The theory of calculi with explicit substitutions revisited

Delia Kesner

The theory of calculi with explicit substitutions revisited

Delia Kesner

October 9, 2006

Abstract

Calculi with explicit substitutions are widely used in different areas of computer science such as functional and logic programming, proof-theory, theorem proving, concurrency, object-oriented languages, etc. Complex systems with explicit substitutions were developed these last 15 years in order to capture the good computational behaviour of the original system (with meta-level substitutions) they were implementing.

In this paper we first survey previous work in the domain by pointing out the motivations and challenges that guided the developement of such calculi. Then we use very simple technology to establish a general theory of explicit substitutions for the lambda-calculus which enjoys all the expected properties such as simulation of one-step beta-reduction, confluence on meta-terms, preservation of beta-strong normalisation, strong normalisation of typed terms and full composition. Also, the calculus we introduce turns out to admit a natural translation into Linear Logic's proof-nets.

Contents

1 Introduction

This paper is about explicit substitutions (ES), an intermediate formalism that - by decomposing the β rule into more atomic steps - allows a better understanding of the execution models of λ-calculus.

We first survey previous work in the domain, by pointing out the motivations that were guided the developement of such calculi as well as the main challenge behind their formulations. The goal of our work is to move back to previous works and results in the domain in order to establish a general and simple theory of explicit substitutions being able to capture all of them by using very simple technology.

Explicit substitutions

In λ-calculus, the evaluation process is modelled by β-*reduction* and the replacement of formal parameters by its corresponding arguments is modelled by *substitution*. While substitution in λ-calculus is a *meta-level* operation described outside the calculus itself, in calculi with ES it is internalised and handled by symbols and reduction rules belonging to the proper syntax of the calculus. However the two formalisms are still very close: let $s\{x/u\}$ denote the result of substituting all the *free* occurrences of x in s by u, then one defines β-reduction as

$$(\lambda x.s)\ v \longrightarrow_\beta s\{x/v\}$$

where the operation $s\{x/v\}$ can be defined modulo α-conversion [1] by induction on s as follows:

$$\begin{array}{lcll} x\{x/v\} & := & v & \\ y\{x/v\} & := & y & \text{if } x \neq y \\ (t\ u)\{x/v\} & := & (t\{x/v\}\ u\{x/v\}) & \\ (\lambda y.t)\{x/v\} & := & \lambda y.(t\{x/v\}) & \text{if } x \neq y \text{ and } y \notin \mathtt{fv}(v) \end{array}$$

Then, the simplest way to specify a λ-calculus with explicit substitution is to explicitly encode the previous definition, so that one still works modulo α-conversion, yielding the calculus known as $\lambda\mathtt{x}$ which is shown in Figure 1.

$$\begin{array}{lcll} (\lambda x.t)\ v & \longrightarrow & t[x/v] & \\ x[x/v] & \longrightarrow & v & \\ x[y/v] & \longrightarrow & x & \text{if } x \neq y \\ (t\ u)[x/v] & \longrightarrow & (t[x/v]\ u[x/v]) & \\ (\lambda x.t)[y/v] & \longrightarrow & \lambda x.(t[y/v]) & \text{if } x \neq y \text{ and } x \notin \mathtt{fv}(v) \end{array}$$

Figure 1: Reduction rules for the $\lambda\mathtt{x}$-calculus

[1] Definition of substitution modulo α-conversion avoids to explicitly deal with the variable capture case as one obtains it for free. Thus, for example $(\lambda x.y)\{y/x\} =_\alpha (\lambda z.y)\{y/x\} =_{def} \lambda z.y\{y/x\} = \lambda z.x$.

This reduction system corresponds to the minimal behaviour that can be found in most of the well-known calculi with ES appearing in the literature: substitutions are incorporated into the language and manipulated explicitly, β-reduction is implemented in two stages, first by the application of the first rule, which activates the calculus of substitutions, then by propagation of the substitution until variables are reached. More sophisticated treatment of substitutions considers also a composition operator allowing interactions between them.

1.1 Related Work

In these last years there has been a growing interest in λ-calculi with explicit substitutions. They were defined in de Bruijn notation [ACCL91a, HL89, Les94, KR95, Kes96, FKP96], or level notation [LRD95], or via combinators [GL99], or simply by named variables notation as shown above [Lin86, Lin92, Ros92, BR95a].

An abstract presentation of such calculi can be found in [Kes96, Kes00], where a (syntactic) axiomatisation is used to define and study them.

In any case, all these calculi were all introduced as a bridge between the classical λ-calculus and concrete implementations of functional programming languages such as CAML [Oca], SML [MTH90], Miranda [Tur85], Haskell [HPJP92] or proof-assistants such as Coq [Coq], PVS [PVS], HOL [HOL], LEGO [LEG], Maude [Mau] et ELAN [ELA].

Now, the implementation of the atomic substitution operation by several elementary explicit steps comes at a price. Indeed, while λ-calculus is perfectly *orthogonal* [2], calculi with ES suffer at least from the well-known diverging example

$$t[y/v][x/u[y/v]] \ {}^{*}\!\longleftarrow ((\lambda x.t)\ u)[y/v] \longrightarrow^{*} t[x/u][y/v]$$

Different solutions were adopted by the calculi in the literature in order to close this diagram. If no new rewriting rules are added to those in Figure 1, then reduction turns out to be confluent on terms but not on *metaterms* [3]. If naive rules for composition are also considered, then one recovers confluence on metaterms but paying an important price: there exist terms which are strongly normalisable in λ-calculus but not in the corresponding explicit version of the λ-calculus. This phenomenon, known as Melliès' counter-example [Mel95], shows a flaw in the design of calculi with ES in that they are supposed to implement their underlying calculus (in our case the λ-calculus) without losing its good properties. More precisely, let us call $\lambda_{\mathtt{Z}}$ a λ-calculus with ES and let consider a mapping $\mathtt{to}_{\lambda}$ from λ-syntax to $\lambda_{\mathtt{Z}}$-syntax (sometimes this mapping is just the identity). We identify the following list of properties:

(SIM) Any evaluation step in λ-calculus can be implemented by $\lambda_{\mathtt{Z}}$: If $t \longrightarrow_{\beta} t'$, then $\mathtt{to}_{\lambda}(t) \longrightarrow^{*}_{\lambda_{\mathtt{Z}}} \mathtt{to}_{\lambda}(t')$.

(CR) The refined reduction relation $\lambda_{\mathtt{Z}}$ is confluent: If $u \ {}^{*}_{\lambda_{\mathtt{Z}}}\!\longleftarrow t \longrightarrow^{*}_{\lambda_{\mathtt{Z}}} v$, then there is t' such that $u \longrightarrow^{*}_{\lambda_{\mathtt{Z}}} t' \ {}^{*}_{\lambda_{\mathtt{Z}}}\!\longleftarrow v$.

[2] Does not have critical pairs.

[3] Terms with metavariables used to represent incomplete proofs

(PSN) The reduction relation $\lambda_{\mathtt{z}}$ preserves β-strong normalisaion: If $t \in \mathcal{SN}_{\beta}$, then $\mathtt{to}_{\lambda}(t) \in \mathcal{SN}_{\lambda_{\mathtt{z}}}$.

(SN) Strong normalisation holds for $\lambda_{\mathtt{z}}$-typed terms: If t is typed, then $t \in \mathcal{SN}_{\lambda_{\mathtt{z}}}$.

(FC) Full composition can be implemented by $\lambda_{\mathtt{z}}$: $t[x/u]$ $\lambda_{\mathtt{z}}$-reduces to $t\{x/u\}$ for an appropriate (and natural) notion of substitution on $\lambda_{\mathtt{z}}$-terms.

The result of Melliès has revived the interest in ES since after his counterexample there was a clear *challenge* to find a calculus having all the good properties mentioned above.

There are several propositions that give (sometimes partial) answers to this challenge, they are summarised in Figure 2.

Calculus	CR	SN	PSN	SIM	FC
$\lambda_{v}\lambda_{s}\lambda_{t}\lambda_{u}\lambda_{\mathtt{x}}\lambda_{d}\lambda_{dn}\lambda_{e}\lambda_{f}$	No	Yes	Yes	Yes	No
$\lambda_{\sigma}\lambda_{\sigma SP}\lambda_{\sigma\Uparrow}\lambda_{se}\lambda_{\mathcal{L}}$	Yes	No	No	Yes	Yes
λ_{ζ}	Yes	Yes	Yes	No	No
λ_{ws}	Yes	Yes	Yes	Yes	No
$\lambda_{\lambda\mathtt{lxr}}$	Yes	Yes	Yes	Yes	Yes

Figure 2: Summarising previous work in the field

In other words, 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 and SN [DCKP00]. But the λ_{ws}-calculus has a complicated syntax and its named version [DCKP00] is even less readable.

The strong normalisation proof for λ_{ws} given in [DCKP00] reveals a natural semantics for composition of explicit substitutions via Linear Logic's proof-nets, suggesting that weakening (explicit erasure) and contraction (explicit duplication) can be added to the calculus without losing termination. These are the starting points of the ideas proposed by the $\lambda\mathtt{lxr}$-calculus [KL05], which is in some sense a (complex) precursor of the $\lambda\mathtt{es}$-calculus that we present in this work. Indeed, λ-terms can not be viewed directly as $\lambda\mathtt{lxr}$-terms, so that we prefer to adopt $\lambda\mathtt{x}$-syntax for $\lambda\mathtt{es}$, thus avoiding special encodings in order to explicitly incorporate weakening and contractions inside

λ-terms. Moreover, the reduction system of $\lambda\mathtt{lxr}$ is defined via 6 equations and 19 rewriting rules, thus requiring an important amount of combinatory reasoning when showing its properties.

Another calculi with safe notions of compositions appear for example in [SFM03, Sak]. The first of them lacks full composition and confluence on metaterms. The second of them specifies commutation of independent substitutions by a rewriting rule (instead of an equation), thus leading to complicated notions and proofs of its underlying normalisation properties. Here, we choose to make a minimal (just one) use of equational reasoning to axiomatise commutation of independent substitution. This will turn out to be essential to achieve the definition of a simple language being easy to understand, which can be projected into another elementary system like proof-nets, and whose properties can be proved with simple and natural proof techniques.

Last but not least, confluence on metaterms of both calculi in [KL05] and [Sak] on metaterms is only conjectured but not yet proved.

The logical meaning of explicit substitutions

Cut elimination is a logical evaluation process allowing to relate explicit substitution to a more atomic process. Indeed, the cut elimination process can be interpreted as the elimination of explicit substitutions. For example, let us consider the following sequent proof:

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

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

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

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

$$\dfrac{\Gamma \vdash v : A \qquad \Gamma, x : A \vdash x : A\ (\mathtt{proj})}{\Gamma \vdash x[x/v] : A}\ (\mathtt{subs})$$

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

$$(\mathtt{Var}) \qquad x[x/v] \longrightarrow v$$

These remarks put in evidence the fact that explicit substitution is a term notation for the cut rule, and that reduction rules for explicit substitutions behave like cut elimination rules. However, λ and $\lambda\mathtt{x}$ basic (typed) syntax are taken from a natural deduction logical system, where application annotates implication elimination and

abstraction annotates implication introduction. That means that $\lambda\mathtt{x}$ (typed) syntax is based on a logical system mixing natural deduction with sequent calculus such that the meta-level operation in the normalisation process is replaced by a more elementary concept of cut elimination.

It is worth noticing that one can either define an explicit substitution calculus interpreting cut-elimination, in such a way to have a perfect Curry-Howard correspondence between them, as is done by Hugo Herbelin in [Her94]: there terms encode proofs, types encode propositions and reduction encodes cut-elimination in intuitionistic sequent calculus. So that the ideas we present in this paper can also be adapted to sequent calculus notation. We refer the reader to [Len06] for a systematic study of cut elimination in intuitionistic sequent calculus via proof-terms.

1.2 Linear logic and proof-nets

Linear Logic decomposes the intuitionistic logical connectives, like the implication, into more atomic, resource-aware connectives, like the linear implication and the explicit erasure and duplication operators given by the exponentials which provide a more refined computational model that the one given by the λ-calculus. However, sequent presentations of Linear Logic can contain a lot of details that are uninteresting (or bureaucratic). The main idea of proof-nets is to solve this problem by providing a sort of representative of an equivalence class of proofs in the sequent calculus style that differ only by the order of application of some logical or structural rules. Cut elimination over proof-nets is then a kind of normalisation procedure over these equivalence classes. Using different translations of the λ-calculus into Proof Nets, new abstract machines have been proposed, exploiting the Geometry of Interaction [Gir89, AJ92], culminating in the recent works on optimal reduction [GAL92, Lam90].

Some calculi with explicit substitutions [DCKP03, KL05] have been already put in relation with natural extended notions of proof-nets. In particular, one defines a typed version of the calculus and shows how to translate it into Proof Nets and how to establish, using this translation, a simulation of the reduction rules for explicit substitutions via cut elimination in Proof Nets. As an immediate consequence of this simulation, one proves that a simply typed version of the calculus is strongly normalizing. An important property of the simulation is that each step in the calculus with ES is simulated by a *constant* number of steps in proof-nets: this shows that the two systems are very close, unlike what happens when simulating the λ-calculus. This gives also a powerful tool to reason about the complexity of β-reduction.

We apply this idea to the $\lambda\mathtt{es}$-calculus that we introduce in this paper so that we obtain strong normalisation for typed $\lambda\mathtt{es}$-terms via simulation of reduction in proof-nets.

1.3 Summary

We present a calculus with ES using the named variable presentation, which makes some essential properties of explicit substitutions more apparent, by abstracting out the details of renaming and updating of de Bruijn notation. The main ideas and results of the paper can be summarised by the following points:

- We define a *simple* calculus with explicit substitutions using named variables and called $\lambda\texttt{es}$. There is no use of explicit contraction and weakening.
- We show simulation of one-step β-reduction, confluence on metaterms, preservation of β-strong normalisation, strong normalisation of typed terms and implementation of full composition.
- We establish connections with untyped λ-calculus and typed λ-calculus.
- We give a natural translation into Linear Logic's proof-nets.
- We give some ideas for future work and applications.

The rest of the paper is organised as follows. Section 2 introduces syntax for $\lambda\texttt{es}$-terms as well as appropriate notions of equivalence and reduction. We show there some fundamental properties of the calculus such as *full composition* and termination of the substitution calculus alone. In Section 3 we develop a proof of confluence for metaterms. This proof uses an interpretation method based on the confluence property of a simpler calculus that we define in the same section. Preservation of β-strong normalisation is studied and proved in Section 4. The proof is based on the terminating properties of other calculi that we introduce in the same section. Relations between reduction in $\lambda\texttt{es}$ and λ-calculus are established in Section 5. The typing system for $\lambda\texttt{es}$ is presented in 6 as well as the subject reduction property. Relations between typing in $\lambda\texttt{es}$ and λ-calculus are established in Section 7. Section 8 introduces proof nets and gives the translation from typed $\lambda\texttt{es}$-terms into proof nets that is used to obtain strong normalisation of typed $\lambda\texttt{es}$. Finally, a simpler proof of strong normalisation based on the main result of Section 4 is given in Section 9.

We refer the reader to [Kes06] for detailed proofs and to [BN98] for standard notions from rewriting that we will use throughout the paper.

2 Syntax

We introduce here the basic notions concerning syntax, α-conversion, reduction and congruence.

The set of $\lambda\texttt{es}$-terms can be defined by the following grammar

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[x/t]$$

A term x is called a *variable*, $(t\ u)$ an *application*, $\lambda x.t$ an *abstraction* and $t[x/u]$ a *closure*. The syntactic object $[x/u]$, which is not a term itself, is called an *explicit substitution*. We do not write the parenthesis of applications if they are clear from the context.

