction sequence would have infinitely many B-steps. But this would lead by Lemma 6 to an infinite R/E-reduction sequence which is impossible by Theorem 3.

The simulation result of $\lambda\mathsf{lxr}$-reduction that we use to conclude the strong normalisation of the simply-typed $\lambda\mathsf{lxr}$-calculus helps understanding its reduction rules and equations, but another technique [Pol04] not using proof-nets but using Preservation of Strong Normalisation (cf. Section 5) together with the strong normalisation of the simply-typed λ-calculus could also be used. Direct proofs of strong normalization using for instance reducibility by perpetuality [Bon01,LLD$^+$04] seem much more difficult to adapt to our case, owing to the fact that rewriting is performed modulo a congruence. However, we expect the study of perpetuality in $\lambda\mathsf{lxr}$ to be particularly interesting, particularly in connection with intersection types and characterisation of strongly normalisable terms. Also, we would expect that strong normalisation of proof-nets can be inferred from a direct proof of strong normalisation of $\lambda\mathsf{lxr}$, i.e. that the two properties have the same strength. We leave those topics for future work.

3.2 Terms having the same box structure

In the rest of this section we restrict our attention to the cut-elimination steps in proof-nets that modify their box structure; thus tackling for $\lambda\mathsf{lxr}$ a problem studied in [Lau03] for the $\lambda\mu$-calculus: characterising those terms that are translated (in our case by $T(\_)$) into proof-nets having identical box structures. Characterising those terms that are translated into the same proof-nets (modulo a weaker congruence such as $\mathtt{A_c}, \mathtt{P_{cb}}$) could also be interesting and is left as further work.

Let TB be the reduction relation on PN generated by the rules that do not modify the box structure, namely $\mathtt{V}$, $\mathtt{U}$, *Ax-cut* and $⅋$-$\otimes$, modulo the congruence $\sim_E$. Since termination holds for the whole system R/E which contains TB, then using known techniques for abstract reduction systems [Hue80] we obtain:

Proposition 1. *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 8. *Let r and r' be two proof-nets. Then the relation $r \approx r'$ is defined as $TB(r) \sim_E TB(r')$.*

In order to characterise the terms that are translated to proof-nets having the same box structure, we identify those rules of $\lambda\mathsf{lxr}$ that do not change the box structure:

Definition 9. *We define the congruence $\cong$ between $\lambda\mathsf{lxr}$-terms by adding to $\equiv$ the following rules turned into equalities:*

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

Note that WSubs and CSubs are captured by $\cong$, in the following sense:

Remark 4. If $t \longleftrightarrow^*_{\mathsf{WSubs} \cup \mathsf{CSubs}} t'$, then $t \cong t'$.

Remark also that the rules $\{\mathsf{App2}, \mathsf{Comp}, \mathsf{Var}, \mathsf{Weak1}, \mathsf{Cont}\}$, which are not in $\cong$, change the structure of boxes. More precisely, $\mathsf{App2}$ and Comp in $\lambda\mathsf{lxr}$ correspond to b-b in R, Var to d-b, $\mathsf{Weak1}$ to w-b and Cont to c-b.

Definition 10. *Given a derivable typing judgement $\Gamma \vdash t : A$, 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 $T(t_1) \approx T(t_2)$ if and only if $NT(t_1) \sim_E NT(t_2)$.

The following *soundness* property relates two $\cong$-convertible terms w.r.t their translations into proof-nets. The reverse result concerning *completeness* is shown later (Theorem 6).

Theorem 5 (Soundness). *Given two simply-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 root cases.

- For $t_1 =_B t_2$, we have shown that $T(t_1) \rightarrow^*_{\mathcal{B}\otimes, Ax\text{-}cut} T(t_2)$ (cf. the proof of Lemma 6) so that $NT(t_1) = NT(t_2)$ and thus $NT(t_1) \sim_E NT(t_2)$ trivially holds.
- For $t_1 =_{\mathsf{Abs},\mathsf{WAbs},\mathsf{CAbs},\mathsf{App1},\mathsf{WApp1},\mathsf{CApp1},\mathsf{CApp2},\mathsf{Weak2},\mathsf{Cross}} t_2$, we have shown that $T(t_1) \sim_E T(t_2)$ (cf. the proof of Lemma 6) so that $NT(t_1) \sim_E NT(t_2)$ also holds.
- For $t_1 =_{\mathsf{WApp2}} t_2$, we have shown that $T(t_1) \rightarrow_{\mathtt{V}} T(t_2)$ (cf. the proof of Lemma 6) so that $NT(t_1) = NT(t_2)$ and thus $NT(t_1) \sim_E NT(t_2)$ trivially holds.
- For $t_1 =_{\mathsf{Merge}} t_2$, we have shown that $T(t_1) \rightarrow_{\mathtt{U}} T(t_2)$ (cf. the proof of Lemma 6) 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 $T(t_1) \sim_E T(t_2)$ (cf. the proof of Lemma 6) 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 which is used further in Lemma 8 to reason according to their particular head-shapes.

Lemma 7 (Revealing the head-shape of a term). *For every term t, there is a term $t' \cong t$ with $|t'| \leq |t|$ such that either*

- $t' = \lambda x.t''$, *or*
- $t' = \mathcal{W}_x(t'')$, *or*
- $t' = \mathcal{C}_x^{y|z}(t'')$, *or*
- $t' = (xt_1 \dots t_n)\langle x_1 \backslash u_1 \rangle \dots \langle x_m \backslash u_m \rangle$ $(n \geq 0, m \geq 0)$, *or*

– $t' = \mathcal{W}_x(t'')\langle x\backslash u\rangle\langle x_1\backslash u_1\rangle \dots \langle x_m\backslash u_m\rangle$ *with* $m \geq 0$ *and* $x_i \notin FV(t'')$ *for all* i, *or*
– $t' = \mathcal{C}_x^{y|z}(t'')\langle x\backslash u\rangle\langle x_1\backslash u_1\rangle \dots \langle x_m\backslash u_m\rangle$ *with* $m \geq 0$ *and* $x_i \notin FV(t'')$ *for all* i.

Proof. By induction on $|t|$. $|t'| \leq |t|$ by induction hypothesis and because the way we use $\cong$ to convert the terms can only decrease their sizes. We abbreviate $\langle x_1\backslash u_1\rangle \dots \langle x_m\backslash u_m\rangle$ as $\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ in what follows.

In each of the following cases we obtain

– If t is a variable (taking $n = m = 0$ in the fourth case above), an abstraction, a weakening or a contraction, it is trivial.
– If $t = t'_1\langle x'\backslash t'_2\rangle$, then we apply the induction hypothesis on t'_1:
  - If $t'_1 \cong \lambda x.t''$ (x is necessarily different from x' by α-equivalence), then $t \cong (\lambda x.t'')\langle x'\backslash t'_2\rangle \cong_{\mathsf{Abs}} \lambda x.t''\langle x'\backslash t'_2\rangle$.
  - If $t'_1 \cong \mathcal{W}_y(t'')$, then for the case $y = x'$ we are done, otherwise we have $t \cong \mathcal{W}_y(t'')\langle x'\backslash t'_2\rangle \cong_{\mathsf{Weak2}} \mathcal{W}_y(t''\langle x'\backslash t'_2\rangle)$.
  - If $t'_1 \cong \mathcal{C}_x^{y|z}(t'')$, then for the case $x = x'$ we are done, otherwise we have $t \cong \mathcal{C}_x^{y|z}(t'')\langle x'\backslash t'_2\rangle \cong_{\mathsf{P_{cs}}} \mathcal{C}_x^{y|z}(t''\langle x'\backslash t'_2\rangle)$.
  - If $t'_1 \cong (xt_1 \dots t_n)\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$, then $t \cong (xt_1 \dots t_n)\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle\langle x'\backslash t'_2\rangle$.
  - If $t'_1 \cong \mathcal{W}_x(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ with $x_i \notin FV(t'')$ for all i, then we have $t \cong \mathcal{W}_x(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle\langle x'\backslash t'_2\rangle$. Now either $x' \notin FV(t'')$ and we are done, or $x' \in FV(t'')$ and $t \cong_{\mathsf{Weak2}} \mathcal{W}_x(t''\langle x'\backslash t'_2\rangle)\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ since x_i is not free in t'_2 by α-equivalence.
  - If $t'_1 \cong \mathcal{C}_x^{y|z}(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ with $x_i \notin FV(t'')$ for all i, then we have $t \cong \mathcal{C}_x^{y|z}(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle\langle x'\backslash t'_2\rangle$. Now either $x' \notin FV(t'')$ and we are done, or $x' \in FV(t'')$ and $t \cong_{\mathsf{P_{cs}}} \mathcal{C}_x^{y|z}(t''\langle x'\backslash t'_2\rangle)\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ since x_i is not free in t'_2 by α-equivalence.
– If $t = t'_1\ t'_2$, then we apply the induction hypothesis on t'_1.
  - If $t'_1 \cong \lambda x.t''$, then $t \cong_B t''\langle x\backslash t'_2\rangle$, on which we apply the induction hypothesis because the size is strictly smaller.
  - If $t'_1 \cong \mathcal{W}_y(t'')$, then $t \cong \mathcal{W}_y(t'')\ t'_2 \cong_{\mathsf{WApp1}} \mathcal{W}_y(t''\ t'_2)$.
  - If $t'_1 \cong \mathcal{C}_x^{y|z}(t'')$, then $t \cong \mathcal{C}_x^{y|z}(t'')\ t'_2 \cong_{\mathsf{CApp1}} \mathcal{C}_x^{y|z}(t''\ t'_2)$.
  - If $t'_1 \cong (xt_1 \dots t_n)\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$, then we have $t \cong (xt_1 \dots t_n)\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle\ t'_2 \cong_{\mathsf{App1}} (xt_1 \dots t_n t'_2)\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$.
  - If $t'_1 \cong \mathcal{W}_x(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ with $x_i \notin FV(t'')$ for all i, then we have $t \cong \mathcal{W}_x(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle\ t'_2 \cong_{\mathsf{App1,WApp1}} \mathcal{W}_x(t''\ t'_2)\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$.
  - If $t'_1 \cong \mathcal{C}_x^{y|z}(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ with $x_i \notin FV(t'')$ for all i, then we have $t \cong \mathcal{C}_x^{y|z}(t'')\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle\ t'_2 \cong_{\mathsf{App1,CApp1}} \mathcal{C}_x^{y|z}(t''\ t'_2)\langle x\backslash u\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$.

3.3 Towards completeness

We say that a node n is *final* in a proof-net if 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 if some c port of n is "free". Final nodes can be seen as the *interface* of a proof-net. Remark that **cut** is never final. We will be particularly interested in final nodes $\mathbf{W}/\mathbf{C}/\invamp$ which have a direct syntactical interpretation in our term calculus. A type is said to be *distinguished* in a proof-net if it is not a formula of the form $?A$ and it is the type of an edge connected to a c port of a final node. Every edge decorated with a formula of the form $?A$ 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 $\mathbf{W}/\mathbf{C}/\invamp$-nodes).

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

A node $\mathbf{W}$ or $\mathbf{C}$ is *final-modulo* in a proof-net r if it is final in some proof-net r' which is $\{\mathtt{V}, \mathtt{P_{cb}}\}$-equivalent to r. Said differently, $\mathbf{W}$ or $\mathbf{C}$ is *final-modulo* in a proof-net seen as a directed graph if there is a path -only with box nodes but possibly of length 0- from this node to some final node. Finally, a node different from $\mathbf{W}$ and $\mathbf{C}$ is *final-modulo* if it is final. Note that the notion of final-modulo is invariant under conversion by the congruence E, so final-modulo nodes can be seen as the *interface* of a proof-net modulo. A node n occurs at *depth* k in a proof-net r if n appears inside (or is a predecessor in the oriented graph) k different boxes of r.

Remark 5. Let t be a $\lambda\mathsf{lxr}$-term. Then the proof-net $T(\Gamma \vdash t : A)$ has a unique distinguished type which is A^*. As a consequence, $\invamp$-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.

The following lemma establishes, for any typed tem t, a connection between the interface of $NT(t)$ and the constructors of t that can be pulled to the top-level by the congruence $\cong$. It will be useful to reason by induction on terms modulo $\cong$: we shall then be able to assume that they have a particular shape insead of having to cover all cases.

