The theory of calculi with explicit substitutions revisited

Delia Kesner

PPS, Université Paris 7 and CNRS (UMR 7126), France

Abstract. Calculi with explicit substitutions (ES) are widely used in different areas of computer science. Complex systems with ES were developed these last 15 years to capture the good computational behaviour of the original systems (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 development of such calculi. Then we use very simple technology to establish a general theory of explicit substitutions for the lambda-calculus which enjoys fundamental properties such as simulation of one-step beta-reduction, confluence on metaterms, preservation of beta-strong normalisation, strong normalisation of typed terms and full composition. The calculus also admits a natural translation into Linear Logic's proof-nets.

1 Introduction

This paper is about *explicit substitutions* (ES), an intermediate formalism that - by decomposing the *higher-order* substitution operation into more atomic steps - allows a better understanding of the execution models of complex languages.

Indeed, higher-order substitution is a *meta-level* operation used in higher-order languages (such as functional, logic, concurrent and object-oriented programming), while ES is an *object-level* notion internalised and handled by symbols and reduction rules belonging to their own worlds. However, the two formalisms are still very close, this can be easily seen for example in the case of the λ-calculus whose reduction rule is given by $(\lambda x.t)\ u \to_\beta t\{x/u\}$, where the operation $t\{x/v\}$ denotes the result of substituting all the *free* occurrences of x in t by u, a notion that can be formally defined *modulo α-conversion* [1] as follows:

$$\begin{array}{ll} x\{x/u\} := u & (t_1\ t_2)\{x/u\} := (t_1\{x/u\}t_2\{x/u\}) \\ y\{x/u\} := y\ (x \neq y) & (\lambda y.v)\{x/u\} := \lambda y.v\{x/u\} \end{array}$$

Then, the simplest way to specify a λ-calculus with ES is to incorporate substitutions into the language, then to transform the equalities of the previous specification into reduction rules (so that one still works modulo α-conversion), thus yielding the following reduction system known as λx [36, 37, 44, 10].

$$\begin{array}{lll} (\lambda x.t)\ u & \to t[x/u] & \\ x[x/u] & \to u & \\ y[x/u] & \to y & (x \neq y) \\ (t_1\ t_2)[x/u] & \to (t_1[x/u]\ t_2[x/u]) & \\ (\lambda y.v)[x/u] & \to \lambda y.v[x/u] & \end{array}$$

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

The $\lambda\mathtt{x}$-calculus corresponds to the minimal behaviour [2] that can be found in most of the calculi with ES appearing in the literature. More sophisticated treatments of substitutions also consider a composition operator allowing much more interactions between them. This is exactly the source of the problems that we discuss below.

Related Work In these last years there has been a growing interest in λ-calculi with ES. They can be defined either with unary [44,35] or n-ary [2,23] substitutions, by using de Bruijn notation [11,12,32,27], or levels [39], or combinators [20], or director strings [46], or ... simply by named variables as in $\lambda\mathtt{x}$. Also, a calculus with ES can be seen as a term notation for a logical system where the reduction rules behave like cut elimination transformations [22,29,16].

In any case, all these calculi were introduced as a bridge between formal higher-order calculi and their concrete implementations. However, implementing an atomic substitution operation by several elementary explicit steps comes at a price. Indeed, while λ-calculus is perfectly *orthogonal* (does not have critical pairs), calculi with ES such as $\lambda\mathtt{x}$ suffer at least from the following well-known diverging example:

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

Different solutions were adopted in the literature to close this diagram. If no new rewriting rule is added to those of $\lambda\mathtt{x}$, then reduction turns out to be confluent on terms but not on *metaterms* (terms with metavariables used to represent incomplete programs and proofs). If naive rules for composition are considered, then one recovers confluence on metaterms but loses normalisation: there exist terms which are strongly normalisable in λ-calculus but not in the corresponding ES version. This phenomenon, known as Melliès' counter-example [40], shows a flaw in the design of ES calculi 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 $\lambda_{\mathtt{Z}}$-*calculus* an arbitrary set of ($\lambda_{\mathtt{Z}}$-)terms together with a set of ($\lambda_{\mathtt{Z}}$-)reduction rules. Also, let us consider a mapping $\mathtt{to}_{\mathtt{Z}}$ from λ-terms to $\lambda_{\mathtt{Z}}$-terms. The following list of properties can be identified:

(C) The $\lambda_{\mathtt{Z}}$-reduction relation is confluent on $\lambda_{\mathtt{Z}}$-terms: If $u \ {}^{*}_{\lambda_{\mathtt{Z}}}\!\!\leftarrow t \rightarrow^{*}_{\lambda_{\mathtt{Z}}} v$, then there is t' such that $u \rightarrow^{*}_{\lambda_{\mathtt{Z}}} t' \ {}^{*}_{\lambda_{\mathtt{Z}}}\!\!\leftarrow v$.
(MC) The $\lambda_{\mathtt{Z}}$-reduction relation is confluent on $\lambda_{\mathtt{Z}}$-metaterms.
(PSN) The $\lambda_{\mathtt{Z}}$-reduction relation preserves β-strong normalisation: If the λ-term t is in $\mathcal{SN}_{\beta}$, then $\mathtt{to}_{\mathtt{Z}}(t)$ is in $\mathcal{SN}_{\lambda_{\mathtt{Z}}}$.
(SN) Strong normalisation holds for $\lambda_{\mathtt{Z}}$-typed terms: If the $\lambda_{\mathtt{Z}}$-term t is typed, then t is in $\mathcal{SN}_{\lambda_{\mathtt{Z}}}$.
(SIM) Any evaluation step in λ-calculus can be implemented by $\lambda_{\mathtt{Z}}$: If $t \rightarrow_{\beta} t'$, then $\mathtt{to}_{\mathtt{Z}}(t) \rightarrow^{*}_{\lambda_{\mathtt{Z}}} \mathtt{to}_{\mathtt{Z}}(t')$.
(FC) Full composition can be implemented by $\lambda_{\mathtt{Z}}$: The $\lambda_{\mathtt{Z}}$-term $t[x/u]$ $\lambda_{\mathtt{Z}}$-reduces to $t\{x/u\}$ for an appropriate notion of (meta)substitution on $\lambda_{\mathtt{Z}}$-terms.

In particular, (MC) implies (C) and (PSN) usually implies (SN).

[2] Some presentations replace the rule $y[x/u] \rightarrow y$ by the more general one $t[x/u] \rightarrow t$ ($x \notin \overline{t}$).

The result of Melliès appeared as a challenge to find a calculus having all the properties mentioned above. There are already several propositions in the literature giving (partial) answers to this challenge; they are summarised in the following table, where we just write one representative calculus for each line, even if there are currently many more references available in the literature (by lack of space we cannot cite all of them).

Calculus	C	MC	PSN	SN	SIM	FC
$\lambda_{\mathtt{x}}$ [44]	Yes	No	Yes	Yes	Yes	No
λ_{σ} [2]	Yes	No	No	No	Yes	Yes
$\lambda_{\sigma\Uparrow}$ [23]	Yes	Yes	No	No	Yes	Yes
λ_{ζ} [41]	Yes	Yes	Yes	Yes	No	No
λ_{ws} [14]	Yes	Yes	Yes	Yes	Yes	No
$\lambda\mathtt{lxr}$ [29]	Yes	?	Yes	Yes	Yes	Yes

In other words, there are many ways to avoid Melliès' counter-example in order to recover the PSN property. More precisely, one can forbid the substitution operators to cross lambda-abstractions [38, 18] or avoid composition of substitutions [6]. One can also impose a simple strategy on the calculus with ES to mimic exactly the calculus without ES. The first solution leads to *weak* lambda calculi, not able to express *strong* beta-equality (used for example in implementations of proof-assistants). The second solution is drastic when composition of substitutions is needed for implementations of HO unification [15] or functional abstract machines [24]. The last one does not take advantage of the notion of ES because they can be neither composed nor even delayed.

In order to cope with this problem David and Guillaume [14] defined a calculus with *labels* called λ_{ws}, which allows *controlled* composition of ES without losing PSN and SN. But the λ_{ws}-calculus has a complicated syntax and its named version [13] is even less intelligible. However, the strong normalisation proof for λ_{ws} given in [13] reveals a natural semantics for composition of ES via Linear Logic's proof-nets [19], suggesting that weakening (explicit erasure) and contraction (explicit duplication) can be added to the calculus without losing strong normalisation.

Explicit weakening and contraction are the starting points of the $\lambda\mathtt{lxr}$-calculus [29], which is in some sense a (complex) precursor of the $\lambda\mathtt{es}$-calculus that we present in this paper. However, while λ-syntax could be seen as a particular case of $\lambda\mathtt{es}$-syntax, a special encoding is needed to incorporate weakening and contraction operators to λ-terms in order to verify the so-called linearity constraints of $\lambda\mathtt{lxr}$. Moreover, the reduction system of $\lambda\mathtt{lxr}$ contains 6 equations and 19 rewriting rules, thus requiring an important amount of combinatorial reasoning. This is notably discouraging when one needs to check properties by cases on the reduction step; a reason why confluence on metaterms for $\lambda\mathtt{lxr}$ is just conjectured but not still proved.... Also, whereas $\lambda\mathtt{lxr}$ gives the evidence that explicit weakening and contraction are *sufficient* to verify all the properties one expects from a calculus with ES, there is no justified reason to think that they are also *necessary*.