The syntax can also be given as a HRS [Nip91], with types $\mathcal{V}$ and $\mathcal{T}$ for variables and (raw)terms respectively, and four function symbols to be used as constructors:

var:	$\mathcal{V} \to \mathcal{T}$	sub:	$(\mathcal{V} \to \mathcal{T}) \to (\mathcal{T} \to \mathcal{T})$
lam:	$(\mathcal{V} \to \mathcal{T}) \to \mathcal{T}$	app:	$\mathcal{T} \to (\mathcal{T} \to \mathcal{T})$

Thus, for example the $\lambda\mathtt{es}$-term $(x\ y)[x/\lambda z.z]$ is represented as the HRS-term $\mathsf{sub}(x.\mathsf{app}(\mathsf{var}(x),\mathsf{var}(y)),\mathsf{lam}(z.\mathsf{var}(z)))$. We prefer however to work with the syntax given by the grammar above which is the one usually used for calculi with ES.

A term is said to be *pure* if it has no explicit substitutions.

The terms $\lambda x.t$ and $t[x/u]$ bind x in t. Thus, the set of *free variables* of a term t, denoted $\mathtt{fv}(t)$, is defined in the usual way as follows:

$$\begin{array}{lcl}\mathtt{fv}(x) & := & \{x\}\\ \mathtt{fv}(t\ u) & := & \mathtt{fv}(t)\cup\mathtt{fv}(u)\\ \mathtt{fv}(\lambda x.t) & := & \mathtt{fv}(t)\setminus x\\ \mathtt{fv}(t[x/u]) & := & (\mathtt{fv}(t)\setminus x)\cup\mathtt{fv}(u)\end{array}$$

As a consequence, we obtain the standard notion of α-conversion on higher-order terms which allows us to use Barendregt's convention [Bar84] to assume that two different bound variables have different names, and no variable is free and bound at the same time.

Besides α-conversion we consider the following equation and set of reduction rules:

Equations :			
$t[x/u][y/v]$	$=_{\mathtt{C}}$	$t[y/v][x/u]$	if $y\notin\mathtt{fv}(u)\ \&\ x\notin\mathtt{fv}(v)$
Reduction Rules :			
$(\lambda x.t)\ u$	$\longrightarrow_{\mathtt{B}}$	$t[x/u]$	
$x[x/u]$	$\longrightarrow_{\mathtt{Var}}$	u	
$t[x/u]$	$\longrightarrow_{\mathtt{Gc}}$	t	if $x\notin\mathtt{fv}(t)$
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}_1}$	$(t[x/v]\ u[x/v])$	if $x\in\mathtt{fv}(t)\ \&\ x\in\mathtt{fv}(u)$
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}_2}$	$(t\ u[x/v])$	if $x\notin\mathtt{fv}(t)\ \&\ x\in\mathtt{fv}(u)$
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}_3}$	$(t[x/v]\ u)$	if $x\in\mathtt{fv}(t)\ \&\ x\notin\mathtt{fv}(u)$
$(\lambda y.t)[x/v]$	$\longrightarrow_{\mathtt{Lamb}}$	$\lambda y.t[x/v]$	if $y\notin\mathtt{fv}(v)\ \&\ x\neq y$
$t[x/u][y/v]$	$\longrightarrow_{\mathtt{Comp}_1}$	$t[y/v][x/u[y/v]]$	if $y\in\mathtt{fv}(u)\ \&\ y\in\mathtt{fv}(t)$
$t[x/u][y/v]$	$\longrightarrow_{\mathtt{Comp}_2}$	$t[x/u[y/v]]$	if $y\in\mathtt{fv}(u)\ \&\ y\notin\mathtt{fv}(t)$

Figure 3: Equations and reduction rules for $\lambda\mathtt{es}$

It is worth noticing that this presentation uses conditional rules, but non conditional rules could be used to specify this system when using for example proof-nets (Section 8) or director strings [KS88]. Also, rules $\{\mathtt{App}_1,\mathtt{App}_2,\mathtt{App}_3\}$ could be gathered into the single rule $(\mathtt{App})\quad (t\ u)[x/v]\longrightarrow(t[x/v]\ u[x/v])$ while rules $\{\mathtt{Comp}_1,\mathtt{Comp}_2\}$ could be replaced by the more compact version $(\mathtt{Comp})\quad t[x/u][y/v]\longrightarrow t[y/v][x/u[y/v]]$, where $y\in\mathtt{fv}(u)$. While this change seems to be sound w.r.t. the properties of the calculus [4] (c.f. Section 10), the translation to proof-nets does not work (c.f. Section 8.3). We thus prefer to work with this version of the calculus all along the paper to simplify into a one single presentation all the material presented here.

[4] While the weaker rule for composition given by $t[x/u][y/v]\longrightarrow t[x/u[y/v]]$, where $y\notin\mathtt{fv}(t)$, is well-known [BG99] to affect strong normalisation and preservation of β-strong normalisation.

The *rewriting system* containing all the previous rewriting rules except $\mathtt{B}$ is denoted by $\mathtt{s}$. We write $\mathtt{Bs}$ for $\mathtt{B} \cup \mathtt{s}$. The *equivalence relation* generated by the conversions α and $\mathtt{C}$ is denoted by $\mathtt{E_s}$. The *reduction relation* generated by the *reduction rules* $\mathtt{s}$ (resp. $\mathtt{Bs}$) *modulo the equivalence relation* $\mathtt{E_s}$ is denoted by $\longrightarrow_{\mathtt{s}} /\mathtt{E_s}$ or $\longrightarrow_{\mathtt{es}}$ (resp. $\longrightarrow_{\mathtt{Bs}} /\mathtt{E_s}$ or $\longrightarrow_{\lambda\mathtt{es}}$ (for **e**quational **s** substitution), the $\mathtt{e}$ is for for equational and the $\mathtt{s}$ for substitution. More precisely

$$\begin{array}{ll} t \longrightarrow_{\mathtt{es}} t' & \text{iff there are } s, s' \text{ s.t. } \quad t =_{\mathtt{E_s}} s \longrightarrow_{\mathtt{s}} s' =_{\mathtt{E_s}} t' \\ t \longrightarrow_{\lambda\mathtt{es}} t' & \text{iff there are } s, s' \text{ s.t. } \quad t =_{\mathtt{E_s}} s \longrightarrow_{\mathtt{Bs}} s' =_{\mathtt{E_s}} t' \end{array}$$

The equivalence relation preserves free variables and the reduction relation does not increase them. Indeed, one can easily show by induction on terms the following property.

Lemma 2.1 (Free variables do not increase) *If* $t \longrightarrow_{\lambda\mathtt{es}} t'$*, then* $\mathtt{fv}(t') \subseteq \mathtt{fv}(t)$*. More precisely,*

- *If* $t =_{\mathtt{E_s}} t'$*, then* $\mathtt{fv}(t) = \mathtt{fv}(t')$*.*
- *If* $t \longrightarrow_{\mathtt{Bs}} t'$*, then* $\mathtt{fv}(t') \subseteq \mathtt{fv}(t)$*.*

The (sub)calculus of substitutions $\mathtt{es}$, which is intended to implement (meta-level) substitution can be shown to be terminating.

Lemma 2.2 (Termination of $\mathtt{es}$) *The reduction relation* $\mathtt{es}$ *is terminating.*

Proof. For each term s we define a size and a multiplicity by structural induction.

$$\begin{array}{lcllcll} \mathtt{S}(x) & := & 1 & \mathtt{M}_x(z) & := & 1 & \\ \mathtt{S}(\lambda x.t) & := & \mathtt{S}(t) & \mathtt{M}_x(\lambda y.t) & := & \mathtt{M}_x(t) + 1 & \\ \mathtt{S}(t\ u) & := & \mathtt{S}(t) + \mathtt{S}(u) & \mathtt{M}_x(t\ u) & := & \mathtt{M}_x(t) + \mathtt{M}_x(u) + 1 & \\ \mathtt{S}(t[x/u]) & := & \mathtt{S}(t) + \mathtt{M}_x(t) \cdot \mathtt{S}(u) & \mathtt{M}_x(t[y/u]) & := & \mathtt{M}_x(t) & \text{If } x \notin \mathtt{fv}(u) \\ & & & \mathtt{M}_x(t[y/u]) & := & \mathtt{M}_x(t) + \mathtt{M}_y(t) \cdot (\mathtt{M}_x(u) + 1) & \text{If } x \in \mathtt{fv}(u) \end{array}$$

Remark that $\mathtt{M}_x(s) \geq 1$ and $\mathtt{S}(s) \geq 1$ for every term s and every variable x.

We can now show, by induction on the definition of $=_{\mathtt{E_s}}$ and $\longrightarrow_{\mathtt{s}}$, that size is compatible with α and $\mathtt{C}$ equality and each $\mathtt{s}$-reduction step strictly decreases the size:

1. If $s =_{\mathtt{E_s}} s'$, then $\mathtt{S}(s) = \mathtt{S}(s')$.
2. If $s \longrightarrow_{\mathtt{s}} s'$, then $\mathtt{S}(s) > \mathtt{S}(s')$.

We then conclude that $\mathtt{es}$-reduction is terminating on all $\lambda\mathtt{es}$-terms by application of the abstract theorem A.1 : $\mathcal{E}$ is $=_{\mathtt{E_s}}$, $\mathcal{R}_1$ is the empty relation, $\mathcal{R}_2$ is $\longrightarrow_{\mathtt{s}}$, $\mathtt{K}$ is the relation given by the function $\mathtt{S}(\_)$ and $\mathcal{S}$ is the standard well-founded order $>$ on natural numbers. ∎

We now address the property of full composition. For that, we introduce the following notion of substitution on $\lambda\texttt{es}$-terms.

Given $\lambda\texttt{es}$-terms t and u, the result of *substituting* all the *free* occurrences of x in t by u is defined by induction, and modulo α-conversion, as follows:

$$\begin{array}{lcll} x\{x/v\} & := & v & \\ y\{x/v\} & := & y & \text{if } x \neq y \\ (t\ u)\{x/v\} & := & (t\{x/v\}\ u\{x/v\}) & \\ (\lambda y.t)\{x/v\} & := & \lambda y.(t\{x/v\}) & \text{if } x \neq y \text{ and } y \notin \texttt{fv}(v) \\ t[y/u]\{x/v\} & := & t\{x/v\}[y/u\{x/v\}] & \text{if } x \neq y \text{ and } y \notin \texttt{fv}(v) \end{array}$$

It is easy to show by induction on $\lambda\texttt{es}$-terms that $t\{x/u\} = t$ if $x \notin \texttt{fv}(t)$.

Lemma 2.3 (Full Composition) *Let t and u be $\lambda\texttt{es}$-terms. Then $t[x/u] \longrightarrow^*_{\lambda\texttt{es}} t\{x/u\}$.*

Proof. By induction on t. ∎

3 Confluence on metaterms

Metaterms are terms containing *metavariables* which are usually used to denote *incomplete* programs and/or proofs in higher-order unification [Hue76]. Each metavariable should come with a minimal amount of information in order to guarantee that some basic operations such as instantiation (replacement of metavariables by metaterms) is sound. Thus, we now consider a countable set of *raw* metavariables $X, Y, \ldots$ that we decorate them with sets of variables $\Gamma, \Delta, \ldots$, thus yielding *decorated* metavariables denoted by X_Γ, Y_Δ, etc.

We now extend the primitive grammar for $\lambda\texttt{es}$-terms to obtain the $\lambda\texttt{es}$-metaterms:

$$t ::= x \mid X_\Delta \mid (t\ t) \mid \lambda x.t \mid t[x/t]$$

We add to the definition of free variables in Section 2 the case $\texttt{fv}(X_\Delta) = \Delta$. Even if this new definition is using to completely specify the free variables of a metaterm, which may sound contradictory with the concept of metaterm, it is worth noticing that the partial specification of the set of (free) variables of an incomplete proof says nothing about the structure of the incomplete proof itself as this structural information remains still unknown. The minimal information inside metavariables given by decoration of set of variables guarantees that different occurrences of the same metavariable inside a metaterm are never instantiated by different metaterms. Indeed, given the (raw) metaterm $t = \lambda y.y\ X\ (\lambda z.X)$, the instantiation of the (raw) metavariable X by a term containing a free occurrence of z would be unsound (see [Muñ97, DHK00, FdK] for details).

From now on, we use $\widehat{y}$ to denote, indistinctly, a variable y or a metavariable Y_Δ.

Remark that decorated metavariables X_Δ in $\lambda\texttt{es}$ can be viewed as HRS-terms of the form $X(x_1, \ldots, x_n)$ where $\Delta = \{x_1, \ldots, x_n\}$ and X is just a simply a variable that can only be instantiated by a closed strict (meta)term. Thus for example $Y_{x,z}$ can be instantiated by $x[y/z]$ but not by x. As a consequence, all the standard concepts on HRS-terms, such as α-conversion, reduction, etc, extend to our notion of

$\lambda\mathtt{es}$-metaterms. In particular, α-conversion is perfectly well-defined on metaterms by extending the renaming of bound variables to the decoration sets. Thus for example $\lambda x.Y_x =_\alpha \lambda z.Y_z$.

Towards confluence by composition of substitutions The idea behind calculi with explicit substitutions having composition is to implement what is known in λ-calculus as the *substitution lemma*: for all λ-terms t, u, v and variables x, y such that $x \neq y$ and $x \notin \mathtt{fv}(v)$ we have

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

It is well-known that confluence on metaterms fails for calculi with ES without composition (as for example $\lambda\mathtt{x}$) as the following example shows

$$s = t[y/v][x/u[y/v]] \ {}^*\!\longleftarrow ((\lambda x.t)\ u)[y/v] \longrightarrow^* t[x/u][y/v] = s'$$

Indeed, while this diagram can be closed for terms *without metavariables* [BR95a], there is no way to find a common reduct between s and s' whenever t is or contains metavariables and no reduction rule is allowed to mimic composition. Remark that this is true not only for raw but also for decorated metavariables.

It is evident that the reduction system $\lambda\mathtt{es}$ cover all the cases of the substitution lemma: indeed,

If $y \in \mathtt{fv}(u)$ and $y \in \mathtt{fv}(t)$, then

$$t[x/u][y/v] \longrightarrow_{\mathtt{Comp}_1} t[y/v][x/u[y/v]]$$

If $y \in \mathtt{fv}(u)$ and $y \notin \mathtt{fv}(t)$, then

$$t[x/u][y/v] \longrightarrow_{\mathtt{Comp}_2} t[x/u[y/v]] \ {}_{\mathtt{Gc}}\!\longleftarrow t[y/v][x/u[y/v]]$$

If $y \notin \mathtt{fv}(u)$ and $y \in \mathtt{fv}(t)$, then

$$t[x/u][y/v] =_{\mathtt{Comp}} t[y/v][x/u] \ {}_{\mathtt{Gc}}\!\longleftarrow t[y/v][x/u[y/v]]$$

If $y \notin \mathtt{fv}(u)$ and $y \notin \mathtt{fv}(t)$, then

$$t[x/u][y/v] \longrightarrow_{\mathtt{Gc}} t[x/u] \ {}^*_{\mathtt{Gc}}\!\longleftarrow t[y/v][x/u[y/v]]$$

Proof techniques to show confluence While most of the calculi with explicit substitutions in the literature are only specified by rewriting rules, $\lambda\mathtt{es}$-reduction is defined by a notion of reduction modulo an equivalence relation. We then need to prove confluence of a *non-terminating reduction relation modulo*, for which the published techniques [Hue80, Ter03, Ohl98, JK86] known by the author fail. More precisely, the *untyped* $\lambda\mathtt{es}$-calculus is trivially non-terminating (as it is able to simulate β-reduction), so these techniques cannot be applied to our case since they require the reduction relation to be terminating.