Lemma 8. *Let t be a $\lambda\mathsf{lxr}$-term and let f be a final-modulo node of $NT(t)$.*

1. *If f is a $\mathbf{W}$-node whose c edge is labelled with x, then there exist a term t' such that $t \cong \mathcal{W}_x(t')$ and $|t| = |\mathcal{W}_x(t')|$.*
2. *If f is a $\mathbf{C}$-node whose c edge is labelled with x, then there exist a term t' such that $t \cong \mathcal{C}_x^{y|z}(t')$ and $|t| = |\mathcal{C}_x^{y|z}(t')|$.*
3. *If f is an $\invamp$-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 $|t|$. See the appendix for details.

The following lemma states how applications are recognisable as cuts.

Lemma 9. *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 $t = \mathcal{W}_y(u)$, then $NT(t)$ has a final-modulo **W**-node.
- If $t = \mathcal{C}_x^{y|z}(u)$, then $NT(t)$ has a **C** final-modulo node.
- If t is an application or a closure, then $t \cong_B u_0 u_1 \dots u_n$ ($n \geq 1$), where u_0 is neither an application nor a closure since for every closure $v\langle y \backslash u\rangle$ in t we have $v\langle y \backslash u\rangle \cong_B (\lambda y.v)u$, having the same size.
  - If $u_0 = y$, then there is no **cut**-node at depth 0 in $NT(t)$.
  - If $u_0 = \mathcal{W}_y(w)$, then $NT(t)$ has a final-modulo **W**-node.
  - If $u_0 = \mathcal{C}_w^{y|z}(t')$, then $NT(t)$ has a final-modulo **C**-node.
  - If $u_0 = (\lambda y.w)$, then we have

$$\begin{aligned} t &\cong_B (\lambda y.w)u_1 u_2 \dots u_n \\ &\cong_B w\langle y \backslash u_1\rangle u_2 \dots u_n \\ &\cong_{\mathsf{App1}} (w u_2 \dots u_n)\langle y \backslash u_1\rangle \\ &\cong_B (\lambda y.w u_2 \dots u_n)u_1 \end{aligned}$$

We can now state one of the main results of this paper, showing that the simply-typed λlxr-calculus captures (intuitionistic) proof-nets using a term syntax.

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. We reason by induction on $NT(t_1)$. In what follows, for any proof-net r and final-modulo node n, we note $r \setminus n$ the proof-net obtained from r by erasing n and its corresponding edges.

- If $NT(t_1)$ has a final-modulo **W**-node called n, then $t_1 \cong \mathcal{W}_x(t'_1)$ and $t_2 \cong \mathcal{W}_x(t'_2)$ by Lemma 8. By definition of the translation $NT(t'_1) = NT(t_1) \setminus n$ and $NT(t'_2) = NT(t_2) \setminus n$. Thus, it is easy to see that $NT(t'_1) \sim_E NT(t'_2)$. By the i.h. we have $t'_1 \cong t'_2$ so that $\mathcal{W}_x(t'_1) \cong \mathcal{W}_x(t'_2)$.
- If $NT(t_1)$ has a final-modulo **C**-node called n, then $t_1 \cong \mathcal{C}_{x_1}^{y_1|z_1}(t'_1)$ and $t_2 \cong \mathcal{C}_{x_2}^{y_2|z_2}(t'_2)$ by Lemma 8. 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 the i.h. we have $t'_1 \cong t'_2$ so that $\mathcal{C}_{x_1}^{y_1|z_1}(t'_1) \cong \mathcal{C}_{x_2}^{y_2|z_2}(t'_2)$.
- If $NT(t_1)$ has a final-modulo $\parr$-node called n, then $t_1 \cong \lambda x.t'_1$ and $t_2 \cong \lambda x.t'_2$ by Lemma 8. 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 the 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 $\mathbf{W}/\mathbf{C}/⅋$-node, then t_1 is an application or a closure.
If $NT(t_1)$ has no **cut** at depth 0, then $t_1 \cong xu_1 \ldots u_n$ and $t_2 \cong xv_1 \ldots v_n$. Since $NT(u_i) \sim_E NT(v_i)$, then by the i.h. we have $u_i \cong v_i$ and thus $xu_1 \ldots u_n \cong xv_1 \ldots v_n$.
If $NT(t_1)$ has some **cut** at depth 0, then by Lemma 9 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 the i.h. The same happens with $NT(v_1)$ and $NT(v_2)$ so that $v_1 \cong v_2$. We can then conclude $(\lambda x.u_1)v_1 \cong (\lambda x.u_2)v_2$.

This result allows us to interpret not only λlxr-terms as proof-nets (Soundness Theorem) but also, the other way around, proof-nets in some particular form to λlxr-terms. 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 λlxr only because the operators of our calculus adequately reflect the structure of proof-nets.

4 Recovering the λ-calculus

We show in this section the relation between λlxr-terms and λ-terms. We refer the reader to [Bar84] for a presentation of λ-calculus and all its standard notions such as free variables, α-equivalence, Barendregt's convention, etc, that we use in this section.

More precisely, we show that the linearity constraints and the use of explicit resource operators in λlxr are sufficient to decompose the β-reduction step into smaller steps. We will also show in this section the relation between the simply-typed λlxr and the simply-typed λ-calculus. For that we first recall in Figure 9 the typing rules for λ-calculus.

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

Figure 9. Typing Rules for λ-calculus

We consider λ-terms as an independent syntax rather than particular λlxr-terms, since they might not be linear. We shall 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 $\boldsymbol{\lambda}$-calculus to $\boldsymbol{\lambda}$lxr-calculus

We now describe how to encode a λ-term into a (linear) λlxr one.

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

$$\begin{array}{lll}
\mathcal{A}(x) & := x & \\
\mathcal{A}(\lambda x.t) & := \lambda x.\mathcal{A}(t) & \text{if } x \in \mathcal{FV}(t) \\
\mathcal{A}(\lambda x.t) & := \lambda x.\mathcal{W}_x(\mathcal{A}(t)) & \text{if } x \notin \mathcal{FV}(t) \\
\mathcal{A}(t\ u) & := \mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(t))\ \mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(u)))\ \text{where} & \begin{cases} \Upsilon, \Omega \text{ are fresh} \\ \Phi := \mathcal{FV}(t) \cap \mathcal{FV}(u) \end{cases}
\end{array}$$

Using the fact that $\mathcal{W}_{\emptyset}(t) = t$, we can write the translation of an abstraction, with only one case, as $\mathcal{A}(\lambda x.t) = \lambda x.\mathcal{W}_{\{x\}\setminus\mathcal{FV}(t)}(\mathcal{A}(t))$. Note that $\mathcal{A}(t\ u) = \mathcal{A}(t)\mathcal{A}(u)$ in the particular case $\mathcal{FV}(t) \cap \mathcal{FV}(u) = \emptyset$. More generally, a λ-term which, viewed as a particular λlxr-term, is linear, is translated by $\mathcal{A}$ to itself. Note also that the weakenings and contractions introduced by this translations are already in their canonical places, i.e. $\mathcal{A}(t)$ is an xr-normal form for every λ-term t.

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

Lemma 10 (Properties of $\mathcal{A}$).

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

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 $\mathcal{A}(t) = \lambda x.\mathcal{W}_x(\lambda y.(y\ \mathcal{C}_z^{z_1|z_2}(z_1\ z_2)))$.

We now want to simulate a β-reduction step in λlxr, so we start by proving that the interaction between (and the propagation of) the three operators of λlxr by means of the system xr do implement the notion of substitution. More precisely, given two λ-terms t_1 and t_2, we identify a λlxr-term, built from the translations by $\mathcal{A}$ of t_1 and t_2 and using a linear substitution operator, that reduces to the translation of $t_1\{x\backslash t_2\}$, as shown by the following lemma:

Lemma 11. *For all λ-terms t_1 and t_2 such that $x \in \mathcal{FV}(t_1)$,*

$$\mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(t_1))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle) \to_{\mathsf{xr}}^{*} \mathcal{A}(t_1\{x\backslash t_2\})$$

where $\Phi := (\mathcal{FV}(t_1) \setminus \{x\}) \cap \mathcal{FV}(t_2)$, provided that the former term is linear.

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

$$\mathcal{A}(t_1)\langle x\backslash\mathcal{A}(t_2)\rangle \to_{\mathsf{xr}}^{*} \mathcal{A}(t_1\{x\backslash t_2\})$$

Proof. By induction on the size of t_1, by propagating the linear substitution operator, pulling out weakenings and pushing in contractions. See the appendix for details.

The *correctness* result concerning linear substitution operators 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

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

As a consequence, the simulation property has to be stated by taking into account the operational behaviour of system xr given by Lemma 11.

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

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

1. The root case is the reduction $(\lambda x.t_1)t_2 \longrightarrow_\beta\ t_1\{x\backslash t_2\}$. By α-equivalence, $x \notin \mathcal{FV}(t_2)$.
   (a) If $x \notin \mathcal{FV}(t_1)$, let $\Phi := \mathcal{FV}(t_1) \cap \mathcal{FV}(t_2)$ and $\Xi := \mathcal{FV}(t_2) \setminus \mathcal{FV}(t_1)$.

$$\begin{array}{rll}
\mathcal{A}((\lambda x.t_1)t_2) = & & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\lambda x.\mathcal{W}_x(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1)))\ \mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))) \\
& \longrightarrow_B & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{W}_x(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1)))\langle x\backslash \mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))\rangle) \\
& \longrightarrow_{\mathsf{Weak1}} & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{W}_{\mathcal{FV}(\mathcal{R}^{\Phi}_{\Omega}(t_2))}(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1)))) \\
& \equiv & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{W}_{\Omega,\Xi}(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1)))) \\
& = & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{W}_{\Omega}(\mathcal{W}_{\Xi}(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1))))) \\
& \longrightarrow^*_{\mathsf{Merge}} & \mathcal{R}^{\Upsilon}_{\Phi}(\mathcal{W}_{\Xi}(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1)))) \\
& = & \mathcal{W}_{\Xi}(\mathcal{A}(t_1))
\end{array}$$

   Now it suffices to notice that $\Xi := \mathcal{FV}((\lambda x.t_1)\ t_2) \setminus \mathcal{FV}(t_1\{x\backslash t_2\})$ using $\mathcal{FV}(t_1\{x\backslash t_2\}) = \mathcal{FV}(t_1)$ since $x \notin \mathcal{FV}(t_1)$.
   (b) If $x \in \mathcal{FV}(t_1)$, let $\Phi := (\mathcal{FV}(t_1) \setminus \{x\}) \cap \mathcal{FV}(t_2)$.

$$\begin{array}{rll}
\mathcal{A}((\lambda x.t_1)t_2) = & & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\lambda x.\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t_1))\ \mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))) \\
& \longrightarrow_B & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{A}(\mathcal{R}^{\Phi}_{\Upsilon}(t_1))\langle x\backslash \mathcal{A}(\mathcal{R}^{\Phi}_{\Omega}(t_2))\rangle) \\
& \longrightarrow^*_{\mathsf{xr}} & \mathcal{A}(t_1\{x\backslash t_2\}) \qquad \text{by Lemma 11}
\end{array}$$

2. Now suppose $\lambda x.u \longrightarrow_\beta\ \lambda x.u'$ with $u \longrightarrow_\beta\ u'$,
   (a) If $x \notin \mathcal{FV}(u)$

$$\begin{array}{rll}
\mathcal{A}(\lambda x.u) = & & \lambda x.\mathcal{W}_x(\mathcal{A}(u)) \\
& \longrightarrow^+_{\lambda\mathsf{lxr}} & \lambda x.\mathcal{W}_x(\mathcal{W}_{\mathcal{FV}(u)\setminus\mathcal{FV}(u')}(\mathcal{A}(u'))) \qquad \text{by the i.h.} \\
& = & \lambda x.\mathcal{W}_x(\mathcal{W}_{\mathcal{FV}(\lambda x.u)\setminus\mathcal{FV}(\lambda x.u')}(\mathcal{A}(u'))) \\
& \longrightarrow^*_{\mathsf{WAbs}} & \mathcal{W}_{\mathcal{FV}(\lambda x.u)\setminus\mathcal{FV}(\lambda x.u')}(\lambda x.\mathcal{W}_x(\mathcal{A}(u')))
\end{array}$$

(b) If $x \in \mathcal{FV}(u)$

$$\begin{array}{rll} \mathcal{A}(\lambda x.u) = & \lambda x.\mathcal{A}(u) & \\ \longrightarrow^+_{\lambda\mathsf{lxr}} & \lambda x.\mathcal{W}_{\mathcal{FV}(u)\setminus\mathcal{FV}(u')}(\mathcal{A}(u')) & \text{by the i.h.} \\ = & \lambda x.\mathcal{W}_{\mathcal{FV}(\lambda x.u)\setminus\mathcal{FV}(u')}(\mathcal{W}_{\{x\}\setminus\mathcal{FV}(u')}(\mathcal{A}(u'))) & \\ = & \lambda x.\mathcal{W}_{\mathcal{FV}(\lambda x.u)\setminus\mathcal{FV}(\lambda x.u')}(\mathcal{W}_{\{x\}\setminus\mathcal{FV}(u')}(\mathcal{A}(u'))) & \\ \longrightarrow^*_{\mathsf{WAbs}} & \mathcal{W}_{\mathcal{FV}(\lambda x.u)\setminus\mathcal{FV}(\lambda x.u')}(\lambda x.\mathcal{W}_{\{x\}\setminus\mathcal{FV}(u')}(\mathcal{A}(u'))) & \end{array}$$

3. Now suppose $t_1\ t_2 \longrightarrow_\beta\ t'_1\ t_2$ with $t_1 \longrightarrow_\beta\ t'_1$, let
$\Sigma := \mathcal{FV}(t'_1) \cap \mathcal{FV}(t_2)$
$\Lambda := \mathcal{FV}(t'_1) \setminus (\mathcal{FV}(t'_1) \cap \mathcal{FV}(t_2))$
$\Psi := (\mathcal{FV}(t_1) \cap \mathcal{FV}(t_2)) \setminus \mathcal{FV}(t'_1)$
$\Xi := \mathcal{FV}(t_1) \setminus (\mathcal{FV}(t'_1) \setminus \mathcal{FV}(t_2))$
Note in particular that $\mathcal{FV}(t_1) \cap \mathcal{FV}(t_2)$ is a permutation of Σ, Ψ. Correspondingly, let Σ_l, Ψ_l and Σ_r, Ψ_r be fresh variables.
We have:

$$\begin{array}{rl} \mathcal{A}(t_1\ t_2) \equiv & \mathcal{C}^{\Sigma_l,\Psi_l|\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{R}^{\Sigma,\Psi}_{\Sigma_l,\Psi_l}(\mathcal{A}(t_1))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))) \\ \longrightarrow^+_{\lambda\mathsf{lxr}} & \mathcal{C}^{\Sigma_l,\Psi_l|\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{R}^{\Sigma,\Psi}_{\Sigma_l,\Psi_l}(\mathcal{W}_{\mathcal{FV}(t_1)\setminus\mathcal{FV}(t'_1)}(\mathcal{A}(t'_1)))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))) \\ & \text{by the i.h.} \\ \equiv & \mathcal{C}^{\Sigma_l,\Psi_l|\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{R}^{\Sigma,\Psi}_{\Sigma_l,\Psi_l}(\mathcal{W}_{\Xi,\Psi}(\mathcal{A}(t'_1)))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))) \\ = & \mathcal{C}^{\Sigma_l,\Psi_l|\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{W}_{\Xi}(\mathcal{W}_{\Psi_l}(\mathcal{R}^{\Sigma}_{\Sigma_l}(\mathcal{A}(t'_1))))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))) \\ \longrightarrow^*_{\mathsf{WApp1}} & \mathcal{C}^{\Sigma_l,\Psi_l|\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{W}_{\Xi}(\mathcal{W}_{\Psi_l}(\mathcal{R}^{\Sigma}_{\Sigma_l}(\mathcal{A}(t'_1))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))))) \\ \longrightarrow^*_{\mathsf{Cross}} & \mathcal{W}_{\Xi}(\mathcal{C}^{\Sigma_l,\Psi_l|\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{W}_{\Psi_l}(\mathcal{R}^{\Sigma}_{\Sigma_l}(\mathcal{A}(t'_1))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))))) \\ \longrightarrow^*_{\mathsf{Merge}} & \mathcal{W}_{\Xi}(\mathcal{C}^{\Sigma_l|\Sigma_r}_{\Sigma}(\mathcal{R}^{\Psi_r}_{\Psi}(\mathcal{R}^{\Sigma}_{\Sigma_l}(\mathcal{A}(t'_1))\ \mathcal{R}^{\Sigma,\Psi}_{\Sigma_r,\Psi_r}(\mathcal{A}(t_2))))) \\ = & \mathcal{W}_{\Xi}(\mathcal{C}^{\Sigma_l|\Sigma_r}_{\Sigma}(\mathcal{R}^{\Sigma}_{\Sigma_l}(\mathcal{A}(t'_1))\ \mathcal{R}^{\Sigma}_{\Sigma_r}(\mathcal{A}(t_2)))) \end{array}$$

Then it suffices to notice that $\Xi = \mathcal{FV}(t_1\ t_2) \setminus \mathcal{FV}(t'_1\ t_2)$.

4. The case $t_1\ t_2 \longrightarrow_\beta\ t_1\ t'_2$ is similar to the previous one.

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

Lemma 12 (Encoding $\mathcal{A}$ preserves types). *If t is a λ-term s.t. $\Gamma \vdash_\lambda t : A$, then $\Gamma \vdash \mathcal{W}_{\Gamma\setminus\mathcal{FV}(t)}(\mathcal{A}(t)) : A$.*

4.2 From $\boldsymbol{\lambda}$lxr-calculus to $\boldsymbol{\lambda}$-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:*

$$\begin{array}{ll}
\mathcal{B}(x) & := x \\
\mathcal{B}(\lambda x.t) & := \lambda x.\mathcal{B}(t) \\
\mathcal{B}(\mathcal{W}_x(t)) & := \mathcal{B}(t) \\
\mathcal{B}(\mathcal{C}_x^{y|z}(t)) & := \mathcal{B}(t)\{y\backslash x\}\{z\backslash x\} \\
\mathcal{B}(t\ u) & := \mathcal{B}(t)\ \mathcal{B}(u) \\
\mathcal{B}(t\langle x\backslash u\rangle) & := \mathcal{B}(t)\{x\backslash \mathcal{B}(u)\}
\end{array}$$

Remark that $\mathcal{B}(t)$ is *not* the xr-normal form of t since weakenings and contractions disappear and thus the linearity constraints need not hold anymore.

Lemma 13 (Properties of $\mathcal{B}$). *The translation $\mathcal{B}$ enjoys the following properties.*

- $\mathcal{B}(\mathcal{R}^{\Phi}_{\Upsilon}(t)) = \mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{B}(t))$
- $\mathcal{FV}(\mathcal{B}(t)) \subseteq \mathcal{FV}(t)$

The following result will allow us to project the λlxr-calculus onto the λ-calculus, as usually done for calculi with explicit substitutions [Ros96].

Lemma 14 (Simulating λlxr-reduction).

1. *If $t_1 \equiv t_2$, then $\mathcal{B}(t_1) = \mathcal{B}(t_2)$.*
2. *If $t_1 \rightarrow_B t_2$, then $\mathcal{B}(t_1) \rightarrow^*_\beta \mathcal{B}(t_2)$.*
3. *If $t_1 \rightarrow_{\mathsf{xr}} t_2$, then $\mathcal{B}(t_1) = \mathcal{B}(t_2)$.*

Proof. 1. This is obvious for the equivalence rule $\mathsf{P_w}$. 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 t_1 corresponds exactly to a β-reduction step at the root of $\mathcal{B}(t_1)$. For the closure under contexts, all cases are trivial except for:
   - the contraction, for which we use the fact that if $\mathcal{B}(t) \rightarrow^*_\beta \mathcal{B}(t')$ then $\mathcal{B}(t)\{y\backslash x\}\{z\backslash x\} \rightarrow^*_\beta \mathcal{B}(t')\{y\backslash x\}\{z\backslash x\}$.
   - the linear substitution operator, for which we use the two following facts: If $\mathcal{B}(t) \rightarrow^*_\beta \mathcal{B}(t')$ then $\mathcal{B}(t)\{x\backslash \mathcal{B}(u)\} \rightarrow^*_\beta \mathcal{B}(t')\{x\backslash \mathcal{B}(u)\}$. If $\mathcal{B}(u) \rightarrow^*_\beta \mathcal{B}(u')$ then $\mathcal{B}(t)\{x\backslash \mathcal{B}(u)\} \rightarrow^*_\beta \mathcal{B}(t)\{x\backslash \mathcal{B}(u')\}$.
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 $\mathsf{Abs}, \mathsf{App1}, \mathsf{App2}, \mathsf{Var}, \mathsf{CAbs}, \mathsf{CApp1}, \mathsf{CApp2}$, then the property follows from the definition of substitution.
- If the rule applied is Comp, then x is not free in t since the left-hand side is linear, so by Lemma 13 x is neither free in $\mathcal{B}(t)$. It suffices to use the substitution lemma as before.
- If the rule applied is $\mathsf{Weak1}$, then x is not free in t since the left-hand side is linear, so by Lemma 13 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 $\mathcal{B}(t)\{z\backslash w\} = \mathcal{B}(\mathcal{R}^z_w(t))$ by Lemma 13.
- If the rule applies is CSubs, then it is sufficient to apply the substitution lemma of λ-calculus.
- If the rule applied is Cont, then, as before, x is not free in $\mathcal{B}(t)$ so that $\mathcal{B}(t_1) = \mathcal{B}(t)\{y\backslash\mathcal{B}(u)\}\{z\backslash\mathcal{B}(u)\}$ by the substitution lemma. For the right-hand side we have

$$\mathcal{B}(t_2) = \mathcal{B}(t)\{y\backslash\mathcal{B}(\mathcal{R}^{\Phi}_{\Psi}(u))\}\{z\backslash\mathcal{B}(\mathcal{R}^{\Phi}_{\Xi}(u))\}\{\Psi\backslash\Phi\}\{\Xi\backslash\Phi\}$$

which, using Lemma 13, is equal to

$$\mathcal{B}(t)\{y\backslash\mathcal{R}^{\Phi}_{\Psi}(\mathcal{B}(u))\}\{z\backslash\mathcal{R}^{\Phi}_{\Xi}(\mathcal{B}(u))\}\{\Psi\backslash\Phi\}\{\Xi\backslash\Phi\}$$

which is

$$\mathcal{R}^{\Xi}_{\Phi}(\mathcal{R}^{\Psi}_{\Phi}(\mathcal{B}(t)\{y\backslash\mathcal{R}^{\Phi}_{\Psi}(\mathcal{B}(u))\}\{z\backslash\mathcal{R}^{\Phi}_{\Xi}(\mathcal{B}(u))\}))$$

which is equal to the left-hand side.

Corollary 1. *If $t_1 \longrightarrow_{\lambda\mathsf{lxr}} t_2$, then $\mathcal{B}(t_1) \longrightarrow^*_{\beta} \mathcal{B}(t_2)$.*

A straightforward induction on typing derivations allows us to show:

Lemma 15 ($\mathcal{B}$ preserves types). *If t is a $\lambda\mathsf{lxr}$-term such that $\Gamma \vdash t : A$, then $\Gamma \vdash_{\lambda} \mathcal{B}(t) : A$.*

We end this section by considering the *composition of the encodings* $\mathcal{B}$ and $\mathcal{A}$. Indeed, 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: $t = \mathcal{B}(\mathcal{A}(t))$ (straightforward induction on t). For $\mathcal{A}(\mathcal{B}(\_))$ we get $t \longrightarrow_{\mathsf{xr}} \mathcal{W}_{\mathcal{FV}(t)\backslash\mathcal{FV}(\mathcal{B}(t))}(t)$, which is in xr-normal form (we leave the proof to Section 5, Lemma 18).