The solution we adopt in this paper consists in using a power version of the *interpretation technique* [Har87]. Thus, we infer confluence of $\lambda\mathtt{es}$ from confluence of $\lambda\mathtt{nss}$, a calculus with *flattened or simultaneous substitutions* whose reduction process does not make use of any equivalence relation.

3.1 A calculus with simultaneous substitutions

We consider here a *total order* on the set of variables $\mathcal{X}$. We then define ss-metaterms as metaterms with n-ary substitutions used to denote **s***imultaneous* **s***ubstitutions*. The grammar can be given by:

$$t ::= x \mid X_\Delta \mid (t\ t) \mid \lambda x.t \mid t[x_{k_1}/t, \ldots, x_{k_n}/t]$$

where substitutions $[x_{k_1}/u_{k_1} \ldots, x_{k_n}/u_{k_n}]$ are non-empty (so that $n \geq 1$) and and $x_{k_1}, \ldots, x_{k_n}$ are all *distinct* variables.

Remark that no order exist in the general syntax between the distinct variables of a simultaneous substitution.

We use letters I, J, K to denote non-empty lists of indexes for variables and $I@J$ to denote concatenation of the lists I and J. If I is the list $k_1 \ldots k_n$, then we write $[x_i/u_i]_I$ for the list $[x_{k_1}/u_{k_1}, \ldots, x_{k_n}/u_{k_n}]$. We might also use the notation $[\mathtt{lst}]$ for any of such (non-empty) lists and $[\mathtt{cs}[x/t]_i]_I$ for a simultaneous substitution of I elements containing x/t at position $i \in I$. Given $[x_i/u_i]_I$, we use the notation $[x_i/u_i]_{I+}$ to denote the substitution where an element has been added at the end of the list $x_{k_1}/u_1, \ldots, x_{k_n}/u_n$ and $[x_i/u_i]_{+I}$ to denote the substitution where an element has been added at the beginning of the list.

If $j \in I$ and $|I| \geq 2$, we write $[x_i/u_i]_{I\setminus j}$ for the list $[x_{k_1}/u_{k_1}, \ldots, x_{k_n}/u_{k_n}]$ whose element x_j/u_j has been erased. Thus for example $x[x_2/z, x_3/w]$ can be written as $x[x_i/u_i]_{[2,3]}$ with $k_1 = 2$, $k_2 = 3$, $u_2 = z$ and $u_3 = w$ and $x[x_i/u_i]_{[2,1]\setminus 2}$ denotes the term $x[x_3/w]$.

For any permutation $\pi(I)$, the notation $[x_i/u_i]_{\pi(I)}$ denotes the (permutated) list $[x_{\pi(k_1)}/u_{\pi(k_1)}, \ldots, x_{\pi(k_n)}/u_{\pi(k_n)}]$. Thus for example, if $I = k_1 \ldots k_n$ and $\mathtt{sort}(I) = j_1 \ldots j_n$, $[x_i/u_i]_{\mathtt{sort}(I)}$ means $[x_{j_1}/u_{j_1}, \ldots, x_{j_n}/u_{j_n}]$.

Definition 3.1 (Free and bound variables) *Free and bound variables of* ss*-metaterms are defined by induction as follows:*

$$\begin{array}{lcl}
\mathtt{fv}(x) & := & \{x\} \\
\mathtt{fv}(X_\Delta) & := & \Delta \\
\mathtt{fv}(t\ u) & := & \mathtt{fv}(u) \cup \mathtt{fv}(u) \\
\mathtt{fv}(\lambda x.t) & := & \mathtt{fv}(t) \setminus \{x\} \\
\mathtt{fv}(t[x_{k_1}/u_{k_1}, \ldots, x_{k_n}/u_{k_n}]) & := & \mathtt{fv}(t) \setminus \{x_{k_1}, \ldots, x_{k_n}\} \cup \mathtt{fv}(u_{k_1}) \ldots \cup \mathtt{fv}(u_{k_n})
\end{array}$$

$$\begin{array}{lcl}
\mathtt{bv}(x) & := & \emptyset \\
\mathtt{bv}(X_\Delta) & := & \emptyset \\
\mathtt{bv}(t\ u) & := & \mathtt{bv}(u) \cup \mathtt{bv}(u) \\
\mathtt{bv}(\lambda x.t) & := & \mathtt{bv}(t) \cup \{x\} \\
\mathtt{bv}(t[x_{k_1}/u_{k_1}, \ldots, x_{k_n}/u_{k_n}]) & := & \mathtt{bv}(t) \cup \{x_{k_1}, \ldots, x_{k_n}\} \cup \mathtt{bv}(u_{k_1}) \ldots \cup \mathtt{bv}(u_{k_n})
\end{array}$$

As before, we work modulo alpha conversion so we assume all bound variables are distinct and no variable is bound and free at the same time. As a consequence, for any term of the form $t[x_{k_1}/u_{k_1}, \ldots, x_{k_n}/u_{k_n}]$ we have $x_{k_i} \notin \mathtt{fv}(u_{k_j})$ for all $1 \leq i, j \leq n$.

$(t\ u)[\mathtt{lst}]$	$\longrightarrow_{\mathtt{fl}_1}$	$t[\mathtt{lst}]\ u[\mathtt{lst}]$	
$(\lambda x.t)[\mathtt{lst}]$	$\longrightarrow_{\mathtt{fl}_2}$	$\lambda x.t[\mathtt{lst}]$	
$t[x_i/u_i]_I[y_j/v_j]_J$	$\longrightarrow_{\mathtt{fl}_3}$	$t[x_i/u_i[y_j/v_j]_J, y_j/v_j]_{I@J}$	
$t[x_i/u_i]_I$	$\longrightarrow_{\mathtt{fl}_4}$	$t[x_i/u_i]_{\mathtt{sort}(I)}$	if I is not sorted

Figure 4: Reduction rules for $\mathcal{F}$

The following reduction system $\mathcal{F}$ is used to transform successive depending unary substitutions into one single (flattened) simultaneous substitution.

Note that by α-conversion there is no capture of variable in the rules $\mathtt{fl}_2$ and $\mathtt{fl}_4$.

As an example we have

$$\begin{array}{ll}
(x[x_4/x_3, x_2/z]\ y)[x_3/w] & \longrightarrow_{\mathtt{fl}_1} \\
(x[x_4/x_3, x_2/z][x_3/w]\ y[x_3/w]) & \longrightarrow_{\mathtt{fl}_3} \\
(x[x_4/x_3[x_3/w], x_2/z[x_3/w], x_3/w]\ y[x_3/w]) & \longrightarrow_{\mathtt{fl}_4} \\
(x[x_2/z[x_3/w], x_3/w, x_4/x_3[x_3/w]]\ y[x_3/w]) &
\end{array}$$

The system $\mathcal{F}$ can be considered as a functional specification thanks to the following property.

Lemma 3.1 *The system $\mathcal{F}$ is confluent and terminating on* ss*-metaterms.*

Proof. Confluence can be shown using the development closed confluence technique in [Ter03]. Termination can be shown using for example a semantic (for the sorting) Lexicographic Path Ordering [Ter03]. ∎

From now on, we denote by $\mathcal{F}(t)$ the $\mathcal{F}$-normal form of t.

Observe that $t \longrightarrow_{\mathcal{F}} t'$ implies $\mathtt{fv}(t) = \mathtt{fv}(t')$ so that $\mathtt{fv}(\mathcal{F}(t)) = \mathtt{fv}(t)$.

The following property will be useful in the rest of this section, it can be shown by induction on ss-metaterms.

Lemma 3.2 ($\mathcal{F}$-normal forms) *The set $\mathtt{nf}(\mathcal{F})$ of* ss*-metaterms that are in $\mathcal{F}$-normal form can be characterised by the following inductive definition.*

- *If $u_i \in \mathtt{nf}(\mathcal{F})$ for all $i \in I$ and $\widehat{y}$ is a variable or a metavariable, then $\widehat{y}[x_i/u_i]_I \in \mathtt{nf}(\mathcal{F})$.*
- *If $u \in \mathtt{nf}(\mathcal{F})$, then $\lambda x.u \in \mathtt{nf}(\mathcal{F})$*
- *If $u, v \in \mathtt{nf}(\mathcal{F})$, then $(u\ v) \in \mathtt{nf}(\mathcal{F})$*

3.2 A calculus with normal simultaneous substitutions

The λnss-metaterms are defined as the subset of the ss-metaterms that are in $\mathcal{F}$-normal form. The λnss-calculus is defined by the following set of reduction rules on λnss-metaterms.

$(\lambda x.t)\ u$	$\longrightarrow_{\mathtt{n}_1}$	$\mathcal{F}(t[x/u])$	
$x_j[x_i/u_i]_I$	$\longrightarrow_{\mathtt{n}_2}$	u_j	$j \in I$
$t[x_i/u_i]_I$	$\longrightarrow_{\mathtt{n}_3}$	$t[x_i/u_i]_{I\setminus j}$	$j \in I\ \&\ x_j \notin \mathtt{fv}(t)$
$t[x/u]$	$\longrightarrow_{\mathtt{n}_4}$	t	$x \notin \mathtt{fv}(t)$

Figure 5: Reduction rules for the $\lambda\mathtt{nss}$-calculus

Note that the $\mathtt{n}_4$ is a particular case of $\mathtt{n}_3$, but we have to specify it separately because we choose to avoid the use of empty substitutions.

The $\lambda\mathtt{nss}$-reduction relation is defined by induction as follows.

- If $t \longrightarrow_{\mathtt{n}_1,\mathtt{n}_2,\mathtt{n}_3,\mathtt{n}_4} t'$, then $t \longrightarrow_{\lambda\mathtt{nss}} t'$.
- If $t \longrightarrow_{\lambda\mathtt{nss}} t'$, then $\lambda x.t \longrightarrow_{\lambda\mathtt{nss}} \lambda x.t'$.
- If $t \longrightarrow_{\lambda\mathtt{nss}} t'$, then $(t\ u) \longrightarrow_{\lambda\mathtt{nss}} (t'\ u)$ and $(u\ t) \longrightarrow_{\lambda\mathtt{nss}} (u\ t')$.
- If $u \longrightarrow_{\lambda\mathtt{nss}} u'$ and $j \in I$, then $y[\mathtt{cs}[x/u]_j]_I \longrightarrow_{\lambda\mathtt{nss}} y[\mathtt{cs}[x/u']_j]_I$ and $y_\Delta[\mathtt{cs}[x/u]_j]_I \longrightarrow_{\lambda\mathtt{nss}} y_\Delta[\mathtt{cs}[x/u']_j]_I$.

As expected, the reduction system is well-defined in the sense that $t \in \mathtt{nf}(\mathcal{F})$ and $t \longrightarrow_{\lambda\mathtt{nss}} t'$ implies $t' \in \mathtt{nf}(\mathcal{F})$.

Lemma 3.3 *$\mathcal{F}$-normal forms are stable by $\lambda\mathtt{nss}$.*

Here is an example of $\lambda\mathtt{nss}$-reduction, where we assume $y < x$.

$$
\begin{array}{ll}
(\lambda x.x\ ((\lambda y.y)\ w))\ z & \longrightarrow_{\mathtt{n}_1} \\
x[x/z]\ ((\lambda y.y[x/z])\ w[x/z]) & \longrightarrow_{\mathtt{n}_1} \\
x[x/z]\ y[y/w[x/z], x/z[y/w[x/z]]] & \longrightarrow_{\mathtt{n}_2} \\
x\ y[y/w[x/z], x/z[y/w[x/z]]] & \longrightarrow_{\mathtt{n}_4} \\
x\ y[y/w[x/z], x/z] & \longrightarrow_{\mathtt{n}_3} \\
x\ y[y/w[x/z]] & \longrightarrow_{\mathtt{n}_2} \\
x\ w[x/z] & \longrightarrow_{\mathtt{n}_4} \\
x\ w &
\end{array}
$$

As expected, the $\lambda\mathtt{nss}$-calculus enjoys confluence

Theorem 3.4 ($\lambda\mathtt{nss}$ is confluent) *The relation $\lambda\mathtt{nss}$ is confluent on metaterms.*

Proof. Confluence can be shown using the development closed confluence theorem in [Ter03]. ∎

3.3 Relating λes and λnss

We now establish a correspondence between λes and λnss-reduction which will be used in the interpretation proof of confluence for λes.

We first need the following lemma.

Lemma 3.5 *Let v and u_i ($i \in I$) be* ss*-terms.*

1. *If $j \in I$, where $|I| \geq 2$ and $x_j \notin \mathtt{fv}(v)$, then $\mathcal{F}(v[x_i/u_i]_I) \longrightarrow^+_{\lambda\mathtt{nss}} \mathcal{F}(v[x_i/u_i]_{I\setminus j})$.*
2. *If $x \notin \mathtt{fv}(v)$, then $\mathcal{F}(v[x/u]) \longrightarrow^+_{\lambda\mathtt{nss}} \mathcal{F}(v)$.*

Proof. We can reason by induction on v.

▪

The λnss-reduction relation is stable by closure followed by flattening, that is,

Lemma 3.6 *Let v be a* ss*-terms and t_1, t_2 be $\mathcal{F}$-normal forms. If $t_1 \longrightarrow_{\lambda\mathtt{nss}} t_2$, then*

1. $\mathcal{F}(t_1) \longrightarrow^+_{\lambda\mathtt{nss}} \mathcal{F}(t_2)$
2. $\mathcal{F}(t_1[x/v]) \longrightarrow^+_{\lambda\mathtt{nss}} \mathcal{F}(t_2[x/v])$
3. $\mathcal{F}(v[\mathtt{cs}[\![x/t_1]\!]_i]_I) \longrightarrow^+_{\lambda\mathtt{nss}} \mathcal{F}(v[\mathtt{cs}[\![x/t_2]\!]_i]_I)$.

Proof. We can show the first and second properties by induction on λnss and the third one by induction on v.

▪

We are now ready to simulate λes-reduction into the system λnss via the flattening function $\mathcal{F}$:

Theorem 3.7 *If $t \longrightarrow_{\lambda\mathtt{es}} t'$, then $\mathcal{F}(t) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(t')$.*

Proof. We proceed by induction. If the reduction is internal, and t is an application or an abstraction, then the proof is straightforward. If $t = t_1[x/v]$ is a closure and $t' = t_2[x/v]$, then $\mathcal{F}(t_1) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(t_2)$ by i.h. and $\mathcal{F}(t) = \mathcal{F}(\mathcal{F}(t_1)[x/v]) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(\mathcal{F}(t_2)[x/v])$ holds by Lemma 3.6:2. If $t = v[x/t_1]$ is a closure and $t' = v[x/t_2]$, then $\mathcal{F}(t_1) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(t_2)$ by i.h. and $\mathcal{F}(t) = \mathcal{F}(v[x/\mathcal{F}(t_1)]) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(v[x/\mathcal{F}(t_2)])$ holds by Lemma 3.6:3. If the reduction is external we have to inspect all the possible cases.

▪

We can then conclude

Corollary 3.8 *If $t \longrightarrow^*_{\lambda\mathtt{es}} t'$, then $\mathcal{F}(t) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(t')$.*

3.4 Relating $\lambda\mathtt{nss}$ and $\lambda\mathtt{es}$

We have projected $\lambda\mathtt{es}$-reductions steps into $\lambda\mathtt{nss}$-reduction steps but we also need to prove that the projection in the other way around is possible too. This will be the second important ingredient of the interpretation proof of confluence that we present at the end of this section.

In order to translate $\lambda\mathtt{nss}$ into $\lambda\mathtt{es}$ we define the following sequentialisation function.

$$\begin{array}{lcll}
\mathtt{seq}(x) & := & x & \\
\mathtt{seq}(t\ u) & := & \mathtt{seq}(t)\ \mathtt{seq}(u) & \\
\mathtt{seq}(\lambda x.t) & := & \lambda x.\mathtt{seq}(t) & \\
\mathtt{seq}(t[x_i/u_i]_I) & := & \mathtt{seq}(t) & \text{if every } x_i \notin \mathtt{fv}(\mathtt{seq}(t)) \\
\mathtt{seq}(t[x_i/u_i]_I) & := & \mathtt{seq}(t)[x_i/\mathtt{seq}(u_i)]_K &
\end{array}$$

where K is the biggest non empty sublist of I such that for all $k \in K$ the variable x_k is free in $\mathtt{seq}(t)$.

We remark that $\mathtt{fv}(\mathtt{seq}(t)) \subseteq \mathtt{fv}(t)$.

As expected, the system $\mathtt{seq}$ can be used to project $\mathcal{F}$-reduction (Theorem 3.9) and $\lambda\mathtt{nss}$-reduction (Theorem 3.10) into $\lambda\mathtt{es}$-reduction.

Theorem 3.9 *If s and s' are $\mathtt{ss}$-terms such that $s \longrightarrow_{\mathcal{F}} s'$, then $\mathtt{seq}(s) \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(s')$.*

Proof. By induction on the reduction $\mathcal{F}$. If the reduction is internal the property is straightforward. Otherwise we have to inspect all the possible cases.

▪

Theorem 3.10 *If $s \longrightarrow_{\lambda\mathtt{nss}} s'$, then $\mathtt{seq}(s) \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(s')$*

Proof. By induction on $\longrightarrow_{\lambda\mathtt{nss}}$. The cases where the reduction is internal are straightforward so we have to inspect the cases of external reductions.

▪

We can now conclude this section with one of the main results of the paper.

Corollary 3.11 *The system $\lambda\mathtt{es}$ is confluent on metaterms.*

Proof. Let $t \equiv t'$, $t \longrightarrow^*_{\lambda\mathtt{es}} t_1$ and $t' \longrightarrow^*_{\lambda\mathtt{es}} t_2$. By Theorem 3.7 we have $\mathcal{F}(t) = \mathcal{F}(t')$ and $\mathcal{F}(t) \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(t_1)$ and $\mathcal{F}(t') \longrightarrow^*_{\lambda\mathtt{nss}} \mathcal{F}(t_2)$. Theorem 3.4 gives confluence of $\lambda\mathtt{nss}$ on $\mathcal{F}$-normal forms so that there is an $\mathcal{F}$-normal form t_3 such that $\mathcal{F}(t_1) \longrightarrow^*_{\lambda\mathtt{nss}} t_3$ and $\mathcal{F}(t_2) \longrightarrow^*_{\lambda\mathtt{nss}} t_3$. Now, $t_1 \longrightarrow^*_{\mathcal{F}} \mathcal{F}(t_1)$ and $t_2 \longrightarrow^*_{\mathcal{F}} \mathcal{F}(t_2)$ imply $\mathtt{seq}(t_1) \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(\mathcal{F}(t_1))$ and $\mathtt{seq}(t_2) \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(\mathcal{F}(t_2))$ by Theorem 3.9. But $\mathtt{seq}(t_1) = \mathtt{Gc}(t_1)$ and $\mathtt{seq}(t_2) = \mathtt{Gc}(t_2)$ so that $t_1 \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(t_1)$ and $t_2 \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(t_2)$. Theorem 3.10 allows us to conclude $\mathtt{seq}(\mathcal{F}(t_1)) \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(t_3)$ and $\mathtt{seq}(\mathcal{F}(t_2)) \longrightarrow^*_{\lambda\mathtt{es}} \mathtt{seq}(t_3)$ which closes the diagram.

▪

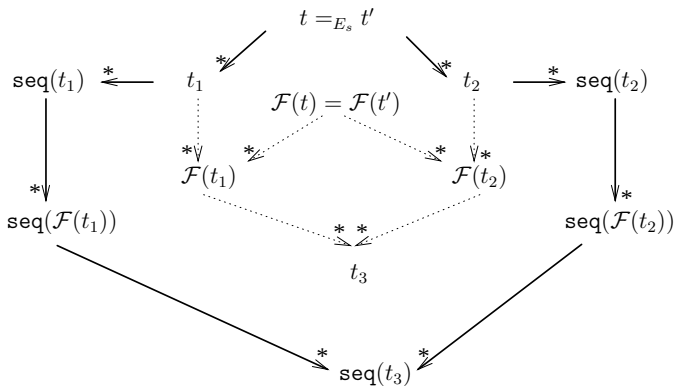

Figure 6: Confluence proof for $\lambda\mathtt{es}$ on metaterms

4 Preservation of β-strong normalisation

Preservation of β-strong normalisation (PSN) in calculi with explicit substitutions received a lot of attention (see for example [ACCL91b, BBLRD96, BR95a, KR95]), starting from an unexpected result 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. This is for example the case of $\lambda\sigma$ [ACCL91b] or $\lambda\sigma_{\Uparrow}$ [HL89].

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. 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 [GL99]. 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. Another solution, called $\lambda\mathtt{lxr}$, has been introduced latter by Kesner and Lengrand [KL05], the idea is the complete control of resources, so

that not only for weakening, but also for *contraction*. Anyway, both calculi can be translated to Linear Logic's proof-nets [DCKP03, KL05], underlying in this way the key points where composition of substitutions must be controlled. The calculus λ_{ws} as well as $\lambda\mathtt{lxr}$ introduces new syntax to handle composition. The claim of this paper is that explicit resources as weakening and contraction are not necessary to define composition correctly. Indeed, while $\lambda\mathtt{lxr}$-reduction is defined via 6 equations and 19 rewriting rules, $\lambda\mathtt{es}$ only uses an equation for commutativity of substitutions and 9 natural rewriting rules.

Preservation of β-strong normalisation is quite difficult to prove in calculi with composition (see for example [Blo97, DG01, ABR00, KL05, KOvO01]). This is mainly because the so-called *decent* terms are not stable by reduction : a term t is said to be *decent* in the calculus Z if every subterm v appearing as body of some substitution (i.e. appearing in some subterm $u[x/v]$ of t) is Z-strongly normalising. As an example, the term $x[x/(y\ y)][y/\lambda w.(w\ w)]$ is decent in $\lambda\mathtt{es}$ since $(y\ y)$ and $\lambda w.(w\ w)$ are $\lambda\mathtt{es}$-strongly normalising, but its $\mathtt{Comp}_2$-reduct $x[x/(y\ y)[y/\lambda w.(w\ w)]$ is not since $(y\ y)[y/\lambda w.(w\ w)]$ is not $\lambda\mathtt{es}$-strongly normalising.

In this paper we prove that $\lambda\mathtt{es}$ preserves β-strong normalisation by using a proof technique based on simulation. The following steps will be developed

1. We define a new calculus $\lambda\mathtt{esw}$ (section 4.1).
2. We define a translation $\mathtt{K}(\_)$ from $\lambda\mathtt{es}$-terms to $\lambda\mathtt{rxw}$ such that
   (a) $t \in \mathcal{SN}_\beta$ implies $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{rxw}}$ (Corollary 4.15)).
   (b) $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$ implies $t \in \mathcal{SN}_{\lambda\mathtt{es}}$ (Corollary 4.6).

4.1 The $\lambda\mathtt{esw}$-calculus

We introduce here the $\lambda\mathtt{esw}$-calculus, an intermediate language between $\lambda\mathtt{es}$ and $\lambda\mathtt{lxr}$ [KL05], which will be used as technical tool to prove PSN.

The grammar of $\lambda\mathtt{esw}$-terms is given as follows:

$$t ::= x \mid \lambda x.t \mid (t\ t) \mid t[x/t] \mid \mathcal{W}_x(t)$$

We will only consider here *strict* terms: every subterm $\lambda x.t$ and $t[x/u]$ is such that $x \in \mathtt{fv}(t)$ and every subterm $\mathcal{W}_x(t)$ is such that $x \notin \mathtt{fv}(t)$. We use the abbreviation $\mathcal{W}_\Gamma(t)$ for $\mathcal{W}_{x_1}(\ldots\mathcal{W}_{x_n}(t))$ whenever $\Gamma = \{x_1, \ldots, x_n\}$. In the particular case Γ is the empty set the notation $\mathcal{W}_\emptyset(t) = t$.

Besides α-conversion we consider the equations and and reduction rules in Figure 7.

The rewriting system containing all the previous rewriting rules except $\mathtt{B}$ is denoted by $\mathtt{sw}$. We write $\mathtt{Bsw}$ for $\mathtt{B}\cup\mathtt{sw}$. The equivalence relation generated by all the equations is denoted by $\mathtt{E}_{\mathtt{sw}}$. The relation generated by the reduction rules $\mathtt{sw}$ (resp. $\mathtt{Bsw}$) modulo the equivalence relation $\mathtt{E}_{\mathtt{sw}}$ is denoted by $\longrightarrow_{\mathtt{sw}}/\mathtt{E}_{\mathtt{sw}}$ or $\longrightarrow_{\mathtt{esw}}$ (resp. $\longrightarrow_{\mathtt{Bsw}}/\mathtt{E}_{\mathtt{sw}}$ or $\longrightarrow_{\lambda\mathtt{esw}}$). More precisely

$$\begin{array}{ll} t \longrightarrow_{\mathtt{esw}} t' & \text{iff there are } s, s' \text{ s.t. } \ t =_{\mathtt{E}_{\mathtt{sw}}} s \longrightarrow_{\mathtt{sw}} s' =_{\mathtt{E}_{\mathtt{sw}}} t' \\ t \longrightarrow_{\lambda\mathtt{esw}} t' & \text{iff there are } s, s' \text{ s.t. } \ t =_{\mathtt{E}_{\mathtt{sw}}} s \longrightarrow_{\mathtt{Bsw}} s' =_{\mathtt{E}_{\mathtt{sw}}} t' \end{array}$$

Equations :			
$t[x/u][y/v]$	$=_{\mathtt{C}}$	$t[y/v][x/u]$	if $y \notin \mathtt{fv}(u)$ & $x \notin \mathtt{fv}(v)$
$\mathcal{W}_x(\mathcal{W}_y(t))$	$=_{\mathtt{WC}}$	$\mathcal{W}_y(\mathcal{W}_x(t))$	
$\mathcal{W}_y(t)[x/u]$	$=_{\mathtt{Weak1}}$	$\mathcal{W}_y(t[x/u])$	if $x \neq y$ & $y \notin \mathtt{fv}(u)$
$\mathcal{W}_y(\lambda x.t)$	$=_{\mathtt{WAbs}}$	$\lambda x.\mathcal{W}_y(t)$	if $x \neq y$
Reduction Rules :			
$(\lambda x.t)\ u$	$\longrightarrow_{\mathtt{B}}$	$t[x/u]$	
$x[x/u]$	$\longrightarrow_{\mathtt{Var}}$	u	
$\mathcal{W}_x(t)[x/u]$	$\longrightarrow_{\mathtt{Gc}}$	$\mathcal{W}_{\mathtt{fv}(u)\setminus\mathtt{fv}(t)}(t)$	
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}_1}$	$(t[x/v]\ u[x/v])$	if $x \in \mathtt{fv}(t)$ & $x \in \mathtt{fv}(u)$
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}_2}$	$(t\ u[x/v])$	if $x \notin \mathtt{fv}(t)$ & $x \in \mathtt{fv}(u)$
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}_3}$	$(t[x/v]\ u)$	if $x \in \mathtt{fv}(t)$ & $x \notin \mathtt{fv}(u)$
$(\lambda y.t)[x/v]$	$\longrightarrow_{\mathtt{Lamb}}$	$\lambda y.t[x/v]$	if $y \notin \mathtt{fv}(v)$& $x \neq y$
$t[x/u][y/v]$	$\longrightarrow_{\mathtt{Comp}_1}$	$t[y/v][x/u[y/v]]$	if $y \in \mathtt{fv}(u)$ & $y \in \mathtt{fv}(t)$
$t[x/u][y/v]$	$\longrightarrow_{\mathtt{Comp}_2}$	$t[x/u[y/v]]$	if $y \in \mathtt{fv}(u)$ & $y \notin \mathtt{fv}(t)$
$(\mathcal{W}_y(t)\ u)$	$\longrightarrow_{\mathtt{WPush}}$	$(t\ u)$	if $y \in \mathtt{fv}(u)$
$(\mathcal{W}_y(t)\ u)$	$\longrightarrow_{\mathtt{WPush}}$	$(\mathcal{W}_y(t\ u))$	if $y \notin \mathtt{fv}(u)$
$(t\ \mathcal{W}_y(u))$	$\longrightarrow_{\mathtt{WPush}}$	$(t\ u)$	if $y \in \mathtt{fv}(t)$
$(t\ \mathcal{W}_y(u))$	$\longrightarrow_{\mathtt{WPush}}$	$(\mathcal{W}_y(t\ u))$	if $y \notin \mathtt{fv}(t)$
$\mathcal{W}_y(t)[x/u]$	$\longrightarrow_{\mathtt{WPush}}$	$t[x/u]$	if $y \in \mathtt{fv}(u)$
$t[x/\mathcal{W}_y(u)]$	$\longrightarrow_{\mathtt{WPush}}$	$\mathcal{W}_y(t[x/u])$	if $y \notin \mathtt{fv}(t)$
$t[x/\mathcal{W}_y(u)]$	$\longrightarrow_{\mathtt{WPush}}$	$t[x/u]$	if $y \in \mathtt{fv}(t)$

Figure 7: Equations and Reduction rules for $\lambda\mathtt{esw}$

The following lemma can be proved by induction on terms.

The following property can be shown by induction on terms.

Lemma 4.1 *The λesw-reduction relation preserves strict terms.*

From now on, we only work with strict terms.

We proceed now to show that esw is a terminating system. We will do this in two steps: first we show that $\longrightarrow_{\mathtt{esw}}$ minus $\longrightarrow_{\mathtt{WPush}}$ is terminating (Lemma 4.2), then we show that $\longrightarrow_{\mathtt{WPush}} / =_{\mathtt{E_{sw}}}$ is terminating (Lemma 4.3). All this allows us to conclude (Corollary 4.4) that the whole system $\longrightarrow_{\mathtt{esw}}$ is terminating.

We will need the following measure for terms.

Definition 4.1 *For each λesw-term s we define a size and a multiplicity by structural induction.*

$$
\begin{array}{lcllcll}
\mathtt{S}(x) & := & 1 & \mathtt{M}_x(z) & := & 1 & \\
\mathtt{S}(\mathcal{W}_x(t)) & := & \mathtt{S}(t) & \mathtt{M}_x(\mathcal{W}_y(t)) & := & \mathtt{M}_x(t) & \\
 & & & \mathtt{M}_x(\mathcal{W}_x(t)) & := & 1 & \\
\mathtt{S}(\lambda x.t) & := & \mathtt{S}(t) & \mathtt{M}_x(\lambda y.t) & := & \mathtt{M}_x(t)+1 & \\
\mathtt{S}(t\ u) & := & \mathtt{S}(t)+\mathtt{S}(u) & \mathtt{M}_x(t\ u) & := & \mathtt{M}_x(t)+\mathtt{M}_x(u)+1 & \\
\mathtt{S}(t[x/u]) & := & \mathtt{S}(t)+\mathtt{M}_x(t)\cdot\mathtt{S}(u) & \mathtt{M}_x(t[y/u]) & := & \mathtt{M}_x(t) & \textit{If } x\notin \mathtt{fv}(u)\\
 & & & \mathtt{M}_x(t[y/u]) & := & \mathtt{M}_x(t)+\mathtt{M}_y(t)\cdot(\mathtt{M}_x(u)+1) & \textit{If } x\in \mathtt{fv}(u)
\end{array}
$$

Remark that $\mathtt{M}_x(s)\geq 1$ and $\mathtt{S}(s)\geq 1$ for every term s and every variable x.