5 Operational Properties

In Sections 2, 3 and 4 we have already established the properties of subject reduction, strong normalisation of simply-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 substitutions 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 $\lambda\mathsf{lxr}$ by reductions in the λI-calculus of [Klo80,Ned73], based on earlier work by [Chu41], 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,Len06], 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,Len06] 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 $\lambda\mathsf{lxr}$-terms and λI-terms (Definition 14).
2. Show that $t\ \mathcal{I}\ T$ and $t \to_{\mathsf{xr}} t'$ imply $t'\ \mathcal{I}\ T$, as well as $t\ \mathcal{I}\ T$ and $t \to_B t'$ imply there exists T' such that $t'\ \mathcal{I}\ T'$ and $T \to^+_{\beta\pi} T'$ (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 $u \in SN_\beta$ then $i(u) \in SN_{\beta\pi}$ (Theorems 9 and 10).
5. Show that $\mathcal{A}(u)\ \mathcal{I}\ i(u)$ (Theorem 11), where $\mathcal{A}(u)$ is the encoding given in Section 4, and conclude PSN (Corollary 3).

This proof can be captured by the following picture:

$$
\begin{array}{ccccccccc}
\mathcal{A}(u) & \to^*_{\mathsf{xr}} & t_1 & \to_B & t_2 & \to^*_{\mathsf{xr}} & t_3 & \to_B & t_4 \ \dots \\
\mathcal{I} & & \mathcal{I} & & \mathcal{I} & & \mathcal{I} & & \mathcal{I} \ \dots \\
i(u) & = & T_1 & \to^+_{\beta\pi} & T_2 & = & T_3 & \to^+_{\beta\pi} & T_4 \ \dots
\end{array}
$$

Figure 10. Proving PSN for $\lambda\mathsf{lxr}$

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 [Klo80] 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 $x \in FV(M)$.

We abbreviate the term $[\ldots[[N, M_1], M_2], \ldots, M_n]$ by $[N, M_1, M_2, \ldots, M_n]$ or simply by $[N, \langle M \rangle]$ assuming that this expression is equal to N when $n = 0$. The term M and the notation $\langle M \rangle$ inside $[N, \langle M \rangle]$ must not be confused.

The following property is straightforward by induction on terms.

Lemma 16 (Substitutions [Klo80]). *Let M, N, L be Λ_I-terms. Then $M\{x\backslash N\}$ is also a Λ_I-term and $M\{x\backslash N\}\{y\backslash L\} = M\{y\backslash L\}\{x\backslash N\{y\backslash L\}\}$ provided there is no variable capture.*

In what follows we consider two reduction rules on Λ_I-terms:

$$\begin{array}{lll}(\lambda x.M)\ N & \to_\beta & M\{x\backslash N\} \\ [M, N]\ L & \to_\pi & [M\ L, N]\end{array}$$

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

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

$$\frac{t\ \mathcal{I}\ T \quad u\ \mathcal{I}\ U}{t\langle x\backslash u\rangle\ \mathcal{I}\ T\{x\backslash U\}} \qquad \frac{t\ \mathcal{I}\ T}{\mathcal{C}_x^{y|z}(t)\ \mathcal{I}\ T\{y\backslash x\}\{z\backslash x\}} \qquad \frac{t\ \mathcal{I}\ T}{\mathcal{W}_x(t)\ \mathcal{I}\ T}\ x \in FV(T)$$

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

Lemma 17. *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. *$\mathcal{R}_\Psi^\Phi(t)\ \mathcal{I}\ \mathcal{R}_\Psi^\Phi(M)$*

Proof. Property (1) is a straightforward induction on the proof tree as well as Property (2) which also uses Lemma 16. Properties (3) and (5) are also proved by induction on the tree, using Lemma 16. For Property (4):

– If $\mathcal{W}_x(\mathcal{W}_y(t))\ \mathcal{I}\ M$ then $M = [[T, \langle T_i \rangle], \langle U_i \rangle]$ with $t\ \mathcal{I}\ T$, $y \in FV(T)$ and $x \in FV([T, \langle T_i \rangle])$. Then $\mathcal{W}_y(\mathcal{W}_x(t))\ \mathcal{I}\ M$.

- If $t\langle x\backslash u\rangle\langle y\backslash v\rangle \mathcal{I} M$ with $y \notin FV(u)$, then $M = [[T\{x\backslash U\}, \langle T_i\rangle]\{y\backslash V\}, \langle U_i\rangle]$ with $t \mathcal{I} T$, $u \mathcal{I} U$ and $v \mathcal{I} V$. By α-equivalence we can assume that $x \notin FV(T_1) \cup \ldots \cup FV(T_m) \cup FV(V)$, so that $M = [[T, \langle T_i\rangle]\{x\backslash U\}\{y\backslash V\}, \langle U_i\rangle] = [[T, \langle T_i\rangle]\{y\backslash V\}\{x\backslash U\{y\backslash V\}\}, \langle U_i\rangle]$. As a consequence $t\langle y\backslash v\rangle\langle x\backslash u\rangle \mathcal{I} M$, since by (3) we get $u \mathcal{I} U\{y\backslash V\}$.
- Associativity and commutativity of contraction are very similar to the previous case.
- If $\mathcal{C}_w^{y|z}(p)\langle x\backslash u\rangle \mathcal{I} M$, then $M = [[P\{y\backslash w\}\{z\backslash w\}, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$, with $p \mathcal{I} P$ and $u \mathcal{I} U$. We then have $M = [P\{x\backslash U\}\{y\backslash w\}\{z\backslash w\}, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$ so that we can conclude $\mathcal{C}_w^{y|z}(p\langle x\backslash u\rangle) \mathcal{I} M$.

Theorem 8 (Simulation in Λ_I).

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

Proof. By induction on the reduction step. Remark that the case $t \cong t'$ is already considered by Lemma 17-4 so that we restrict the proof here to basic reduction steps.

- B: $(\lambda x.p)\ u \rightarrow p\langle x\backslash u\rangle$.
  Then $T = [[\lambda x.P, \langle P_i\rangle]U, \langle U_i\rangle]$ with $p \mathcal{I} P$ and $u \mathcal{I} U$. We then obtain the reduction sequence $T \rightarrow^*_\pi [(\lambda x.P)U, \langle P_i\rangle, \langle U_i\rangle] \rightarrow_\beta [P\{x\backslash U\}, \langle P_i\rangle, \langle U_i\rangle] = T'$.
- Abs: $(\lambda y.p)\langle x\backslash u\rangle \rightarrow \lambda y.p\langle x\backslash u\rangle$. Then $T = [[\lambda y.P, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$ with $p \mathcal{I} P$ and $u \mathcal{I} U$. We have $T = [\lambda y.(P\{x\backslash U\}), \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$.
- $\mathsf{App1}$,$\mathsf{App2}$: Similar to the previous case.
- Var: $x\langle x\backslash u\rangle \rightarrow u$. Then $T = [[x, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$ with $u \mathcal{I} U$. We have $T = [U, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$.
- $\mathsf{Weak1}$: $\mathcal{W}_x(p)\langle x\backslash u\rangle \rightarrow \mathcal{W}_{FV(u)}(p)$.
  Then $T = [[P, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$ with $p \mathcal{I} P$, $u \mathcal{I} U$, and $x \in FV(P)$. We have $T = [P\{x\backslash U\}, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$. Since $x \notin FV(p)$, then by Lemma 17-3 $p \mathcal{I} P\{x\backslash U\}$, and since $x \in FV(P)$, $FV(U) \subseteq FV(P\{x\backslash U\})$. By Lemma 17-1 $FV(u) \subseteq FV(U)$ so that $FV(u) \subseteq FV(P\{x\backslash U\})$ concludes the proof.
- $\mathsf{Weak2}$: $\mathcal{W}_y(p)\langle x\backslash u\rangle \rightarrow \mathcal{W}_y(p\langle x\backslash u\rangle)$.
  Then $T = [[P, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$ with $p \mathcal{I} P$, $u \mathcal{I} U$, and $y \in FV(P)$. We have $T = [P\{x\backslash U\}, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$ and we still have $y \in FV(P\{x\backslash U\})$.
- Cont: $\mathcal{C}_x^{y|z}(p)\langle x\backslash u\rangle \rightarrow \mathcal{C}_\Phi^{\Psi|\Upsilon}(p\langle y\backslash \mathcal{R}_\Psi^\Phi(u)\rangle\langle z\backslash \mathcal{R}_\Upsilon^\Phi(u)\rangle)$.
  Then $T = [[P\{y\backslash x\}\{z\backslash x\}, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$ with $p \mathcal{I} P$ and $u \mathcal{I} U$. We obtain the following equality $T = [P\{y\backslash U\}\{z\backslash U\}, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$ which can be expressed as

  $$T = [P\{y\backslash U'\}\{z\backslash U''\}\{\Psi\backslash\Phi\}\{\Upsilon\backslash\Phi\}, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$$

  where $U' = U\{\Phi\backslash\Psi\}$ and $U'' = U\{\Phi\backslash\Upsilon\}$. We obtain $\mathcal{R}_\Psi^\Phi(u) \mathcal{I} U'$ and $\mathcal{R}_\Upsilon^\Phi(u) \mathcal{I} U''$ by Lemma 17-5.

– Comp: $p\langle y\backslash v\rangle\langle x\backslash u\rangle \rightarrow\ p\langle y\backslash v\langle x\backslash u\rangle\rangle$ where $x \in FV(v)$. Then we necessarily have $T = [[P\{y\backslash Q\}, \langle P_i\rangle]\{x\backslash U\}, \langle U_i\rangle]$ with $t\ \mathcal{I}\ P$, $v\ \mathcal{I}\ Q$, and $u\ \mathcal{I}\ U$.
Since $T = [P\{x\backslash U\}\{y\backslash Q\{x\backslash U\}\}, \langle P_i\{x\backslash U\}\rangle, \langle U_i\rangle]$, then we obtain $t\ \mathcal{I}\ P\{x\backslash U\}$ by Lemma 17-3.
– WAbs, $\mathsf{WApp1}$, $\mathsf{WApp2}$, WSubs, Cross are straightforward because the condition $x \in FV(P)$ that is checked by $\mathcal{W}_x()$ 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: $\mathcal{C}^{y|z}_w(\mathcal{W}_y(p)) \rightarrow\ \mathcal{R}^z_w(p)$.
Then $T = [[P, \langle P_i\rangle]\{y\backslash w\}\{z\backslash w\}, \langle U_i\rangle]$ with $t\ \mathcal{I}\ P$ and $y \in FV(P)$. We then have the equality $T = [[P\{z\backslash w\}, \langle P_i\{z\backslash w\}\rangle]\{y\backslash w\}, \langle U_i\rangle]$ and we conclude by Lemma 17-3.
– CAbs: $\mathcal{C}^{y|z}_w(\lambda x.t) \rightarrow\ \lambda x.\mathcal{C}^{y|z}_w(p)$.
Then $T = [[\lambda x.P, \langle P_i\rangle]\{y\backslash w\}\{z\backslash w\}, \langle U_i\rangle]$ with $t\ \mathcal{I}\ P$.
We have $T = [\lambda x.(P\{y\backslash w\}\{z\backslash w\}), \langle P_i\{y\backslash w\}\{z\backslash w\}\rangle, \langle U_i\rangle]$.
– $\mathsf{CApp1}$, $\mathsf{CApp2}$: Similar to the previous case.
– CSubs: We have $\mathcal{C}^{y|z}_w(p\langle x\backslash u\rangle)\ \mathcal{I}\ [[P\{x\backslash U\}, \langle P_i\rangle]\{y\backslash w\}\{z\backslash w\}, \langle U_i\rangle]$ which is equal to $T = [[P\{y\backslash w\}\{z\backslash w\}\{x\backslash U\{y\backslash w\}\{z\backslash w\}\}, \langle P_i\rangle\{y\backslash w\}\{z\backslash w\}], \langle U_i\rangle]$ by Lemma 16. We have $p\langle x\backslash \mathcal{C}^{y|z}_w(u)\rangle\ \mathcal{I}\ T$ by Lemma 17-3, which concludes this case.

Now for the closure under context, we use the fact that if $P \rightarrow_{\beta\pi}\ P'$ then $P\{x\backslash U\} \rightarrow_{\beta\pi}\ P'\{x\backslash U\}$, and if moreover $x \in FV(P)$ and $U \rightarrow_{\beta\pi}\ U'$ then $P\{x\backslash U\}\rightarrow^+_{\beta\pi} P\{x\backslash U'\}$. The latter is useful for the closure: if $p\langle x\backslash t\rangle\ \mathcal{I}\ Q$ and $t \rightarrow_B\ t'$, then $Q = [P\{x\backslash T\}, \langle U_i\rangle]$ with $p\ \mathcal{I}\ P$, $t\ \mathcal{I}\ T$ and by the i.h. we get $T\rightarrow^+_{\beta\pi} T'$ such that $t'\ \mathcal{I}\ T'$. Since $x \in FV(p)$, $x \in FV(P)$ by Lemma 17-1, and hence $Q\rightarrow^+_{\beta\pi} [P\{x\backslash T'\}, \langle U_i\rangle]$.