This measure enjoys the following property:

Lemma 4.2 *Let s, s' be λrxw-terms.*

1. *If $s =_{\mathtt{E_{sw}}} s'$, then $\mathtt{S}(s)=\mathtt{S}(s')$.*
2. *If $s \longrightarrow_{\mathtt{WPush}} s'$, then $\mathtt{S}(s)=\mathtt{S}(s')$.*
3. *If $s \longrightarrow_{\mathtt{sw}\setminus\mathtt{WPush}} s'$, then $\mathtt{S}(s)>\mathtt{S}(s')$.*

Proof. The proof is by induction on $\longrightarrow_{\mathtt{esw}}$. ∎

Lemma 4.3 $\longrightarrow_{\mathtt{WPush}} /\mathtt{E_{sw}}$ *is a terminating system.*

Proof. For each term s we define a measure $\mathtt{P}(s)$ by induction as follows:

$$
\begin{array}{lcl}
\mathtt{P}(x) & := & 1\\
\mathtt{P}(t\ u) & := & 2\cdot\mathtt{P}(t)+2\cdot\mathtt{P}(u)\\
\mathtt{P}(\lambda x.t) & := & \mathtt{P}(t)+1\\
\mathtt{P}(\mathcal{W}_x(t)) & = & \mathtt{P}(t)+1\\
\mathtt{P}(t[x/u]) & := & \mathtt{P}(t)+2\cdot\mathtt{P}(u)
\end{array}
$$

Remark that $\mathtt{P}(s)\geq 1$ for every s.

Now, given s we consider $\langle \mathtt{nbw}(s), \mathtt{P}(s)\rangle$, where $\mathtt{nbw}(s)$ is the number of weakenings in s. We show that $s \longrightarrow_{\mathtt{WPush}/\mathtt{E_{sw}}} s'$ implies $\langle \mathtt{nbw}(s), \mathtt{P}(s)\rangle >_{lex} \langle \mathtt{nbw}(s'), \mathtt{P}(s')\rangle$.

The proof proceeds by induction on $\longrightarrow_{\mathtt{WPush}} / \mathtt{E}_{\mathtt{sw}}$.

We can then conclude that $\{\mathtt{WPush}\}/\mathtt{E}_{\mathtt{sw}}$-reduction is terminating on all $\lambda\mathtt{esw}$-terms by application of the abstract theorem A.1 : $\mathcal{E}$ is $=_{\mathtt{E}_{\mathtt{sw}}}$, $\mathcal{R}_1$ is the empty relation, $\mathcal{R}_2$ is $\longrightarrow_{\mathtt{WPush}}$, $\mathtt{K}$ is the relation given by the measure $\langle \mathtt{nbw}(\_), \mathtt{P}(\_)\rangle$ and $\mathcal{S}$ is $>_{lex}$ which is the standard (well-founded) lexicographic order on $\mathbb{N} \times \mathbb{N}$. ∎

In order to conclude with that the whole system $\mathtt{esw}$ is terminating on all $\lambda\mathtt{esw}$-terms we apply again Theorem A.1: $\mathcal{E}$ is $\mathtt{E}_{\mathtt{sw}}$, $\mathcal{R}_1$ is the relation $\longrightarrow_{\mathtt{WPush}}$ (so that $\longrightarrow_{\mathtt{WPush}} / \mathtt{E}_{\mathtt{sw}}$ is well-founded by Lemma 4.3), $\mathtt{K}$ is the relation given by the function $\mathtt{S}(\_)$, $\mathcal{R}_2$ is the relation $\longrightarrow_{\mathtt{sw}\setminus\{\mathtt{WPush}\}}$ which strictly decreases the measure $\mathtt{S}(\_)$ by Lemma 4.2 and $\mathcal{S}$ is the standard well-founded order $>$ on $\mathbb{N}$.

Corollary 4.4 *The reduction relation* $\mathtt{esw}$ *is terminating.*

4.2 Relating $\lambda\mathtt{es}$ and $\lambda\mathtt{esw}$

The aim of this section is to relate $\lambda\mathtt{es}$ and $\lambda\mathtt{esw}$-reduction in order to infer that$\lambda\mathtt{esw}$-normalisation implies $\lambda\mathtt{es}$-normalisation.

We start by giving a translation from $\lambda\mathtt{es}$-terms to $\lambda\mathtt{esw}$-terms which introduces as many weakening constructors as is necessary to build strict $\lambda\mathtt{esw}$-terms.

Definition 4.2 (From $\lambda\mathtt{es}$-terms to (strict $\lambda\mathtt{esw}$-terms) *The translation from $\lambda\mathtt{es}$-terms to strict $\lambda\mathtt{esw}$-terms is defined by induction as follows:*

$$
\begin{array}{lcll}
\mathtt{K}(x) & = & x & \\
\mathtt{K}(u\ v) & = & \mathtt{K}(u)\ \mathtt{K}(v) & \\
\mathtt{K}(\lambda x.t) & = & \lambda x.\mathtt{K}(t) & \textit{If } x \in \mathtt{fv}(t) \\
\mathtt{K}(\lambda x.t) & = & \lambda x.\mathcal{W}_x(\mathtt{K}(t)) & \textit{If } x \notin \mathtt{fv}(t) \\
\mathtt{K}(u[x/v]) & = & \mathtt{K}(u)[x/\mathtt{K}(v)] & \textit{If } x \in \mathtt{fv}(t) \\
\mathtt{K}(u[x/v]) & = & \mathcal{W}_x(\mathtt{K}(u))[x/\mathtt{K}(v)] & \textit{If } x \notin \mathtt{fv}(t)
\end{array}
$$

Remark that $\mathtt{fv}(\mathtt{K}(t)) = \mathtt{fv}(t)$.

The relevant point to relate now $\lambda\mathtt{es}$ and $\lambda\mathtt{esw}$-reduction consists in pulling out weakening constructors:

Lemma 4.5 *If* $s \longrightarrow_{\lambda\mathtt{es}} s'$, *then* $\mathtt{K}(s) \longrightarrow^{+}_{\lambda\mathtt{esw}} \mathcal{W}_{\mathtt{fv}(s)\setminus\mathtt{fv}(s')}(\mathtt{K}(s'))$.

Proof. By induction on $\longrightarrow_{\lambda\mathtt{es}}$.

∎

It is worth noticing that we really need in this proof $\mathtt{Weak1}$ and $\mathtt{WAbs}$ as equations and not as rewriting rules.

We can then now conclude this part with the main result of this section.

Corollary 4.6 *If* $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$, *then* $t \in \mathcal{SN}_{\lambda\mathtt{es}}$.

4.3 The Λ_I-calculus

Definition 4.3 *The set Λ_I of terms of the λI-calculus [Klo80] is defined by the following grammar:*

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

We only consider *strict* terms: every subterm $\lambda x.M$ satisfies $x \in \mathtt{fv}(M)$.

We use $[N, \langle M\rangle]$ or $[N, M_1, M_2, \ldots, M_n]$ to denote the term $[\ldots[[N, M_1], M_2], \ldots, M_n]$ 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.

As in the λ-calculus, the following property is straightforward by induction on terms.

Lemma 4.7 (Substitutions [Klo80]) *For all Λ_I-terms M, N, L, we have $M\{x/N\} \in \Lambda_I$ and $M\{x/N\}\{y/L\} = M\{y/L\}\{x/N\{y/L\}\}$ provided there is no variable capture.*

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

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

Figure 8: Reduction rules for Λ_I

The reduction relation $\beta\pi$ on Λ_I-terms preserves free variables.

Lemma 4.8 (Preservation of free variables) *Let $t \in \Lambda_I$. Then $t \longrightarrow_{\beta\pi} t'$ implies $\mathtt{fv}(t') = \mathtt{fv}(t)$.*

Proof. By induction on t using the fact that any abstraction in t is of the form $\lambda x.u$ with $x \in \mathtt{fv}(u)$. ∎

As a consequence $\beta\pi$-reduction preserves strict Λ_I-terms.

4.4 Relating $\lambda\mathtt{esw}$ and Λ_I

We now introduce a translation from $\lambda\mathtt{esw}$ to Λ_I by means of the relation $\_\ \mathcal{I}\ \_$. The reason to use a relation (and not a function) is that we want to translate the $\lambda\mathtt{esw}$-term into Λ_I-syntax by adding some *garbage* information which is not uniquely determined. Thus, each $\lambda\mathtt{esw}$-term can be projected in different Λ_I-terms, this will essential in the simulation property (Theorem 4.10).

Definition 4.4 *The relation $\mathcal{I}$ between* strict $\lambda\mathtt{esw}$*-terms and* strict Λ_I*-terms which is inductively given by the following rules:*

$$\frac{}{x\ \mathcal{I}\ x} \qquad \frac{t\ \mathcal{I}\ T}{\lambda x.t\ \mathcal{I}\ \lambda x.T} \qquad \frac{t\ \mathcal{I}\ T \quad u\ \mathcal{I}\ U}{t\ u\ \mathcal{I}\ T\ U} \qquad \frac{t\ \mathcal{I}\ T \quad u\ \mathcal{I}\ U}{t[x/u]\ \mathcal{I}\ T\{x/U\}}$$

$$\frac{t\ \mathcal{I}\ T}{t\ \mathcal{I}\ [T,M]}\ M \text{ is a } \Lambda_I\text{-term} \qquad \frac{t\ \mathcal{I}\ T}{\mathcal{W}_x(t)\ \mathcal{I}\ T}\ x \in \mathtt{fv}(T)$$

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

Lemma 4.9 *Let t be a $\lambda\mathtt{esw}$-term and M be a Λ_I-term. If $t\ \mathcal{I}\ M$, then*

1. $\mathtt{fv}(t) \subseteq \mathtt{fv}(M)$
2. $M \in \Lambda_I$
3. $x \notin \mathtt{fv}(t)$ *and* $N \in \Lambda_I$ *implies* $t\ \mathcal{I}\ M\{x/N\}$

Proof. Property (1) is a straightforward induction on the proof tree as well as Property (2) which also uses Lemma 4.7. Property (3) is also proved by induction on the tree, using Lemma 4.7. ∎

Remark that property 1 in Lemma 4.9 holds since we work with *strict* terms : indeed, the rule for substitution does not imply $\mathtt{fv}(t[x/u]) \subseteq \mathtt{fv}(T\{x/U\})$ when $x \notin \mathtt{fv}(t) \cup \mathtt{fv}(T)$. This is also an argument to exclude from our calculus rewriting rules not preserving strict terms like

$$\begin{array}{lllll} (\mathtt{App}) & (t\ u)[x/v] & \longrightarrow & (t[x/v]\ u[x/v]) & \\ (\mathtt{Comp}) & t[x/u][y/v] & \longrightarrow & t[y/v][x/u[y/v]] & \text{if } y \in \mathtt{fv}(u) \end{array}$$

Reduction in $\lambda\mathtt{esw}$ related to reduction in Λ_I by means of the following simulation property.

Theorem 4.10 (Simulation in Λ_I) *Let t be a $\lambda\mathtt{esw}$-term and T be a Λ_I-term.*

1. *If $s\ \mathcal{I}\ S$ and $s =_{\mathtt{E_{sw}}} s'$, then $s'\ \mathcal{I}\ S$.*
2. *If $s\ \mathcal{I}\ S$ and $s \longrightarrow_{\mathtt{sw}} s'$, then $s'\ \mathcal{I}\ S$.*
3. *If $s\ \mathcal{I}\ S$ and $s \longrightarrow_{\mathtt{B}} s'$, then there is $S' \in \Lambda_I$ such that $s'\ \mathcal{I}\ S'$ and $S \longrightarrow^{+}_{\beta\pi} S'$.*

Proof. By induction on the reduction/equivalence step. ∎

We can thus immediately conclude

Corollary 4.11 *If $t\ \mathcal{I}\ T$ and $T \in \mathcal{SN}_{\beta\pi}$, then $t \in \mathcal{SN}_{\lambda\mathtt{esw}}$.*

Proof. We apply the abstract theorem A.1: $\mathcal{E}$ is $=_{\mathtt{E_{sw}}}$, $\mathcal{R}_1$ is $\mathtt{sw}$, $\mathcal{R}_2$ is $\longrightarrow_{\mathtt{B}}$, $\mathtt{K}$ is the relation $\mathcal{I}$ and $\mathcal{S}$ is $\longrightarrow_{\beta\pi}$ which is well-founded on T by hypothesis. ∎

4.5 Solving the puzzle

In this section we put all the parts of the puzzle together in order to obtain preservation of β-strong normalisation.

Since we want to relate λ and $\lambda\mathtt{es}$-reduction, we first need to encode λ-terms into one of the calculi of this section. We proceed as follows.

Definition 4.5 ([Len05]) *Encoding of λ-terms into Λ_I is defined by induction follows:*

$$\begin{array}{lcll} \mathtt{I}(x) & := & x & \\ \mathtt{I}(\lambda x.t) & := & \lambda x.\mathtt{I}(t) & x \in \mathtt{fv}(t) \\ \mathtt{I}(\lambda x.t) & := & \lambda x.[\mathtt{I}(t), x] & x \notin \mathtt{fv}(t) \\ \mathtt{I}(t\ u) & := & \mathtt{I}(t)\ \mathtt{I}(u) & \end{array}$$

Theorem 4.12 (Lengrand[Len05]) *For any λ-term t, if $t \in \mathcal{SN}_\beta$, then $\mathtt{I}(t) \in WN_{\beta\pi}$.*

Theorem 4.13 (Nederpelt[Ned73]) *For any λ-term t, if $\mathtt{I}(t) \in WN_{\beta\pi}$ then $\mathtt{I}(t) \in SN_{\beta\pi}$.*

Theorem 4.14 *For any λ-term u, $\mathtt{K}(u)\ \mathcal{I}\ \mathtt{I}(u)$.*

Proof. By induction on u:

- $x\ \mathcal{I}\ x$ trivially holds.
- If $u = \lambda x.t$, then $\mathtt{K}(t)\ \mathcal{I}\ i(t)$ holds by the i.h. Therefore, we obtain $\lambda x.\mathtt{K}(t)\ \mathcal{I}\ \lambda x.i(t)$ in the case $x \in \mathtt{fv}(t)$ and $\lambda x.\mathcal{W}_x(\mathtt{K}(t))\ \mathcal{I}\ \lambda x.[i(t), x]$ in the case $x \notin \mathtt{fv}(t)$.
- If $u = (t\ v)$, then $\mathtt{K}(t)\ \mathcal{I}\ i(t)$ and $\mathtt{K}(v)\ \mathcal{I}\ i(v)$ hold by the i.h. and thus we can conclude $\mathtt{K}(t)\ \mathtt{K}(v)\ \mathcal{I}\ i(t)\ i(v)$.

∎

Corollary 4.15 ($\lambda\mathtt{esw}$ preserves β-strong normalisation) *For any λ-term t, if $t \in \mathcal{SN}_\beta$, then $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$.*

Proof. If $t \in \mathcal{SN}_\beta$, then $\mathtt{I}(t) \in SN_{\beta\pi}$ by Theorems 4.12 and 4.13. As $\mathtt{K}(t)\ \mathcal{I}\ \mathtt{I}(t)$ by Theorem 4.14, then we conclude $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$ by Corollary 4.11. ∎

Corollary 4.16 ($\lambda\mathtt{es}$ preserves β-strong normalisation) *For any λ-term t, if $t \in \mathcal{SN}_\beta$, then $t \in \mathcal{SN}_{\lambda\mathtt{es}}$.*

Proof. If $t \in \mathcal{SN}_\beta$, then $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$ by Corollary 4.15 and $t \in \mathcal{SN}_{\lambda\mathtt{es}}$ by Corollary 4.6. ∎

5 Recovering the untyped λ-calculus

We establish here the basic connections between λ and $\lambda\mathtt{es}$-reduction. As expected from a calculus with explicit substitutions, β-reduction can be implemented by $\lambda\mathtt{es}$ (Theorem 5.2) and $\lambda\mathtt{es}$-reduction can be projected into β (Corollary 5.4).

5.1 From λ-calculus to $\lambda\mathtt{es}$-calculus

We start by a simple lemma stating that explicit substitution can be used to implement meta-level substitution on pure-terms.

Definition 5.1 *The encoding of λ-terms into $\lambda\mathtt{es}$-terms is given by the identity function.*

Lemma 5.1 *Let t and u be λ-terms. Then $t[x/u] \longrightarrow^{+}_{\lambda\mathtt{es}} t\{x/u\}$*

Proof. By induction on the λ-term t.

∎

The *correctness* result obtained in the previous lemma enables us to prove a more general property concerning simulation of β-reduction in $\lambda\mathtt{es}$.

Theorem 5.2 (Simulating β-reduction) *Let t be a λ-term such that $t \longrightarrow_{\beta} t'$. Then $t \longrightarrow^{+}_{\lambda\mathtt{es}} t'$.*

Proof. By induction on β-reduction.

∎

5.2 From $\lambda\mathtt{es}$-calculus to λ-calculus

We now show how to encode a $\lambda\mathtt{es}$-term into a λ-term in order to project $\lambda\mathtt{es}$-reduction into β-reduction.

Definition 5.2 *Let t be a $\lambda\mathtt{es}$-term. We define the function $\mathtt{L}(t)$ by induction on the structure of t as follows:*

$$\begin{array}{lcl} \mathtt{L}(x) & := & x \\ \mathtt{L}(\lambda x.t) & := & \lambda x.\mathtt{L}(t) \\ \mathtt{L}(t\ u) & := & (\mathtt{L}(t)\ \mathtt{L}(u)) \\ \mathtt{L}(t[x/u]) & := & \mathtt{L}(t)\{x/\mathtt{L}(u)\} \end{array}$$

The translation $\mathtt{L}$ enjoys $\mathtt{fv}(\mathtt{L}(t)) \subseteq \mathtt{fv}(t)$.

Lemma 5.3 (Simulating $\lambda\mathtt{es}$-reduction)

1. *If $t =_{\mathtt{E_s}} u$, then $\mathtt{L}(t) = \mathtt{L}(u)$.*
2. *If $t \longrightarrow_{\mathtt{s}} u$, then $\mathtt{L}(t) = \mathtt{L}(u)$.*
3. *If $t \longrightarrow_{\mathtt{B}} u$, then $\mathtt{L}(t) \longrightarrow^{*}_{\beta} \mathtt{L}(u)$.*

Proof. By induction on $\lambda\mathtt{es}$-reduction.

1. This is obvious by the well-known [Bar84] substitution lemma of λ-calculus stating that for any λ-terms t, u, v, $t\{x/u\}\{y/v\} = t\{y/v\}\{x/\{u\{y/v\}\}$.

2. All the es-reduction steps are trivially projected into an equality.

3. A B-reduction step at the root of t corresponds exactly to a β-reduction step at the root of $\mathtt{L}(t)$ using the Definition of the translation.

■

We can finish this part with the following conclusion.

Corollary 5.4 *If* $t \longrightarrow_{\lambda\mathtt{es}} u$, *then* $\mathtt{L}(t) \longrightarrow_{\beta}^{*} \mathtt{L}(u)$.

6 The typed λes-calculus

In this section we present the *simply-typed* λes-calculus for which we show Subject Reduction in Section 6.2 and Strong Normalisation in Sections 8 and 4.5.

In contrast to standard systems for typed λ-calculus [] and typed λx-calculus [], for which typing judgements $\Gamma \vdash t : A$ are built in such a way that the free variables of t belong to Γ, we define here more precise typing rules which ensures that every environment Γ in a typing judgement $\Gamma \vdash t : A$ contains *exactly* the set of free variables of the term t it types. This property turns out to be essential to obtain tha simple translation of λes-terms into proof-nets that we given in Section 8.

Simply types are built over a countable set of atomic symbols $\mathcal{A}t$ by means of the following grammar:

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

where $\sigma \in \mathcal{A}t$.

An *environment* is a finite set of pairs of the form $x : A$. Two environments Γ and Δ are said to be *compatible* iff for all $x : A \in \Gamma$ and $y : B \in \Delta$, $x = y$ implies $A = B$. We denote the *union of compatible contexts* by $\Gamma \uplus \Delta$. Thus for example $(x : A, y : B) \uplus (x : A, z : C) = (x : A, y : B, z : C)$.

Set properties on environments are:

Remark 6.1

1. *If* $\Gamma \subseteq \Gamma'$ *and* $\Delta \subseteq \Delta'$, *then* $\Gamma \uplus \Delta \subseteq \Gamma' \uplus \Delta'$.

2. *If* Γ, Δ *and* Π *are all compatible, then* $\Gamma \uplus (\Delta \uplus \Pi) = (\Gamma \uplus \Delta) \uplus \Pi$.

6.1 Typing Rules

Typing judgements have the form $\Gamma \vdash t : A$ where t is a term, A is a type and Γ is an environment. *Derivations* of typing judgements can be obtained by application of the Typing Rules given in Figure 9.

In contrast to standard typing rules for λ-calculus [Bar92] and λx-calculus [LLD^{+}04], our axiom rule types a variable in a *singleton* environment. Variables which do not appear free in terms may be introduced by means of the $\mathtt{abs}_2$ or $\mathtt{subs}_2$ rule. As a consequence, the typing system enjoys the following property:

$$\frac{}{x : A \vdash x : A}\ (\mathtt{axiom}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Delta \vdash u : A}{\Gamma \uplus \Delta \vdash (t\ u) : B}\ (\mathtt{app})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\mathtt{abs}_1) \qquad \frac{\Gamma \vdash t : B \text{ and } x \notin \Gamma}{\Gamma \vdash \lambda x.t : A \to B}\ (\mathtt{abs}_2)$$

$$\frac{\Gamma \vdash u : B \qquad \Delta, x : B \vdash t : A}{\Gamma \uplus \Delta \vdash t[x/u] : A}\ (\mathtt{subs}_1) \qquad \frac{\Gamma \vdash u : B \qquad \Delta \vdash t : A \text{ and } x \notin \Gamma}{\Gamma \uplus \Delta \vdash t[x/u] : A}\ (\mathtt{subs}_2)$$

Figure 9: Typing Rules for $\lambda\mathtt{es}$-calculus

Lemma 6.2 *If* $\Gamma \vdash_{\lambda\mathtt{es}} t : A$, *then* $\Gamma = \mathtt{fv}(t)$.

Proof. by induction on typed derivations. ∎

6.2 Subject Reduction

As expected, the calculus enjoys the subject reduction property. More precisely, the calculus enjoys a *local* subject reduction property, that is, no meta-theorem is needed to show preservation of types.

Lemma 6.3 (Subject Reduction I) *If* $\Gamma \vdash_{\lambda\mathtt{es}} s : A$ *and* $s =_{\mathtt{E_s}} s'$, *then* $\Gamma \vdash_{\lambda\mathtt{es}} s' : A$.

Proof. By induction on $=_{\mathtt{E_s}}$. ∎

Lemma 6.4 (Subject Reduction II) *If* $\Pi \vdash_{\lambda\mathtt{es}} s : A$ *and* $s \longrightarrow_{\lambda\mathtt{es}} s'$, *then* $\Pi' \vdash_{\lambda\mathtt{es}} s' : A$ *for some* $\Pi' \subseteq \Pi$.

Proof. By induction on $\longrightarrow_{\lambda\mathtt{es}}$. ∎

7 Recovering the typed λ-calculus

We established in Sections 5.1 and 5.2 the connexion between the the two notions of reduction in λ and $\lambda\mathtt{es}$ which gives an *untyped* understanding of one calculus into the other one. We define here natural translations from typed λ-calculus to typed $\lambda\mathtt{es}$-calculus and vice-versa, thus completing the connection between λ and $\lambda\mathtt{es}$ in a type setting.

We first recall in Figure 10 the typing rules for λ-calculus.

A straightforward induction on typing derivations allows us to show the soundness of the projection of λ into $\lambda\mathtt{es}$:

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

Figure 10: Typing Rules for λ-calculus

Lemma 7.1 *If t is a λ-term s.t. $\Gamma \vdash_\lambda t : A$, then $\Gamma \cap \mathtt{fv}(t) \vdash_{\lambda\mathtt{es}} t : A$.*

Proof. By induction on the typing derivation $\Gamma \vdash_\lambda t : A$. ∎

The type derivations are also preserved in the other sense around. To show that, we first state the following known properties of typed lambda calculus (they can be shown by a straightforward induction on typing derivations).

Lemma 7.2

1. *If $\Gamma \vdash_\lambda t : A$, then $\Gamma, x : B \vdash_\lambda t : A$.*
2. *If $\Gamma, x : B \vdash_\lambda t : A$ and $\Gamma \vdash_\lambda u : B$, then $\Gamma \vdash_\lambda t\{x/u\} : A$.*

We can now conclude with the following.

Lemma 7.3 ($\mathtt{L}$ preserves types) *If t is a $\lambda\mathtt{es}$-term such that $\Gamma \vdash_{\lambda\mathtt{es}} t : A$, then $\Gamma \vdash_\lambda \mathtt{L}(t) : A$.*

Proof. By induction on the typing derivation $\Gamma \vdash_{\lambda\mathtt{es}} t : A$. ∎

8 Strong normalisation of typed $\lambda\mathtt{es}$-terms

In this section we present a translation of the typed $\lambda\mathtt{es}$-calculus into proof nets. To do so, we will translate simply types into MELL formulae, typed $\lambda\mathtt{es}$-terms into typed proof-nets, then we will show that $\lambda\mathtt{es}$-reduction can be simulated by a corresponding reduction relation on proof-nets which is known to be normalising.

This same technique has been already applied to other calculi with explicit substitutions and resources [DCK97, DCKP03, KL05].

8.1 Linear Logic's proof-nets

We recall here some classical notions from Linear Logic's proof-nets. We refer the interested reader to [Gir87] or [Laf95] for more details.

Let $\mathcal{A}t$ be a set of *atom symbols*. The set of formulae of the multiplicative exponential fragment of linear logic (called MELL) is defined by the grammar:

$$A ::= \sigma \mid \overline{\sigma} \mid A \otimes A \mid A \parr A \mid ?A \mid !A$$

where the atomic symbol σ in the formulae σ and $\overline{\sigma}$ belongs to the set $\mathcal{A}t$.

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

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

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

The set of proof-nets, that we denote by PN, is defined inductively in Figure 11 where we use rectangles having rounded corners to denote already defined nets used in the inductive constructions.

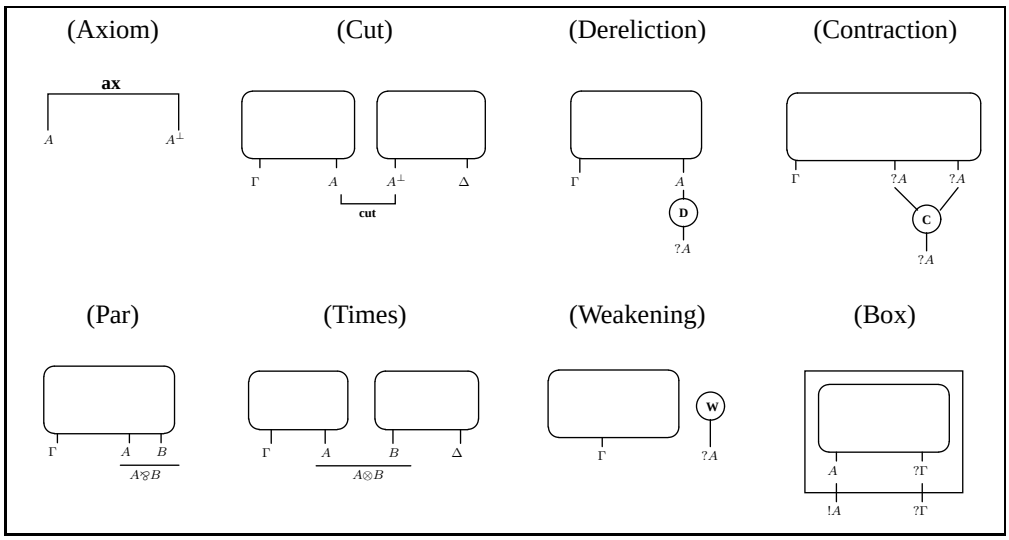

Figure 11: MELL Proof-nets

The traditional reduction system for MELL consists in the set of *cut elimination rules* appearing in Figure 12.

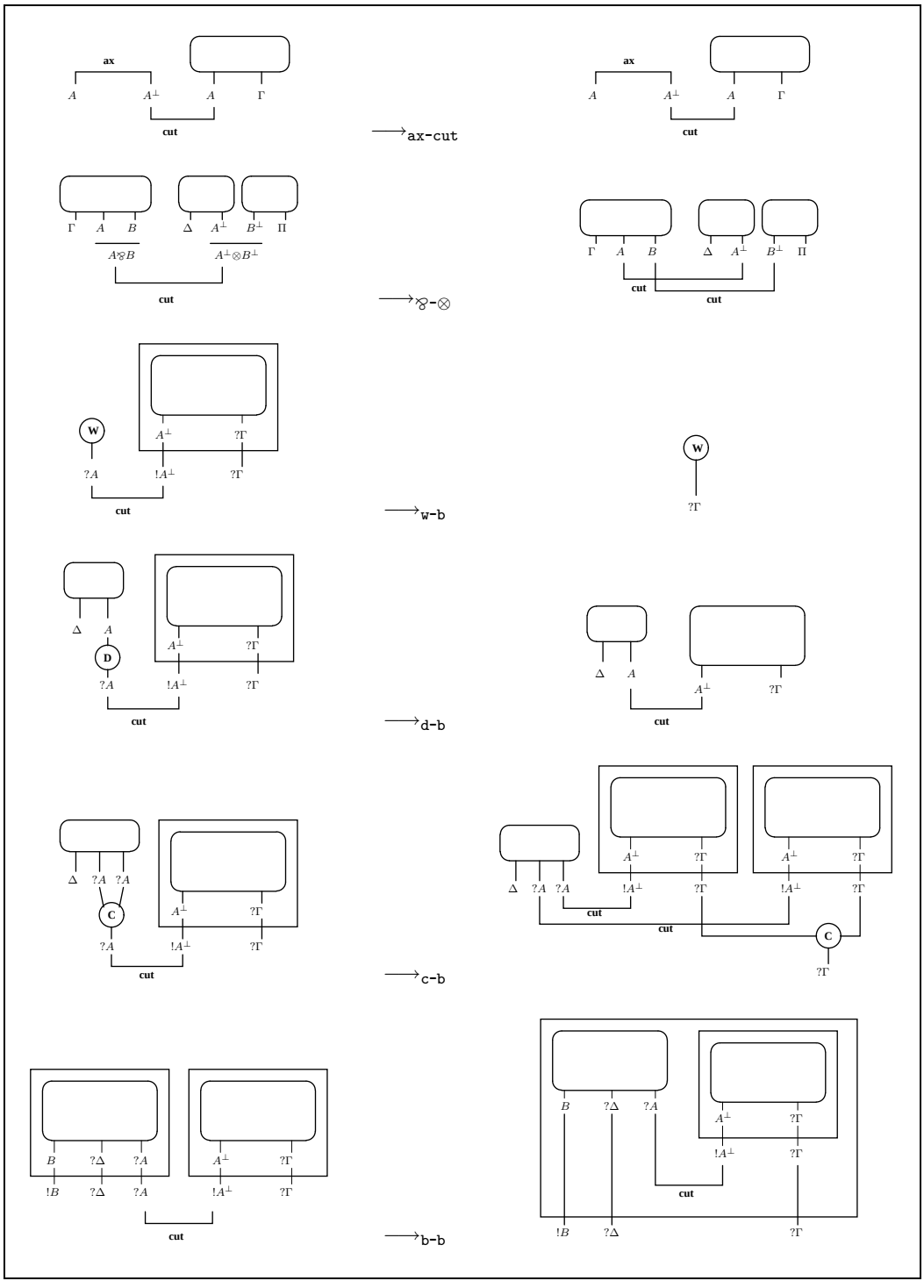

Figure 12: Cut elimination rules for MELL Proof-nets

We also consider an equivalence relation on PN, as in [DCG99], where two equations $\sim_{\mathtt{A}}$ and $\sim_{\mathtt{B}}$ are introduced (see Figure 13).