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

$$\sigma : t\rightarrow^*_{\mathsf{xr}}\ \rightarrow_B\ t_1 \cdots \rightarrow^*_{\mathsf{xr}}\ \rightarrow_B\ t_i\rightarrow^*_{\mathsf{xr}}\ \rightarrow_B\ \cdots$$

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

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

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 $i(t) \in WN_{\beta\pi}$.*

Theorem 10 ([Ned73]). *If $i(t) \in WN_{\beta\pi}$ then $i(t) \in SN_{\beta\pi}$.*

Theorem 11. *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 the i.h. Therefore, we obtain either $\lambda x.\mathcal{A}(t)\ \mathcal{I}\ \lambda x.i(t)$ or $\lambda x.\mathcal{W}_x(\mathcal{A}(t))\ \mathcal{I}\ \lambda x.[i(t), x]$.
- If $u = (t\ u)$, then $\mathcal{A}(t)\ \mathcal{I}\ i(t)$ and $\mathcal{A}(u)\ \mathcal{I}\ i(u)$ hold by the i.h. so that $\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(t))\ \mathcal{I}\ \mathcal{R}^{\Phi}_{\Upsilon}(i(t))$ and $\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(u))\ \mathcal{I}\ \mathcal{R}^{\Phi}_{\Upsilon}(i(u))$ hold by Lemma 17-5. Since $\mathcal{R}^{\Phi}_{\Upsilon}(i(t))\{\Upsilon \backslash \Phi\} = i(t)$ and $\mathcal{R}^{\Phi}_{\Upsilon}(i(u))\{\Upsilon \backslash \Phi\} = i(u)$, we can then conclude $\mathcal{C}^{\Psi|\Upsilon}_{\Phi}(\mathcal{R}^{\Phi}_{\Psi}(\mathcal{A}(t))\ \mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(u)))\ \mathcal{I}\ i(t)\ i(u)$.

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 Theorems 9 and 10. As $\mathcal{A}(t)\ \mathcal{I}\ i(t)$ by Theorem 11, 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 λlxr 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 12. *If t is an* xr*-normal form, then $t \equiv \mathcal{W}_{\mathcal{FV}(t)\setminus\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.*

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 $t = \lambda x.u$, then we know $u \equiv \mathcal{W}_{\mathcal{FV}(u)\setminus\mathcal{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u)))$ by the i.h. But t is an xr-normal form, so $\mathcal{FV}(u) \setminus \mathcal{FV}(\mathcal{B}(u)) \subseteq \{x\}$, otherwise it can be reduced by WAbs. Now, if $\mathcal{FV}(u) \setminus \mathcal{FV}(\mathcal{B}(u)) = \emptyset$, then also $\mathcal{FV}(t) \setminus \mathcal{FV}(\mathcal{B}(t)) = \emptyset$ and the claim $t \equiv \mathcal{A}(\mathcal{B}(\lambda x.u))$ immediately holds. Otherwise, $\mathcal{FV}(u) \setminus \mathcal{FV}(\mathcal{B}(u)) = \{x\}$ and $t \equiv \lambda x.\mathcal{W}_x(\mathcal{A}(\mathcal{B}(u))) = \mathcal{A}(\mathcal{B}(t))$.
- If $t = u\ v$, $t \equiv \mathcal{W}_{\mathcal{FV}(u)\setminus\mathcal{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u)))\ \mathcal{W}_{\mathcal{FV}(v)\setminus\mathcal{FV}(\mathcal{B}(v))}(\mathcal{A}(\mathcal{B}(v)))$ by the 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 WApp1). 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 $t = u\langle x\backslash v\rangle$ is not possible by Lemma 2.
- If $t = \mathcal{W}_x(u)$, $t \equiv \mathcal{W}_x(\mathcal{W}_{\mathcal{FV}(u)\backslash\mathcal{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u))))$ by the i.h. This last term is equal to $\mathcal{W}_{\mathcal{FV}(t)\backslash\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$ since $x \in \mathcal{FV}(t)$ but $x \notin \mathcal{FV}(\mathcal{B}(t))$.
- If $t = \mathcal{C}_x^{y|z}(u)$, $t \equiv \mathcal{C}_x^{y|z}(\mathcal{W}_{\mathcal{FV}(u)\backslash\mathcal{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u))))$ by the i.h.
  We first remark that y and z are free in u since t is linear, and also x is not free in u, hence neither is it free in $\mathcal{B}(u)$.
  Secondly, since t is an xr-normal form, we have $\mathcal{FV}(u) \setminus \mathcal{FV}(\mathcal{B}(u)) = \emptyset$ (otherwise t could be reduced by Cross or Merge). Hence, y and z are free in $\mathcal{B}(u)$ and $t \equiv \mathcal{C}_x^{y|z}(\mathcal{A}(\mathcal{B}(u)))$.
  But $\mathcal{B}(t) = \mathcal{B}(u)\{y\backslash x\}\{z\backslash x\}$, so x is free in $\mathcal{B}(t)$. We conclude $\mathcal{FV}(t) = \mathcal{FV}(\mathcal{B}(t))$.
  Third, notice that $\mathcal{B}(u)$ can be neither a variable (otherwise t would not be linear) nor an abstraction (otherwise t could be reduced by CAbs), so $\mathcal{B}(u) = w\ v$,
  and $\mathcal{A}(\mathcal{B}(u)) = \mathcal{C}_\Phi^{\Upsilon|\Psi}(\mathcal{R}_\Upsilon^\Phi(\mathcal{A}(w))\ \mathcal{R}_\Psi^\Phi(\mathcal{A}(v)))$ with $\Phi = \mathcal{FV}(w) \cap \mathcal{FV}(v)$. Hence, $t \equiv \mathcal{C}_x^{y|z}(\mathcal{C}_\Phi^{\Upsilon|\Psi}(\mathcal{R}_\Upsilon^\Phi(\mathcal{A}(w))\ \mathcal{R}_\Psi^\Phi(\mathcal{A}(v))))$.
  Now it would suffice that $y \in \mathcal{FV}(w) \setminus \mathcal{FV}(v)$ and $z \in \mathcal{FV}(v) \setminus \mathcal{FV}(w)$ to prove that this term is in fact

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

  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 $w\ v$.
  Suppose that one of them, say y, is both in w and in v. Then $y \in \Phi$, so

  $$t \equiv \mathcal{C}_x^{y|z}(\mathcal{C}_{\Phi',y}^{(\Upsilon',y')|(\Psi',y'')}(\mathcal{R}_\Upsilon^\Phi(\mathcal{A}(w))\ \mathcal{R}_\Psi^\Phi(\mathcal{A}(v))))$$

  which we can rearrange into

  $$t \equiv \mathcal{C}_x^{y|y''}(\mathcal{C}_{\Phi',y}^{(\Upsilon',y')|(\Psi',z)}(\mathcal{R}_\Upsilon^\Phi(\mathcal{A}(w))\ \mathcal{R}_\Psi^\Phi(\mathcal{A}(v))))$$

  if $z \in \mathcal{FV}(w)$, or $t \equiv \mathcal{C}_x^{y|y'}(\mathcal{C}_{\Phi',y}^{(\Upsilon',z)|(\Psi',y'')}(\mathcal{R}_\Upsilon^\Phi(\mathcal{A}(w))\ \mathcal{R}_\Psi^\Phi(\mathcal{A}(v))))$ if $z \in \mathcal{FV}(v)$.
  In the first case, t can be reduced by $\mathsf{CApp1}$ (on $\mathcal{C}_y^{y'|z}()$), and in the second by $\mathsf{CApp2}$ (on $\mathcal{C}_y^{z|y''}()$). 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 w. Then t can be reduced by $\mathsf{CApp1}$ on $\mathcal{C}_x^{y|z}()$. Similarly, they cannot be both in v (t could be reduced by $\mathsf{CApp2}$). Hence one of them is only in w, and the other is only in v, as required.

Lemma 18. *The system* xr *is confluent and terminating, and the* xr*-normal form of* t *is* $\mathcal{W}_{\mathcal{FV}(t)\backslash\mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

Proof. By Theorem 1 the system xr is terminating so that we can take any xr-normal form t' of t such that $t \rightarrow^*_{\mathsf{xr}} t'$. We then have $\mathcal{FV}(t) = \mathcal{FV}(t')$ by Lemma 1 and $\mathcal{B}(t) = \mathcal{B}(t')$ by Lemma 14. Since t' is an xr-normal form, then $t' \equiv \mathcal{W}_{\mathcal{FV}(t') \setminus \mathcal{FV}(\mathcal{B}(t'))}(\mathcal{A}(\mathcal{B}(t')))$ by Theorem 12. Hence $t' \equiv \mathcal{W}_{\mathcal{FV}(t) \setminus \mathcal{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

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

Theorem 13. *The system* $\lambda\mathsf{lxr}$ *is confluent.*

Proof. Suppose $N \equiv N'$, $N \rightarrow^*_{\lambda\mathsf{lxr}} N_1$ and $N' \rightarrow^*_{\lambda\mathsf{lxr}} N_2$. We get $\mathcal{B}(N) = \mathcal{B}(N')$ and $\mathcal{B}(N) \rightarrow^*_{\beta} \mathcal{B}(N_1)$ and $\mathcal{B}(N') \rightarrow^*_{\beta} \mathcal{B}(N_2)$ using Lemma 14. Confluence of $\rightarrow_{\beta}$ gives us a λ-term M such that $\mathcal{B}(N_1) \rightarrow^*_{\beta} M$ and $\mathcal{B}(N_2) \rightarrow^*_{\beta} M$. By Theorem 7, there exist two lists of variables Φ_1, Φ_2 such that $\mathcal{A}(\mathcal{B}(N_1)) \rightarrow^*_{\lambda\mathsf{lxr}} \mathcal{W}_{\Phi_1}(\mathcal{A}(M))$ and $\mathcal{A}(\mathcal{B}(N_2)) \rightarrow^*_{\lambda\mathsf{lxr}} \mathcal{W}_{\Phi_2}(\mathcal{A}(M))$.

We can now reduce N_1 and N_2 to their respective xr-normal form so that Lemma 18 provides Φ'_1 and Φ'_2 such that

$$N_1 \rightarrow^*_{\mathsf{xr}} \mathcal{W}_{\Phi'_1}(\mathcal{A}(\mathcal{B}(N_1))) \rightarrow^*_{\lambda\mathsf{lxr}} \mathcal{W}_{\Upsilon_1}(\mathcal{A}(M))$$

$$N_2 \rightarrow^*_{\mathsf{xr}} \mathcal{W}_{\Phi'_2}(\mathcal{A}(\mathcal{B}(N_2))) \rightarrow^*_{\lambda\mathsf{lxr}} \mathcal{W}_{\Upsilon_2}(\mathcal{A}(M))$$

for $\Upsilon_1 = \Phi'_1, \Phi_1$ and $\Upsilon_2 = \Phi'_2, \Phi_2$.

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

$$\begin{array}{l}\mathcal{FV}(\mathcal{W}_{\Upsilon_1}(\mathcal{A}(M))) = \mathcal{FV}(N_1) = \mathcal{FV}(N) = \mathcal{FV}(N') \\ \mathcal{FV}(\mathcal{W}_{\Upsilon_2}(\mathcal{A}(M))) = \mathcal{FV}(N_2) = \mathcal{FV}(N') = \mathcal{FV}(N)\end{array}$$

We can conclude that Υ_1 is a permutation of Υ_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 (typed) calculus with operators for erasure, duplication and substitution.

Our term calculus is expressed by a simple syntax, and enjoys natural operational semantics via a well-established [Ter03] notion of reduction modulo a set of equations. Soundness and completeness of (typed) $\lambda\mathsf{lxr}$ are shown with respect to its proof-nets model.

Although this paper uses typed proof-nets, our (untyped) system could also be put in relation with an appropriate notion of reduction in formalism of *untyped* proof-nets. This would provide soundness and completeness w.r.t. an untyped model. However, one of the main results of this paper is strong normalisation of typed λlxr-terms, and this can only be obtained from typed proof-nets, which we therefore chose to use for this paper.

In contrast to other term calculi in the literature, λlxr has full composition and enjoys PSN. Moreover, λlxr enjoys confluence, strong normalisation of simply-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.

Weakening operators are a useful tool to handle 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 how the following rules $\{\mathsf{App2}, \mathsf{Comp}, \mathsf{Var}, \mathsf{Weak1}, \mathsf{Cont}\}$ tightly correspond to the manipulation of boxes in proof-nets. More precisely, App2 and Comp in λlxr correspond to b-b in PN, Var to d-b, Weak1 to w-b and Cont to c-b.

It is worth mentioning the calculus obtained by turning the equation $\mathsf{P_{cs}}$ 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 2,4,7,1,13, and Corollary 3. However, these rules seem to be necessary for the confluence on meta terms (ongoing work).

We think that many points raised in this work deserve further development. The first one concerns the study of reduction strategies well-adapted to handle the operators for 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], labelled RPO [BG99], PSN by standardisation [KOvO01], or intersection types [DL03]) are not all easy to adapt/extend to reduction modulo and other formalisms. We believe that the proof technique used here is really flexible since only a small part of the proof depends on the calculus itself: the simulation result 8 and the start relation result (Theorem 11). We think however that a more direct proof of PSN would be possible by incorporating a memory operator inside λlxr in such a way that the resulting calculus becomes non-erasing. This would require for example a proof to show that weak normalisation implies strong normalisation in the new calculus.

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 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 calculus even closer to the proof-nets model which has no notion of binding.

Connections with similar approaches relating graph formalisms to term calculi, as for example that of Hasegawa [Has99] also merits further investigations.

Another interesting issue would be to study the combination of $\lambda\mathsf{lxr}$ with other (higher-order) reduction systems, particularly with eta-reduction (contraction or expansion) rules. Because of the linearity constraints, no additional side-condition is needed to specify such kind of rules $\lambda x.t\ x \leftrightarrow^* t$.

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 Theoret. Comput. Sci.* 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. *Math. Structures Comput. Sci.*, 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. *Math. Structures Comput. Sci.*, 13(3):409–450, 2003.

[DG01] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitution. *Math. Structures Comput. Sci.*, 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.

[DL03] D. Dougherty and P. Lescanne. Reductions, intersection types, and explicit substitutions. *Math. Structures Comput. Sci.*, 13(1):55–85, 2003.

[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.

[Gue99] S. Guerrini. A general theory of sharing graphs. *Theoret. Comput. Sci.*, 227(1–2):99–151, 1999.

[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.

[Has99] M. Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of* `let` *and* `letrec`. Distinguished Dissertation Series. Springer-Verlag, Kyoto University, Japan, 1999. PhD Thesis.

[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`.

[Hue80] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[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.

[Laf95] Y. Lafont. From proof-nets to interaction nets. In *Advances in Linear Logic*, volume 222 of *London Mathematical Society, Lecture Notes*, pages 225–247. Cambridge University Press, 1995.

[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. Available at `http://hal.ccsd.cnrs.fr/ccsd-00004358`.

[Len06] S. Lengrand. *Normalisation and Equivalence in Proof Theory and Type Theory.* PhD thesis, Université Denis Diderot Paris VII and University of St Andrews, 2006.

[LLD$^+$04] S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Inform. and Comput.*, 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.

[Ned73] R. Nederpelt. *Strong Normalization in a Typed Lambda Calculus with Lambda Structured Types.* PhD thesis, Eindhoven University of Technology, 1973.

[Nip91] T. Nipkow. Higher-order critical pairs. In *6th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 342–349. IEEE Computer Society Press, July 1991.

[OH06] Y. Ohta and M. Hasegawa. A terminating and confluent linear lambda calculus. In F. Pfenning, editor, *17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science. Springer-Verlag, Aug. 2006. To appear.

[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.

[Ros96] K. Rose. Explicit substitution - tutorial & survey. Available as `http://www.brics.dk/LS/96/3/BRICS-LS-96-3/BRICS-LS-96-3.html`, 1996.

[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. *Inform. and Comput.*, 37:35–71, 1997.

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoret. Comput. Sci.* Cambridge University Press, 2003.

[TS00] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, 2000.

[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. *Math. Structures Comput. Sci.*, 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.

Remark 1 The notions of multiplicity and term complexity are invariant under conversion by $\equiv$.

Proof. Indeed, as two non-trivial cases, let us consider the case $\mathcal{C}_w^{x|v}(\mathcal{C}_x^{y|z}(t)) \equiv \mathcal{C}_w^{x|y}(\mathcal{C}_x^{z|v}(t))$ for which we have:

$$\mathcal{M}_w(\mathcal{C}_w^{x|v}(\mathcal{C}_x^{y|z}(t))) = \mathcal{M}_y(t) + \mathcal{M}_z(t) + \mathcal{M}_v(t) + 2 = \mathcal{M}_w(\mathcal{C}_w^{x|y}(\mathcal{C}_x^{z|v}(t)))$$

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

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

  We then obtain

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

Lemma 3 (Decrease of multiplicities and term complexities).

- If $t \rightarrow_{\mathsf{xr}} u$, then for all $w \in \mathcal{FV}(t)$, $\mathcal{M}_w(t) \geq \mathcal{M}_w(u)$.
- If $t \rightarrow_{\mathsf{xr}} u$, then $\mathcal{S}(t) \geq \mathcal{S}(u)$. Moreover,
  if $t \rightarrow_{\mathsf{Var,Weak1,Cont,Comp}} u$, then $\mathcal{S}(t) > \mathcal{S}(u)$.

Proof.
- Since the congruence steps preserve the multiplicity, we only have to consider the basic reduction relation. This is done by induction on the reduction step, the base cases being shown in Figure 11. Note that we use the fact that $\mathcal{M}_x(t) > 0$ (provided $x \in \mathcal{FV}(t)$) and $\mathcal{S}(t) > 0$.
- Since the congruence steps preserve the term complexity, we only have to consider the basic reduction relation. The proof can be done by structural induction on terms. The inductive cases are straightforward by using by the first point. We show in Figure 12 the root reductions.
  The last line holds because the term complexity measure forgets weakenings, contractions, abstractions and applications.

	Left-hand side		Right-hand side
(Var)	$x\langle x\backslash u\rangle$	$\to$	u
	$\mathcal{M}_w(u)+1$	$>$	$\mathcal{M}_w(u)$
(Weak1)	$\mathcal{W}_x(t)\langle x\backslash u\rangle$	$\to$	$\mathcal{W}_{FV(u)}(t)$
$w \in FV(u)$	$\mathcal{M}_w(u)+1$	$>$	1
$w \in FV(t)\setminus\{x\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
(Cont)	$\mathcal{C}_x^{y\vert z}(t)\langle x\backslash u\rangle$	$\to$	$\mathcal{C}_{\Phi}^{\Psi\vert\Upsilon}(t\langle y\backslash\mathcal{R}_{\Psi}^{\Phi}(u)\rangle\langle z\backslash\mathcal{R}_{\Upsilon}^{\Phi}(u)\rangle)$
$w \in FV(u)=\Phi$	$(\mathcal{M}_y(t)+\mathcal{M}_z(t)+1)\cdot(\mathcal{M}_w(u)+1)$	$>$	$\mathcal{M}_y(t)\cdot(\mathcal{M}_w(u)+1)+\mathcal{M}_z(t)\cdot(\mathcal{M}_w(u)+1)+1$
$w \in FV(t)\setminus\{x,y,z\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
(Comp)	$t\langle y\backslash v\rangle\langle x\backslash u\rangle$	$\to$	$t\langle y\backslash v\langle x\backslash u\rangle\rangle$
$w \in FV(t)\setminus\{y\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w \in FV(v)\setminus\{x\}$	$\mathcal{M}_y(t)\cdot(\mathcal{M}_w(v)+1)$	$=$	$\mathcal{M}_y(t)\cdot(\mathcal{M}_w(v)+1)$
$w \in FV(u)$	$\mathcal{M}_y(t)\cdot(\mathcal{M}_x(v)+1)\cdot(\mathcal{M}_w(u)+1)$	$>$	$\mathcal{M}_y(t)\cdot(\mathcal{M}_x(v)\cdot(\mathcal{M}_w(u)+1)+1)$
Other x	$t\langle x\backslash u\rangle$	$\to$	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)	$\mathcal{C}_{w'}^{y\vert z}(\mathcal{W}_y(t))$	$\to$	$\mathcal{R}_{w'}^{z}(t)$
$w=w'$	$\mathcal{M}_z(t)+2$	$>$	$\mathcal{M}_z(t)$
$w\neq w'$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
(WSubs)	$t\langle x\backslash\mathcal{W}_y(u)\rangle$	$\to$	$\mathcal{W}_y(t\langle x\backslash u\rangle)$
$w \in FV(t)\setminus\{x\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w=y$	$\mathcal{M}_x(t)\cdot(1+1)$	$>$	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)	$\mathcal{C}_z^{x\vert y}(t\langle y'\backslash u\rangle)$	$\to$	$t\langle y'\backslash\mathcal{C}_z^{x\vert y}(u)\rangle$
$w \in FV(t)\setminus\{y'\}$	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t)$
$w=z$	$\mathcal{M}_{y'}(t)\cdot(\mathcal{M}_x(u)+\mathcal{M}_y(u)+2)+1$	$>$	$\mathcal{M}_{y'}(t)\cdot(\mathcal{M}_x(u)+\mathcal{M}_y(u)+1+1)$
$w \in FV(u)$	$\mathcal{M}_{y'}(t)\cdot(\mathcal{M}_w(u)+1)$	$=$	$\mathcal{M}_{y'}(t)\cdot(\mathcal{M}_w(u)+1)$
Other r	t	$\to$	t'
	$\mathcal{M}_w(t)$	$=$	$\mathcal{M}_w(t')$

Figure 11. Decrease of multiplicities

Left-hand side		Right-hand side
$x\langle x\backslash u\rangle$	$\to_{\mathsf{Var}}$	u
$1+\mathcal{S}(u)$	$>$	$\mathcal{S}(u)$
$\mathcal{W}_x(t)\langle x\backslash u\rangle$	$\to_{\mathsf{Weak1}}$	$\mathcal{W}_{FV(u)}(t)$
$\mathcal{S}(t)+\mathcal{S}(u)$	$>$	$\mathcal{S}(t)$
$\mathcal{C}_x^{y\vert z}(t)\langle x\backslash u\rangle$	$\to_{\mathsf{Cont}}$	$\mathcal{C}_{\Phi}^{\Psi\vert\Upsilon}(t\langle y\backslash\mathcal{R}_{\Psi}^{\Phi}(u)\rangle\langle z\backslash\mathcal{R}_{\Upsilon}^{\Phi}(u)\rangle)$
$\mathcal{S}(t)+\mathcal{S}(u)\cdot(\mathcal{M}_y(t)+\mathcal{M}_z(t)+1)$	$>$	$\mathcal{S}(t)+\mathcal{S}(u)\cdot\mathcal{M}_y(t)+\mathcal{S}(u)\cdot\mathcal{M}_z(t)$
$t\langle y\backslash v\rangle\langle x\backslash u\rangle$	$\to_{\mathsf{Comp}}$	$t\langle y\backslash v\langle x\backslash u\rangle\rangle$
$\mathcal{S}(t)+\mathcal{M}_y(t)\cdot(\mathcal{S}(v)+(\mathcal{M}_x(v)+1)\cdot\mathcal{S}(u))$	$>$	$\mathcal{S}(t)+\mathcal{M}_y(t)\cdot(\mathcal{S}(v)+\mathcal{M}_x(v)\cdot\mathcal{S}(u))$
t	$\to_{\text{Other }\mathsf{xr}}$	t'
$\mathcal{S}(t)$	$=$	$\mathcal{S}(t')$

Figure 12. Decrease of term complexity

Remark 2 The polynomial interpretation $\mathcal{I}(\_)$ is invariant under conversion by $\equiv$.