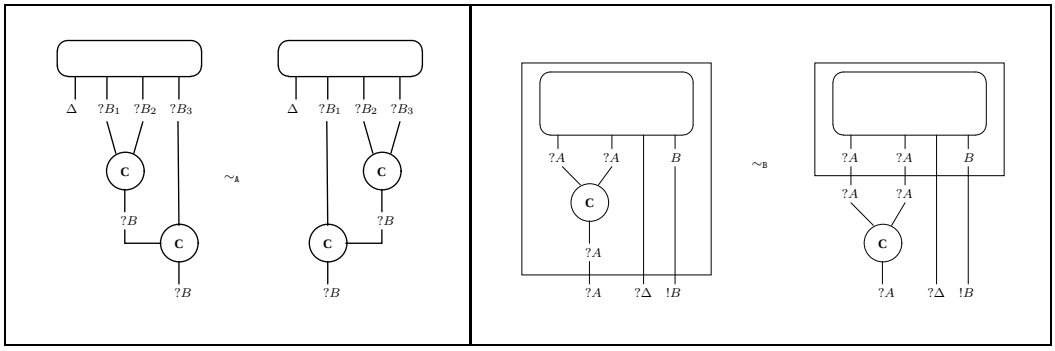

Figure 13: Equations for MELL proof-nets

Finally, we shall also use the two extra reduction rules in Figure 14 : `U` is used to simplify weakening linked to contraction nodes and `V` allows weakening links to go outside boxes in order to bring them together at the top of the proof-nets.

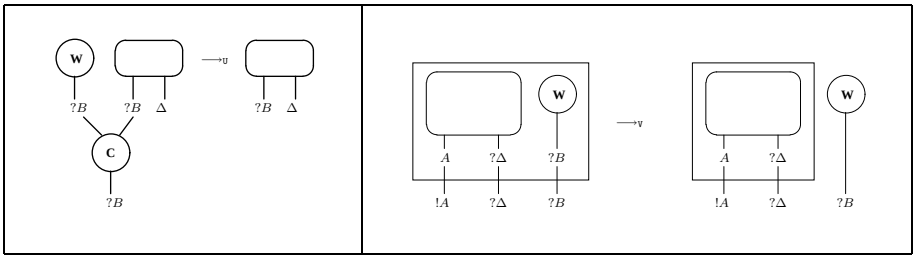

Figure 14: Extra reduction rules for MELL proof-nets

W call R the system made of rules `ax-cut`, $⅋$-$\otimes$, `w-b`, `d-b`, `c-b`, `b-b` and `U` and `V`. We shall write $\sim_E$ for the congruence (reflexive, symmetric, transitive, closed by proof-net contexts) relation on proof-nets generated by equations `A`, `B`. We shall write R/E for the reduction relation generated by the rules in R and the equations in $\sim_E$, given by $r \longrightarrow_{R/E} s$ if and only if there exist r' and s' such that $r \sim_E r' \longrightarrow_R s' \sim_E s$.

The following result is well-known [Pol04] (see also [KL05] for details).

Theorem 8.1 *The reduction relation R/E on typed proof-nets is strongly normalising.*

8.2 From λes-terms to Proof-nets

We now present the natural translation from λes-terms to proof-nets. For that, let's start by the usual translation of intuitionistic types [Gir87] into MELL formulae given by :

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

Now we can give our translation $T(\_)$ from typed λes-terms to proof-nets, which is defined by induction on the derivation of typing judgements as shown in Figure 15. Every proof-net $T(\Gamma \vdash t : A)$ has one wire labelled with $?(D^*)^\perp$ for every $D \in \Gamma$ and one unique wire labelled with A^*. We shall often write $T(t)$ instead of $T(\Gamma \vdash t : A)$ when Γ and A are clear from the context.

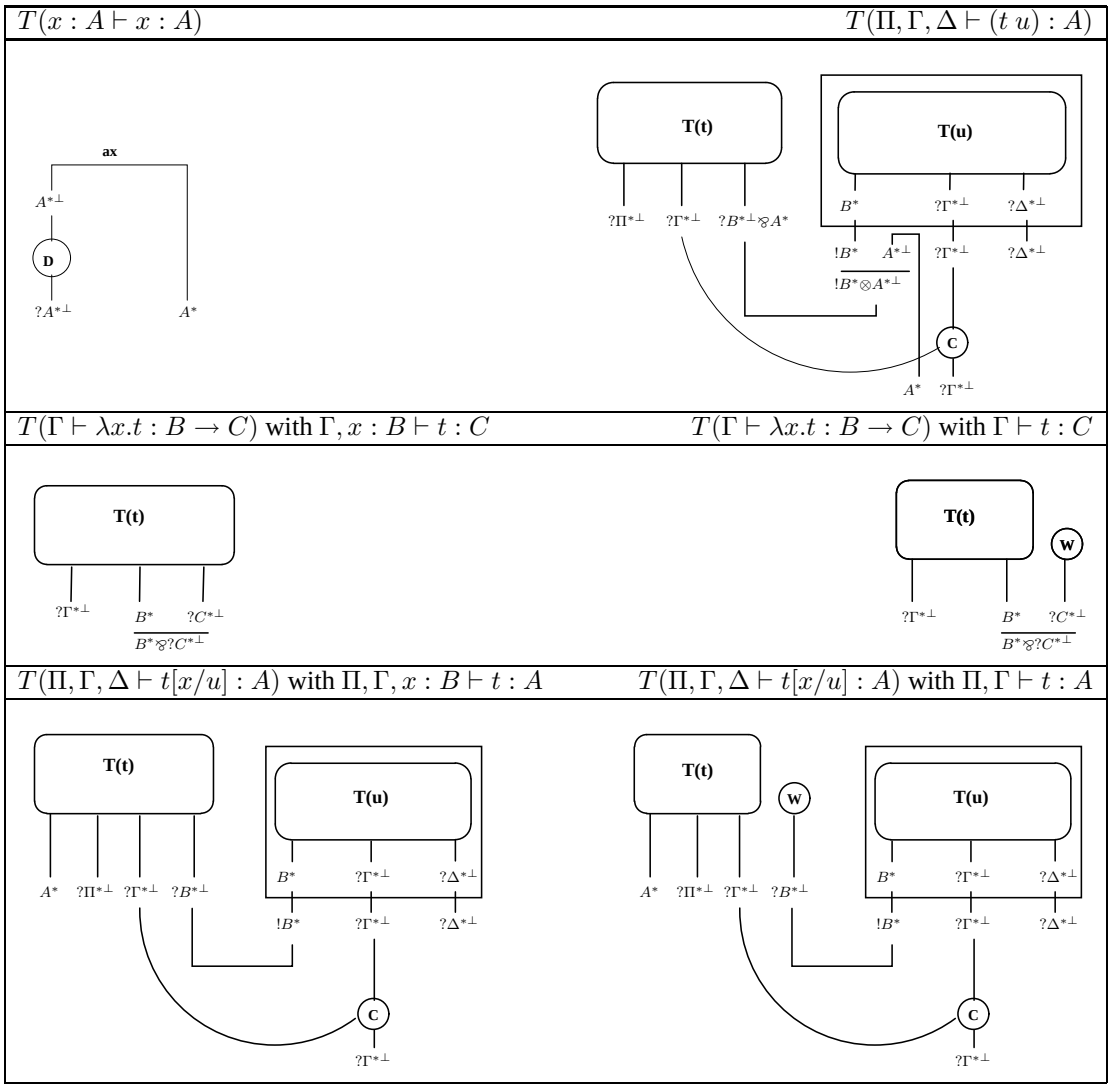

Figure 15: Encoding typed λes-terms into MELL proof-nets

Now we can state the main theorem of this section. The proof also justifies the use of the additional equations A and B as well as the additional reduction rules V and U. In the following statement, we write $C[p]$ the proof-net obtained from p by adding a finite number of weakening wires on the top level of p (outside all the boxes).

Theorem 8.2 *Let s be a λes-typed term.*

1. *If $s =_{\mathtt{E_s}} s'$, then $T(s) \sim_e T(s')$.*

2. *If* $s \longrightarrow_{\mathtt{App}_3,\mathtt{Lamb}} s'$, *then* $T(s) \sim_e T(s')$.

3. *If* $s \longrightarrow_{\mathtt{Bs}\setminus\{\mathtt{App}_3,\mathtt{Lamb}\}} s'$, *then* $T(s) \longrightarrow^{+}_{R/E} C[T(s')]$.

Proof. The proof proceeds by induction on $\longrightarrow_{\lambda\mathtt{es}}$. We first show that cases where $s \longrightarrow_{\lambda\mathtt{es}} s'$ is an external reduction step, for which we consider all the root reduction/equivalence cases.

▪

Remark that the only case where we get a non empty context in Lemma 8.2 is when simulating the rule Gc. This is because Gc is the only rule which looses free variables, all the other ones preserve the same set of free variables.

Corollary 8.3 (SN for $\lambda\mathtt{es}$-typed terms) *If* $\Gamma \vdash_{\lambda\mathtt{es}} t : A$, *then* $t \in \mathcal{SN}_{\lambda\mathtt{es}}$.

Proof. We can apply the abstract theorem A.1 : $\mathcal{E}$ is $\mathtt{E_s}$, $\mathcal{R}_1$ is the relation $\longrightarrow_{\mathtt{App}_3,\mathtt{Lamb}}$ (for which we can trivially show that $\longrightarrow_{\mathtt{App}_3,\mathtt{Lamb}} / =_{\mathcal{E}}$ is well-founded), $\mathcal{R}_2$ is the relation $\longrightarrow_{\mathtt{es}\setminus\{\mathtt{App}_3,\mathtt{Lamb}\}}$, K is the relation given by the translation $T(\_)$, $\mathcal{S}$ is the reduction relation R/E on proof-nets which is well-founded on typed proof-nets by Theorem 8.1 and properties **(ES),(WS),(SS)** hold by Lemma 8.2. ▪

8.3 Discussion

In this section we want to discuss some other alternative typing/reduction rules appearing in the litterature for calculi with ES.

As mentioned in Section 2 one is tempted to replace rules $\{\mathtt{App}_1, \mathtt{App}_2, \mathtt{App}_3\}$ by the single rule

$$(\mathtt{App}) \qquad (t\ u)[x/v] \longrightarrow (t[x/v]\ u[x/v])$$

where no condition is used to distribute the explicit substituton $[x/u]$ w.r.t the application $(t\ u)$.

In the typing system presented in Section 6.1 this rule would be sound, i.e. subject reduction holds. However, $(\mathtt{App})$ could not be translated anymore to proof-nets. Indeed, suppose x is free in u but not in t. Then the proof-net s obtained by translating the $\lambda\mathtt{es}$-term $(t\ u)[x/v]$ contains a cut between the wire representing x which is coming out from the box containing $T(u)$ and the single !-wire coming out from the box containing $T(v)$. It is evident that s does not reduce to the proof-net $s' = T(t[x/v]\ u[x/v])$ since the box containing $T(v)$ in s cannot be duplicated at all to obtain s'.

However, this problem could be solved by using a more standard *additive* typing system for explicit substituions [Blo97] where the axioms are *weakened*, there is a single rule for abstraction and rules for application and substituion are *additive* :

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\mathtt{axiom}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\ (\mathtt{app})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\mathtt{abs}) \qquad \frac{\Gamma \vdash u : B \qquad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x/u] : A}\ (\mathtt{subs})$$

Now, the Lamb-rewrite rule in Figure 3 cannot be translated anymore to R/E-reduction in proof-nets as subject reduction becomes non local: in order to construct a typing derivation of $\lambda y.t[x/u]$ from that of $(\lambda y.t)[x/u]$ one needs a weakening meta-theorem saying that $\Gamma \vdash u : B$ implies $\Gamma, y : A \vdash u : B$. It is evident that this kind of manipulation on proof-nets is not possible during R/E-reduction.

A third possible typing system coming up which makes possible the translation of the App and Lamb-rewrite rules into proof-nets is the one appearing in [DCK97] : the subs-typing rule is replaced by

$$\frac{\Gamma \vdash u : B \qquad \Gamma, x : B, \Delta \vdash t : A}{\Gamma, \Delta \vdash t[x/u] : A}$$

Unfortunately, it is straightforward to verify that rewriting rules $\mathtt{Comp}_1$ and $\mathtt{Comp}_2$ (not considered in [DCK97]) do not enjoy anymore subject reduction.

Summing up, while the standard additive typing system for ES gives a technical solution to prove the subject reduction property for $\lambda\mathtt{es}$ and its more compact variants mentioned in Section 2, it does not provide a correct tool to translated $\lambda\mathtt{es}$ into proof-nets.

9 PSN implies SN

We give here a second proof of strong-normalisation for $\lambda\mathtt{es}$-typed terms. The proof-technique we use here to derive strong normalisation from PSN was suggested by Hugo Herbelin some years ago.

Theorem 9.1 (Strong Normalisation) *Every typable $\lambda\mathtt{es}$-term M is in $SN_{\lambda\mathtt{es}}$.*

Proof. Let us define the following translation $\mathtt{C}()$ from $\lambda\mathtt{es}$-terms to λ-terms:

$$\begin{array}{lcl} \mathtt{C}(x) & := & x \\ \mathtt{C}(MN) & := & \mathtt{C}(M)\,\mathtt{C}(N) \\ \mathtt{C}(\lambda x.M) & := & \lambda x.\mathtt{C}(M) \\ \mathtt{C}(M[x/N]) & := & (\lambda x.\mathtt{C}(M))\,\mathtt{C}(N) \end{array}$$

Thus for example, $\mathtt{C}((x[x/y]\ z)[w/(w_1\ w_2)]) = (\lambda w.((\lambda x.x)\ y)\ z)(w_1\ w_2)$.