Proof. The polynomial interpretation is blind to the variables' names, so it is trivially sound with respect to α-equivalence, and rules $\mathsf{A_c}$, $\mathsf{C_c}$, $\mathsf{P_c}$ and $\mathsf{P_w}$. For the equivalence rule $\mathsf{P_s}$ we have by commutativity of multiplication the following equality:

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

For the equivalence rule $\mathsf{P_{cs}}$ we have:

$$\mathcal{I}(\mathcal{C}_w^{y|z}(t)\langle x\backslash u\rangle) = 2\cdot\mathcal{I}(t)\cdot\mathcal{I}(u) + 2\cdot\mathcal{I}(t) = \mathcal{I}(\mathcal{C}_w^{y|z}(t\langle x\backslash u\rangle))$$

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

Proof. Since the congruence steps preserve the interpretation, we only have to consider the basic 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)	$2\cdot\mathcal{I}(t)\cdot(\mathcal{I}(u)+1)$	$>$	$\mathcal{I}(t)\cdot(2\cdot\mathcal{I}(u)+1)$

Lemma 8 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 $t \cong \mathcal{W}_x(t')$ and $|t| = |\mathcal{W}_x(t')|$.
2. If f is a **C**-node whose c edge is labelled with x, then there exist a term t' such that $t \cong \mathcal{C}_x^{y|z}(t')$ and $|t| = |\mathcal{C}_x^{y|z}(t')|$.
3. If f is an $\wp$-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 7 and thus assume t to be of a particular shape (without increasing its size).

- If $t \cong \lambda y.u$, then if f is a final-modulo $\parr$-node of $NT(t)$, (3) trivially holds. Otherwise, f is a final-modulo node of $NT(u)$.
  If f is a $\mathbf{W}$-node, we have $u \cong \mathcal{W}_x(u')$ and $|u| = |\mathcal{W}_x(u')|$ by the i.h.(1) so that $t \cong \lambda y.\mathcal{W}_x(u')$ and $|t| = |\lambda y.\mathcal{W}_x(u')|$. If $x \neq y$, then $\lambda y.\mathcal{W}_x(u') \cong_{\mathsf{WAbs}} \mathcal{W}_x(\lambda y.u')$ and we are done. If $x = y$, the $\mathbf{W}$-node f is not final-modulo in $NT(t)$ which leads to a contradiction.
  If f is a $\mathbf{C}$-node, we have $u \cong \mathcal{C}_x^{y|z}(u')$ and $|u| = |\mathcal{C}_x^{y|z}(u')|$ by the i.h.(2) so that $t \cong \lambda w.\mathcal{C}_x^{y|z}(u')$ and $|t| = |\lambda w.\mathcal{C}_x^{y|z}(u')|$. If $x \neq w$, then $\lambda w.\mathcal{C}_x^{y|z}(u') \cong_{\mathsf{CAbs}} \mathcal{C}_x^{y|z}(\lambda x.u')$ and we are done. If $x = w$, the $\mathbf{C}$-node f is not final-modulo in $NT(t)$ which leads to a contradiction.
- If $t \cong \mathcal{W}_y(u)$, then if f is a $\mathbf{W}$-node labelled with $x = y$, (1) trivially holds. Otherwise, $x \neq y$ and f is also final-modulo in $NT(u)$.
  If f is a $\mathbf{W}$-node labelled by a variable $x \neq y$, we have $u \cong \mathcal{W}_x(u')$ and $|u| = |\mathcal{W}_x(u')|$ by the i.h.(1), thus, $\mathcal{W}_y(u) \cong \mathcal{W}_y(\mathcal{W}_x(u')) \cong_{\mathsf{P_w}} \mathcal{W}_x(\mathcal{W}_y(u'))$ which concludes this case.
  The cases where f is a $\parr$-node or a $\mathbf{C}$-node are very similar.
- If $t \cong \mathcal{C}_w^{y'|z'}(u)$, then if f is a $\mathbf{C}$-node labelled with $x = w$, (2) trivially holds. Otherwise, f is already final-modulo in $NT(u)$.
  If f is a $\mathbf{W}$-node, we have $u \cong \mathcal{W}_x(u')$ and $|u| = |\mathcal{W}_x(u')|$ by the i.h.(1) so that $t \cong \mathcal{C}_w^{y'|z'}(\mathcal{W}_x(u'))$. If $x \neq y', z'$, then $\mathcal{C}_w^{y'|z'}(\mathcal{W}_x(u')) \cong_{Cross} \mathcal{W}_x(\mathcal{C}_w^{y'|z'}(u'))$ and we are done. Otherwise, the $\mathbf{W}$-node f is not final-modulo in $NT(t)$ which leads to a contradiction.
  The cases where f is a $\parr$-node or a $\mathbf{C}$-node labelled with a variable different from w are very similar.
- Suppose $t \cong (xt_1 \dots t_n)\langle x_{n+1}\backslash t_{n+1}\rangle \dots \langle x_m\backslash t_m\rangle$.
  We know that if f is a final-modulo $\mathbf{W}/\mathbf{C}/\parr$ node of $NT(t)$, then either it was already in $T(t)$ or it was created by the TB-reduction of $T(t)$. But the only TB-reductions that can be applied on $T(t)$ are either the *Ax-cut* reduction steps concerning the sequence of applications, which do not create new final-modulo nodes, or reductions inside some box containing $T(t_i)$ for some i. As a consequence, f cannot be a $\parr$-node, and it is also final-modulo in some $NT(t_j)$.
  Hence, we apply the induction hypothesis on t_j to get $t_j \cong \mathcal{W}_x(t'_j)$ (resp. $t_j \cong \mathcal{C}_x^{y|z}(t'_j)$) with the same size as t_j. Let $t'_i = t_i$ for all $i \neq j$. If $j \leq n$, we use rules $\mathsf{WApp2}, \mathsf{WApp1}, \mathsf{Weak2}$ (resp. $\mathsf{CApp2}, \mathsf{CApp1}, \mathsf{P_{cs}}$) to get the congruence $t \cong \mathcal{W}_x((xt'_1 \dots t'_n)\langle x_{n+1}\backslash t'_{n+1}\rangle \dots \langle x_m\backslash t'_m\rangle)$ (resp. the congruence $t \cong \mathcal{C}_x^{y|z}((xt'_1 \dots t'_n)\langle x_{n+1}\backslash t'_{n+1}\rangle \dots \langle x_m\backslash t'_m\rangle))$. Otherwise, we use WSubs and $\mathsf{Weak2}$ (resp. CSubs and $\mathsf{P_{cs}}$).
- Suppose $t \cong \mathcal{W}_v(u)\langle v\backslash u_0\rangle\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$, where the notation $\langle \overrightarrow{x}\backslash\overrightarrow{u}\rangle$ is used to abbreviate $\langle x_1\backslash u_1\rangle \dots \langle x_m\backslash u_m\rangle$ with $m \geq 0$ and $x_i \notin FV(u)$ for all i.

Again, the only TB-reductions that can be applied on $T(t)$ are either in $T(u)$ or inside some box containing $T(u_i)$ for some i.
In the former case we use the i.h. on u to get $u \cong \mathcal{W}_x(u')$ (resp. $u \cong \mathcal{C}_x^{y|z}(u')$ or $u \cong \lambda x.u'$). Then we use rules $\mathsf{Weak2}$ (resp. $\mathsf{Cross}, \mathsf{P_{cs}}$ or $\mathsf{WAbs}, \mathsf{Abs}$) to get $t \cong \mathcal{W}_x(\mathcal{W}_v(u')\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle)$ (resp. $t \cong \mathcal{C}_x^{y|z}(\mathcal{W}_v(u')\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle)$ or $t \cong \lambda x.(\mathcal{W}_v(u')\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle))$.
In the latter case, notice that f cannot be a $⅋$-node and we use the induction hypothesis on u_j to get $u_j \cong \mathcal{W}_x(u'_j)$ (resp. $u_j = \mathcal{C}_x^{y|z}(u'_j)$). Let $u'_i = u_i$ for all $i \neq j$. We use rules $\mathsf{WSubs}, \mathsf{Weak2}$ (resp. $\mathsf{CSubs}, \mathsf{P_{cs}}$), and we get $t \cong \mathcal{W}_x(\mathcal{W}_v(u)\langle v\backslash u'_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u'}\rangle)$ (resp. $t \cong \mathcal{C}_x^{y|z}(\mathcal{W}_v(u)\langle v\backslash u'_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u'}\rangle))$.

– Suppose $t \cong \mathcal{C}_v^{v_1|v_2}(u)\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle$, where the notation $\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle$ is used to abbreviate $\langle x_1\backslash u_1\rangle \ldots \langle x_m\backslash u_m\rangle$ with $m \geq 0$ and $x_i \notin FV(u)$ for all i.
Again, the only TB-reductions that can be applied on $T(t)$ are either inside $T(u)$ or inside a some box containing $T(u_i)$ for some i.
In the former case we use the i.h. on u to get $u \cong \mathcal{W}_x(u')$ (resp. $u \cong \mathcal{C}_x^{y|z}(u')$ or $u \cong \lambda x.u'$). Then we use rules $\mathsf{Cross}, \mathsf{Weak2}$ (resp. $\mathsf{P_{cs}}$ or $\mathsf{CAbs}, \mathsf{Abs}$) to get $t \cong \mathcal{W}_x(\mathcal{C}_v^{v_1|v_2}(u')\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle)$ (resp. $t \cong \mathcal{C}_x^{y|z}(\mathcal{C}_v^{v_1|v_2}(u')\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle)$ or $t \cong \lambda x.(\mathcal{C}_v^{v_1|v_2}(u')\langle v\backslash u_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u}\rangle))$.
In the latter case, notice that f cannot be a $⅋$-node and we use the induction hypothesis on u_j to get $u_j \cong \mathcal{W}_x(u'_j)$ (resp. $u_j = \mathcal{C}_x^{y|z}(u'_j)$). Let $u'_i = u_i$ for all $i \neq j$. We use rules $\mathsf{WSubs}, \mathsf{Weak2}$ (resp. $\mathsf{CSubs}, \mathsf{P_{cs}}$), and we get $t \cong \mathcal{W}_x(\mathcal{C}_v^{v_1|v_2}(u)\langle v\backslash u'_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u'}\rangle)$ (resp. $t \cong \mathcal{C}_x^{y|z}(\mathcal{C}_v^{v_1|v_2}(u)\langle v\backslash u'_0\rangle\langle\overrightarrow{x}\backslash\overrightarrow{u'}\rangle))$.

Lemma 11 For all λ-terms t_1 and t_2 such that $x \in \mathcal{FV}(t_1)$,

$$\mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(t_1))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle) \to_{\mathsf{xr}}^{*} \mathcal{A}(t_1\{x\backslash t_2\})$$

where $\Phi := (\mathcal{FV}(t_1) \setminus \{x\}) \cap \mathcal{FV}(t_2)$, provided that the former term is linear.

Proof. By induction on the size of t_1. We shall always suppose, by Barendregt's convention, that $x \notin \mathcal{FV}(t_2)$. Moreover, whenever we use the induction hypothesis throughout the proof, it will be applied to a term which is linear (Lemma 1, Property 1).