We remark that for every $\lambda\mathtt{es}$-term one has $\mathtt{C}(M) \longrightarrow^*_{\lambda\mathtt{es}} M$. We also remark that when M is typable in $\lambda\mathtt{es}$, then also $\mathtt{C}(M)$ is typable in $\lambda\mathtt{es}$ (just change the use of $\mathtt{subs}_1$ and $\mathtt{subs}_2$ by $\mathtt{abs}_1$ and $\mathtt{abs}_2$ followed by app). By Lemma 7.3 the term $\mathtt{L}(\mathtt{C}(M)) = \mathtt{C}(M)$ is also typable in simply typed λ-calculus and thus it is in SN_β by Strong Normalisation of typed λ-calculus [Bar92]. As a consequence we have that $\mathtt{C}(M)$ is in $SN_{\lambda\mathtt{es}}$ by Theorem 4.16 and thus M is necessarily in $SN_{\lambda\mathtt{es}}$ too. ∎

We remark that this proof technique, which is very simple in the case of the $\lambda\mathtt{es}$-calculus, needs some additional work to be applied to other calculi [Pol04, Arb06].

10 Conclusion

In this paper we survey some properties concerning explicit substitutions calculi and we describe work done in the domain during these last 15 years.

As we pointed out in [DCK97], "the interpretation of explicit substitution via Linear Logic's proof-nets suggests that there really exists a typed calculus of explicit substitution with full composition, being able to simulate any one-step β-reduction and yet strongly normalizing (thus avoiding Mellies' counterexample): indeed, the composition of substitution is already present in the proof-nets reduction system, as the box-box reduction, yet strong normalization is not lost."

We propose here simple syntax and simple equations and rules to modelise a formalism enjoying all these good properties, specially confluence on metaterms, preservation of β-strong normalisation, strong normalisation of typed terms and implementation of full composition.

We believe however that some of our proofs can be simplified. In particular, PSN and confluence on metaterms might be proved directly without using translations of $\lambda\mathtt{es}$ to other formalisms. We leave this for futur work.

Another interesting issue is the extension of Pure Type Systems (PTS) with explicit substitution systems in order to improve the understanding of proof systems based on them. Some work already done in this direction uses sequent calculi [LDM06], some other [KL04, Muñ97] use an intermediate formalism between natural deduction and sequent calculi, which is obtained by adding a system with ES to λ-calculus. The main contribution of $\lambda\mathtt{es}$ w.r.t these formalisms previously mentioned would be our sound notion of composition which is necessary to obtain a system preserving types [KL04].

It is also legitimate to ask whether $\lambda\mathtt{es}$ is minimal w.r.t. the number of rewriting rules as one is tempted to gather the rules $\{\mathtt{App}_1, \mathtt{App}_2, \mathtt{App}_3\}$ (resp. $\{\mathtt{Comp}_1, \mathtt{Comp}_2\}$) into one single rule for application (resp composition). The resulting calculus would be given by

Equations :			
$t[x/u][y/v]$	$=_{\mathtt{C}}$	$t[y/v][x/u]$	if $y \notin \mathtt{fv}(u)$ & $x \notin \mathtt{fv}(v)$
Reduction Rules :			
$(\lambda x.t)\ u$	$\longrightarrow_{\mathtt{B}}$	$t[x/u]$	
$x[x/u]$	$\longrightarrow_{\mathtt{Var}}$	u	
$t[x/u]$	$\longrightarrow_{\mathtt{Gc}}$	t	if $x \notin \mathtt{fv}(t)$
$(t\ u)[x/v]$	$\longrightarrow_{\mathtt{App}}$	$(t[x/v]\ u[x/v])$	
$(\lambda y.t)[x/v]$	$\longrightarrow_{\mathtt{Lamb}}$	$\lambda y.t[x/v]$	if $y \notin \mathtt{fv}(v)$ & $x \neq y$
$t[x/u][y/v]$	$\longrightarrow_{\mathtt{Comp}}$	$t[y/v][x/u[y/v]]$	if $y \in \mathtt{fv}(u)$

Note that $\lambda\mathtt{es}$-reduction can be translated to the correspondent notion of reduction in this calculus : thus for example $\mathtt{App}_1$ can be obtained by App followed by Gc. Besides that, strong normalisation of this calculus, which we conjecture to hold, cannot be obtained via a standard translation to Girard's proof-nets (c.f. discussion in Section 8.3).

Another interesting question is whether we can extract from $\lambda\mathtt{es}$ a pure rewriting system (without equations) verifying the same properties than $\lambda\mathtt{es}$. We believe that

simultaneous substitutions will be needed for that, even if translation to proof-nets will be much more intricated. Also, a total order $>$ on variables would be necessary in order to obtain canonical representatives for simultaneous substitutions. The first ideas of such a solution could be found in the ss-calculus defined in Section 3.1. A more elementary representation of a calculus with simultaneous substitutions and controlled composition could be given by

Terms

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[s] \mid t(s)$$

Substitutions

$$s ::= id \mid x/u.s \mid s \circ s$$

Reduction Rules

$(\lambda x.t)\ u$	$\longrightarrow$	$t[x/u]$	
$(t\ u)[s]$	$\longrightarrow$	$(t[s]\ u[s])$	
$(\lambda x.t)[s]$	$\longrightarrow$	$\lambda x.t[s]$	
$x[(x/u).s]$	$\longrightarrow$	u	
$t[(x/u).s]$	$\longrightarrow$	$t[s]$	If $x \notin \mathtt{fv}(t)$
$t[s][p]$	$\longrightarrow$	$t[s \circ p]$	
$(s \circ p) \circ q$	$\longrightarrow$	$s \circ (p \circ q)$	
$id \circ s$	$\longrightarrow$	s	
$x[id]$	$\longrightarrow$	x	
$(x/u.s) \circ p$	$\longrightarrow$	$x/u(t).(s \circ p)$	
$u(id)$	$\longrightarrow$	id	
$u(y/v.s)$	$\longrightarrow$	$u[y/v](s)$	If $y \in \mathtt{fv}(u)$
$u(y/v.s)$	$\longrightarrow$	$u(s)$	If $y \notin \mathtt{fv}(u)$
$y/v.x/u.s$	$\longrightarrow$	$x/u.y/v.s$	If $x < y$

Then, one can verify for example that the critical pair

$$t[y/v.id][x/u[y/v.id].id] \ {}^{*}\!\longleftarrow ((\lambda x.t)\ u)[y/v.id] \longrightarrow^{*} t[x/u.id][y/v.id]$$

can be closed by $t[x/u[y/v.id].y/v.id]$ when $y \in \mathtt{fv}(u)$, or by $t[x/u.y/v.id]$ when $y \notin \mathtt{fv}(u)$, if $x < y$ holds in the total order on variables which is necessary to obtain a canonical order between simultaneous substitutions.

Acknowledgements

This work has received substantial benefit from fruitful discussions with my colleagues E. Bonelli, R. David, R. Di Cosmo, J-P. Jouannaud, S. Lengrand, C. Muñoz and V. van Oostrom.

A Appendix: An abstract theorem

Theorem A.1 *Let $\mathcal{R}_1, \mathcal{R}_2$ (resp. $\mathcal{E}$) be two relations (resp. equivalence relation) over $\mathcal{O}$ such that $\mathcal{R}_1/\mathcal{E}$ is well-founded. Let consider a relation* K *on $\mathcal{O} \times \mathcal{P}$ such that*

(ES) *$t \mathcal{E} t'$ and $t \mathtt{K} T$ implies $t' \mathtt{K} T$*

(WS) *$t \mathcal{R}_1 t'$ and $t \mathtt{K} T$ implies there is T' such that $t' \mathtt{K} T'$ and $T \mathcal{S}^* T'$*

(SS) *$t \mathcal{R}_2 t'$ and $t \mathtt{K} T$ implies there is T' such that $t' \mathtt{K} T'$ and $T \mathcal{S}^+ T'$*

Then, if $t \mathtt{K} T$ and $\mathcal{S}$ is a well-founded relation on T, then $(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{E}$ is well-founded on t.

Proof. Suppose $(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{E}$ is not well-founded on t. Since $\mathcal{R}_1/\mathcal{E}$ is well-founded by hypothesis, then there is an infinite sequence on $\mathcal{O}$ of the form

$$t \longrightarrow^*_{\mathcal{R}_1/\mathcal{E}} \longrightarrow_{\mathcal{R}_2} t_1 \cdots \longrightarrow^*_{\mathcal{R}_1/\mathcal{E}} \longrightarrow_{\mathtt{B}} t_i \longrightarrow^*_{\mathcal{R}_1/\mathcal{E}} \longrightarrow_{\mathtt{B}} \cdots$$

By hypothesis there are $T_1, \ldots, T_i, \ldots \in \mathcal{P}$ such that $t \mathtt{K} T, t_1 \mathtt{K} T_1, \ldots, t_i \mathtt{K} T_i, \ldots$ and the following infinite $\mathcal{S}$-reduction sequence exists

$$T \mathcal{S}^+ T_1 \mathcal{S}^+ \ldots \mathcal{S}^+ T_i \mathcal{S}^+ \cdots$$

This leads to a contradiction with the fact that $\mathcal{S}$ is well-founded on T. ∎

References

[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.

[ACCL91a] M. Abadi, L. Cardelli, P. L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[ACCL91b] M. Abadi, L. Cardelli, P. L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[AJ92] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Logic in Computer Science (LICS)*, pages 211–222, 1992.

[Arb06] A. Arbiser. *Explicit Substitution Systems and Subsystems*. PhD thesis, Universidad Buenos Aires, Argentina, 2006.

[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.

[Bar92] H. Barendregt. Lambda calculus with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.

[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.

[Blo97] R. Bloo. *Preservation of Termination for Explicit Substitution*. PhD thesis, Eindhoven University of Technology, 1997.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BR95a] R. Bloo and K. Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. In *Computer Science in the Netherlands (CSN)*, pages 62–72, 1995.

[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 (RTA)*, 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, March 2000.

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

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

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

[DHK00] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157:183–235, 2000.

[ELA] The ELAN system. `http://elan.loria.fr/`.

[FdK] M. A.-R. Flavio de Moura and F. Kamareddine. Higher order unification: A structural relation between huet's method and the one based on explicit substitution. Available from `http://www.macs.hw.ac.uk/~fairouz/papers/`.

[FKP96] M. C. Ferreira, D. Kesner, and L. Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalization (extended abstract). In M. Hanus and M. Rodrguez-Artalejo, editors, *5th International Conference on Proceedings of International Symposium Algebraic and Logic Programming (ALP)*, volume 1139 of *Lecture Notes in Computer Science*, pages 284–298. Springer-Verlag, September 1996.

[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 (RTA)*, volume 2378 of *Lecture Notes in Computer Science*, pages 174–191. Springer-Verlag, July 2002.

[GAL92] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of POPL*, pages 15–26, Albuquerque, New Mexico, 1992. Association for Computing Machinery.

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

[Gir89] J.-Y. Girard. Geometry of interaction I: interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic colloquium 1988*, pages 221–260. North Holland, 1989.

[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, December 1998.

[GL99] J. Goubault-Larrecq. Conjunctive types and SKInT. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Proceedings of the International Workshop Types for Proofs and Programs*, volume 1657 of *Lecture Notes in Computer Science*, pages 106–120. Springer-Verlag, March 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.

[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, September 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*, pages 25–33. ACM Press, May 1996.

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

[HPJP92] P. Hudak, S. Peyton-Jones, and Philip Wadler (editors). Report on the programming language Haskell, a non-strict, purely functional language (version 1.2). Sigplan Notices, 1992.

[Hue76] G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Thèse de doctorat d'état, Université Paris VII, 1976.

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

[JK86] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.

[Kes96] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions. In H. Ganzinger, editor, *7th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1103 of *Lecture Notes in Computer Science*, pages 184–199. Springer-Verlag, July 1996.

[Kes00] D. Kesner. Confluence of extensional and non-extensional lambda-calculi with explicit substitutions. *Theoretical Computer Science*, 238(1-2):183–220, 2000.

[Kes06] D. Kesner. The theory of calculi with explicit substitutions revisited, 2006. Available as `http://www.pps.jussieu.fr/~kesner/papers`.

[KL04] R. Kervarc and P. Lescanne. Pure type systems, cut and explicit substitutions. In D. Kesner, F. van Raamsdonk, and J. Wells, editors, *2nd International Workshop on Higher-Order Rewriting (HOR'04)*, Technical Report AIB-2004-03, RWTH Aachen, pages 72–77, June 2004.

[KL05] D. Kesner and S. Lengrand. Extending the explicit substitution paradigm. In J. Giesl, editor, *16th International Conference on Rewriting Techniques and Applications (RTA)*, volume 3467 of *Lecture Notes in Computer Science*, pages 407–422. Springer-Verlag, April 2005.

[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 (RTA)*, 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, September 1995.

[KS88] R. Kennaway and R. Sleep. Director strings as combinators. *ACM Transactions on Programming Languages and Systems*, 10(4):602–626, 1988.

[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.

[Lam90] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Procedings of POPL*, pages 16–30, San Francisco, California, 1990. Association for Computing Machinery.

[LDM06] S. Lengrand, R. Dyckhoff, and J. McKinna. A sequent calculus for type theory. In Z. Esik, editor, *Proceedings of the 15th Annual Conference of the European Association for Computer Science Logic (CSL)*, volume 4207 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2006.

[LEG] The LEGO Proof Assistant. `http://www.dcs.ed.ac.uk/home/lego/`.

[Len05] S. Lengrand. Induction principles as the foundation of the theory of normalisation: Concepts and techniques. Technical report, PPS laboratory, Université Paris 7, March 2005. available at `http://hal.ccsd.cnrs.fr/ccsd-00004358`.

[Len06] S. Lengrand. *Normalisation and Equivalence in Proof Theory and Type Theory*. PhD thesis, University Paris 7 and University of St Andrews, November 2006.

[Les94] P. Lescanne. From λ_σ to λ_υ, a journey through calculi of explicit substitutions. In *Proceedings of the 21st Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 60–69. ACM, 1994.

[Lin86] R. Lins. A new formula for the execution of categorical combinators. In *8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 89–98. Springer-Verlag, August 1986.

[Lin92] R. Lins. Partial categorical multi-combinators and Church Rosser theorems. Technical Report 7/92, Computing Laboratory, University of Kent at Canterbury, May 1992.

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

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

[LRD95] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn levels. In J. Hsiang, editor, *6th International Conference on Rewriting Techniques and Applications (RTA)*, volume 914 of *Lecture Notes in Computer Science*, pages 294–308. Springer-Verlag, April 1995.

[Mau] The MAUDE System. `http://maude.cs.uiuc.edu/`.

[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 (TLCA)*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334. Springer-Verlag, April 1995.

[MTH90] R. Milner, M. Tofte, and R. Harper. *The definition of Standard ML*. MIT Press, 1990.

[Muñ97] C. Muñoz. *Un calcul de substitutions pour la reprsentation de preuves partielles en thorie de types*. PhD thesis, Université Paris 7, November 1997.

[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.

[Oca] The Objective Caml language. `http://caml.inria.fr/`.

[Ohl98] E. Ohlebusch. Church-rosser theorems for abstract reduction modulo an equivalence relation. In T. Nipkow, editor, *9th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1379 of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, April 1998.

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

[PVS] The PVS system. `http://pvs.csl.sri.com/`.

[Ros92] K. Rose. Explicit cyclic substitutions. In M. Rusinowitch and J.-L. Rmy, editors, *Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems (CTRS)*, volume 656 of *Lecture Notes in Computer Science*, pages 36–50. Springer-Verlag, July 1992.

[Sak] T. Sakurai. Strong normalizability of calculus of explicit substitutions with composition. Draft.

[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 (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 46–60. Springer-Verlag, June 2003.

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[Tur85] D. Turner. Miranda: A non-strict functional language with polymorphic types. In *fplc85*, pages 1–16, 1985.