1. If t_1 is a variable, then it must be x, so there is no contraction and

$$x\langle x\backslash\mathcal{A}(t_2)\rangle \to_{\mathsf{Var}} \mathcal{A}(t_2) = \mathcal{A}(x\{x\backslash t_2\})$$

2. If $t_1 = (t\ u)$, then by α-equivalence we can suppose $x \notin \mathcal{FV}(t_2)$, and let
$\Sigma := \mathcal{FV}(t_2) \cap \mathcal{FV}(t) \cap \mathcal{FV}(u)$
$\Lambda := (\mathcal{FV}(t_2) \cap \mathcal{FV}(t)) \setminus (\mathcal{FV}(t_2) \cap \mathcal{FV}(t) \cap \mathcal{FV}(u))$
$\Psi := (\mathcal{FV}(t_2) \cap \mathcal{FV}(u)) \setminus (\mathcal{FV}(t_2) \cap \mathcal{FV}(t) \cap \mathcal{FV}(u))$
$\Xi := (\mathcal{FV}(t) \cap \mathcal{FV}(u)) \setminus (\mathcal{FV}(t_2) \cap \mathcal{FV}(t) \cap \mathcal{FV}(u))$
$\Theta := \mathcal{FV}(t_2) \setminus (\mathcal{FV}(t) \cup \mathcal{FV}(u))$

Note that $\Phi = \mathcal{FV}(t_1) \cap \mathcal{FV}(t_2)$ is a permutation of Σ, Λ, Ψ.
Also note that $\mathcal{FV}(t) \cap \mathcal{FV}(u)$ is a permutation of Σ, Ξ and hence

$$\mathcal{A}(t_1) \equiv \mathcal{C}^{\Sigma_3,\Xi_3|\Sigma_4,\Xi_4}_{\Sigma,\Xi}(\mathcal{R}^{\Sigma,\Xi}_{\Sigma_3,\Xi_3}(\mathcal{A}(t))\ \mathcal{R}^{\Sigma,\Xi}_{\Sigma_4,\Xi_4}(\mathcal{A}(u)))$$

We then have:

$$\begin{array}{l} \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1|\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_1,\Lambda_1,\Psi_1}(\mathcal{A}(t_1))\langle x \backslash \mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(t_2))\rangle) \\ = \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1|\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi_3|\Sigma_4,\Xi_4}_{\Sigma_1,\Xi}(t'\ u')\langle x \backslash \mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(t_2))\rangle) \end{array}$$

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

(a) If $x \in \mathcal{FV}(t) \cap \mathcal{FV}(u)$, then x is necessarily in Ξ (since $x \notin \mathcal{FV}(t_2)$), so Ξ is a permutation of Ξ', x for some list Ξ'. Hence, the contractions $\mathcal{C}^{\Sigma_3,\Xi_3|\Sigma_4,\Xi_4}_{\Sigma_1,\Xi}()$ are equivalent by $\equiv$ to $\mathcal{C}^{\Sigma_3,\Xi'_3|\Sigma_4,\Xi'_4}_{\Sigma_1,\Xi'}(\mathcal{C}^{x_3|x_4}_{x}())$ (where Ξ'_3, x_3 and Ξ'_4, x_4 are the corresponding permutations of Ξ_3 and Ξ_4, respectively). Noticing that $\mathcal{FV}(t_2)$ is a permutation of $\Theta, \Sigma, \Lambda, \Psi$, the term h can be transformed by $\mathsf{P_{cs}}$ and then by rule Cont to:

$$\mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1|\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi'_3|\Sigma_4,\Xi'_4}_{\Sigma_1,\Xi'}(\mathcal{C}^{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5|\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}_{\Theta,\Sigma_2,\Lambda_2,\Psi_2}(v_1)))$$

where

$$\begin{array}{rl} v_1 := & (t'\ u')\langle x_3 \backslash \mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(t_2))\rangle\langle x_4 \backslash \mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(t_2))\rangle \\ \longrightarrow_{\mathsf{App1}} & (t'\langle x_3 \backslash \mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(t_2))\rangle\ u')\langle x_4 \backslash \mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(t_2))\rangle \\ \longrightarrow_{\mathsf{App2}} & t'\langle x_3 \backslash \mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(t_2))\rangle\ u'\langle x_4 \backslash \mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(t_2))\rangle \end{array}$$

which we call v'_1. Now we rearrange the contractions:

$$\begin{array}{rl} & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1|\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi'_3|\Sigma_4,\Xi'_4}_{\Sigma_1,\Xi'}(\mathcal{C}^{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5|\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}_{\Theta,\Sigma_2,\Lambda_2,\Psi_2}(v'_1))) \\ \equiv & \mathcal{C}^{\Theta_5|\Theta_6}_{\Theta}(\mathcal{C}^{\Xi'_3|\Xi'_4}_{\Xi'}(\mathcal{C}^{\Lambda_1|\Lambda_2}_{\Lambda}(\mathcal{C}^{\Lambda_5|\Lambda_6}_{\Lambda_2}(\mathcal{C}^{\Psi_1|\Psi_2}_{\Psi}(\mathcal{C}^{\Psi_5|\Psi_6}_{\Psi_2}(v_2)))))) \\ & \text{where } v_2 := \mathcal{C}^{\Sigma_1|\Sigma_2}_{\Sigma}(\mathcal{C}^{\Sigma_3|\Sigma_4}_{\Sigma_1}(\mathcal{C}^{\Sigma_5|\Sigma_6}_{\Sigma_2}(v'_1))) \\ \equiv & \mathcal{C}^{\Theta_5|\Theta_6}_{\Theta}(\mathcal{C}^{\Xi'_3|\Xi'_4}_{\Xi'}(\mathcal{C}^{\Lambda_2|\Lambda_6}_{\Lambda}(\mathcal{C}^{\Lambda_1|\Lambda_5}_{\Lambda_2}(\mathcal{C}^{\Psi_5|\Psi_2}_{\Psi}(\mathcal{C}^{\Psi_1|\Psi_6}_{\Psi_2}(v'_2)))))) \\ & \text{where } v'_2 := \mathcal{C}^{\Sigma_1|\Sigma_2}_{\Sigma}(\mathcal{C}^{\Sigma_3|\Sigma_5}_{\Sigma_1}(\mathcal{C}^{\Sigma_4|\Sigma_6}_{\Sigma_2}(v'_1))) \\ \equiv & \mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1|\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(\mathcal{C}^{\Lambda_1,\Sigma_3|\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(\mathcal{C}^{\Psi_1,\Sigma_4|\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(v'_1))) \end{array}$$

This term can be reduced by $\mathsf{CApp2}$ and then by $\mathsf{CApp1}$ to

$$h' := \mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1|\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(p\ q)$$

where

$$\begin{aligned}p &:= \mathcal{C}^{\Lambda_1,\Sigma_3|\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(t'\langle x_3\backslash\mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}(\mathcal{A}(t_2))\rangle)\\ &= \mathcal{R}^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1}(\mathcal{C}^{\Lambda_1,\Sigma_3|\Lambda_5,\Sigma_5}_{\Lambda,\Sigma}(\mathcal{R}^{\Lambda,\Sigma}_{\Lambda_1,\Sigma_3}(\mathcal{R}^{x}_{x_3}(\mathcal{A}(t)))\langle x_3\backslash\mathcal{R}^{\Sigma,\Lambda}_{\Sigma_5,\Lambda_5}(\mathcal{A}(t_2))\rangle))\end{aligned}$$

$$\begin{aligned}q &:= \mathcal{C}^{\Psi_1,\Sigma_4|\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(u'\langle x_4\backslash\mathcal{R}^{\Theta,\Sigma,\Lambda,\Psi}_{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}(\mathcal{A}(t_2))\rangle)\\ &= \mathcal{R}^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}(\mathcal{C}^{\Psi_1,\Sigma_4|\Psi_6,\Sigma_6}_{\Psi,\Sigma}(\mathcal{R}^{\Psi,\Sigma}_{\Psi_1,\Sigma_4}(\mathcal{R}^{x}_{x_4}(\mathcal{A}(u)))\langle x_4\backslash\mathcal{R}^{\Sigma,\Psi}_{\Sigma_6,\Psi_6}(\mathcal{A}(t_2))\rangle))\end{aligned}$$

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

$$\begin{aligned}p \to^*_{\mathsf{xr}} p' &:= \mathcal{R}^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1}(\mathcal{A}(t\{x\backslash t_2\}))\\ q \to^*_{\mathsf{xr}} q' &:= \mathcal{R}^{\Theta,\Xi',\Lambda,\Psi,\Sigma}_{\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}(\mathcal{A}(u\{x\backslash t_2\}))\end{aligned}$$

So h' reduces to
$$\mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1|\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(p'\ q')$$
which is $\mathcal{A}(t\{x\backslash t_2\}\ \ u\{x\backslash t_2\}) = \mathcal{A}((t\ u)\{x\backslash t_2\})$.

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

$$\begin{array}{ll} & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1|\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi_3|\Sigma_4,\Xi_4}_{\Sigma_1,\Xi}((t'\ u')\langle x\backslash\mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(t_2))\rangle))\\ \to_{\mathsf{App1}} & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1|\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi_3|\Sigma_4,\Xi_4}_{\Sigma_1,\Xi}(t'\langle x\backslash\mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(t_2))\rangle\ u'))\\ \equiv & \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3|\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\mathcal{C}^{\Sigma_3,\Lambda_1|\Sigma_2,\Lambda_2}_{\Sigma_1,\Lambda}(t'\langle x\backslash\mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(t_2))\rangle\ u'))\\ \to_{\mathsf{CApp1}} & \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3|\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\mathcal{C}^{\Sigma_3,\Lambda_1|\Sigma_2,\Lambda_2}_{\Sigma_1,\Lambda}(t'\langle x\backslash\mathcal{R}^{\Sigma,\Lambda,\Psi}_{\Sigma_2,\Lambda_2,\Psi_2}(\mathcal{A}(t_2))\rangle)\ u')\\ = & \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3|\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\mathcal{R}^{\Sigma,\Psi,\Xi}_{\Sigma_1,\Psi_2,\Xi_3}(v)\ \mathcal{R}^{\Sigma,\Psi,\Xi}_{\Sigma_4,\Psi_1,\Xi_4}(u))\end{array}$$

where $v := \mathcal{C}^{\Sigma_3,\Lambda_1|\Sigma_2,\Lambda_2}_{\Sigma,\Lambda}(\mathcal{R}^{\Lambda,\Sigma}_{\Lambda_1,\Sigma_3}(\mathcal{A}(t))\langle x\backslash\mathcal{R}^{\Sigma,\Lambda}_{\Sigma_2,\Lambda_2}(\mathcal{A}(t_2))\rangle)$, which reduces, by induction hypothesis, to $\mathcal{A}(t\{x\backslash t_2\})$. Hence,

$$h \to^*_{\mathsf{xr}} \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3|\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\mathcal{R}^{\Sigma,\Psi,\Xi}_{\Sigma_1,\Psi_2,\Xi_3}(\mathcal{A}(t\{x\backslash t_2\}))\ \mathcal{R}^{\Sigma,\Psi,\Xi}_{\Sigma_4,\Psi_1,\Xi_4}(u))$$

which is exactly $\mathcal{A}(t\{x\backslash t_2\}\ \ u) = \mathcal{A}((t\ u)\{x\backslash t_2\})$.

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

(d) The case $x \notin \mathcal{FV}(t)$ and $x \notin \mathcal{FV}(u)$ cannot happen since we assumed $x \in \mathcal{FV}(t_1)$.

3. If $t_1 = \lambda y.v$ then by α-equivalence we can suppose $y \neq x$ and $y \notin \mathcal{FV}(t_2)$.

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

$$\begin{array}{ll} & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{R}^{\Phi}_{\Upsilon}(\lambda y.\mathcal{A}(v))\langle x\backslash\mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))\rangle)\\ = & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}((\lambda y.\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(v)))\langle x\backslash\mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))\rangle)\\ \to_{\mathsf{Abs}} & \mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\lambda y.(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(v))\langle x\backslash\mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))\rangle))\\ \to_{\mathsf{CAbs}} & \lambda y.\mathcal{C}^{\Upsilon|\Omega}_{\Phi}(\mathcal{R}^{\Phi}_{\Upsilon}(\mathcal{A}(v))\langle x\backslash\mathcal{R}^{\Phi}_{\Omega}(\mathcal{A}(t_2))\rangle)\end{array}$$

and we get the result by the induction hypothesis.

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

$$\begin{array}{ll}
& \mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\mathcal{R}_{\Upsilon}^{\Phi}(\lambda y.\mathcal{W}_y(\mathcal{A}(v)))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle) \\
= & \mathcal{C}_{\Phi}^{\Upsilon|\Omega}((\lambda y.\mathcal{W}_y(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(v))))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle) \\
\to_{\mathsf{Abs}} & \mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\lambda y.(\mathcal{W}_y(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(v))))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle) \\
\to_{\mathsf{Weak2}} & \mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\lambda y.\mathcal{W}_y(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(v))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle)) \\
\to_{\mathsf{CAbs}} & \lambda y.\mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\mathcal{W}_y(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(v))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle)) \\
\to_{\mathsf{Cross}}^{*} & \lambda y.\mathcal{W}_y(\mathcal{C}_{\Phi}^{\Upsilon|\Omega}(\mathcal{R}_{\Upsilon}^{\Phi}(\mathcal{A}(v))\langle x\backslash\mathcal{R}_{\Omega}^{\Phi}(\mathcal{A}(t_2))\rangle))
\end{array}$$

and we get the result by the induction hypothesis.