We choose here to introduce the $\lambda\mathtt{es}$-calculus by using concise and simple syntax in named variable notation style (as in $\lambda\mathtt{x}$) in order to dissociate all the renaming details which are necessary to specify higher-order substitution on first-order terms (such as for example terms in de Bruijn notation). Even if this choice implies the use of

α-equivalence, we think that this presentation is more appropriate to focus on the fundamental computational properties of the calculus. Moreover, this can also be justified by the fact that it is now perfectly well-understood in the literature how to translate terms with named variables into equivalent terms in first-order notation. Another important choice made in this paper is the use of minimal equational reasoning (just one equation) to specify commutation of independent substitutions. This will turn out to be essential to obtain a *safe* notion of (full)composition which does not need the complex use of explicit operators for contraction and weakening. Also, simultaneous substitution (also called n-ary substitution), can be simply expressed within our framework.

We thus achieve the definition of a simple language being easy to understand, and enjoying a useful set of properties: confluence on metaterms (and thus on terms), simulation of one-step β-reduction, strong normalisation of typed terms, preservation of β-strong normalisation, simulation of one-step β-reduction and full composition. Moreover, these properties can be proved using very simple proof techniques while this is not the case for other calculi axiomatising commutation of substitutions. Thus for example, the calculus proposed in [45] specifies commutation of independent substitutions by a *non-terminating* rewriting system (instead of an equation), thus leading to complicated notions and proofs of its underlying normalisation properties.

The $\lambda\texttt{es}$-calculus admits a natural translation into Linear Logic's proof-nets, thus providing an alternative proof of strong normalisation. Also, a more implementation oriented calculus based on $\lambda\texttt{es}$ could be specified by means of de Bruijn notation and n-ary substitutions. These two last topics are however omitted in this paper because of lack of space, we refer the interested reader to [28].

The rest of the paper is organised as follows. Section 2 introduces syntax for $\Lambda\texttt{es}$-terms and appropriate notions of equivalence and reduction. In Section 3 we develop a proof of confluence for metaterms. Preservation of β-strong normalisation is studied and proved in Section 4. The typing system for $\lambda\texttt{es}$ is presented in Section 5 as well as the subject reduction property and the relation between typing derivations in $\lambda\texttt{es}$ and λ-calculus. Finally, strong normalisation based on PSN is proved in this same section.

We refer the reader to [28] for detailed proofs and to [9, 47] for standard notions from rewriting that we will use throughout the paper.

2 Syntax

A $\Lambda\texttt{es}$-term is inductively defined by a *variable* x, an *application* $t\ u$, an *abstraction* $\lambda x.t$ or a *substituted term* $t[x/u]$, when t and u are $\Lambda\texttt{es}$-terms. The syntactic object $[x/u]$, which is not a term itself, is called an *explicit substitution*.

The terms $\lambda x.t$ and $t[x/u]$ bind x in t. The sets of *free* and *bound* variables of a term t, denoted $\overline{t}$ and $\underline{t}$ respectively, can be defined as usual. Thus, the standard notion of α-conversion on higher-order terms is obtained so that one may assume, when *necessary*, that two bound variables have different names, and no variable is free and bound at the same time. Indeed, when using different symbols x and y to talk about two *nested* bound variables, as for example in the terms $(\lambda y.t)[x/u]$ and $t[x/u][y/v]$, we implicitly mean $x \neq y$. The use of the same name for bound variables appearing in *parallel/disjoint* positions, as for example in $t[x/u]\ v[x/u]$ or $(\lambda x.x)\ (\lambda x.x)$ is not problematic.

Besides α-conversion the following equations and reduction rules are considered.

Equations	**Reduction Rules**			
$t[x/u][y/v] =_{\mathtt{C}} t[y/v][x/u]$	$(\lambda x.t)\ u$	$\rightarrow_{\mathtt{B}}$	$t[x/u]$	
$(y \notin \overline{u}\ \&\ x \notin \overline{v})$	**The (sub)set of rules** $\mathtt{s}$**:**			
	$x[x/u]$	$\rightarrow_{\mathtt{Var}}$	u	
	$t[x/u]$	$\rightarrow_{\mathtt{Gc}}$	t	$(x \notin \overline{t})$
	$(t\ u)[x/v]$	$\rightarrow_{\mathtt{App}_1}$	$t[x/v]\ u[x/v]$	$(x \in \overline{t}\ \&\ x \in \overline{u})$
	$(t\ u)[x/v]$	$\rightarrow_{\mathtt{App}_2}$	$t\ u[x/v]$	$(x \notin \overline{t}\ \&\ x \in \overline{u})$
	$(t\ u)[x/v]$	$\rightarrow_{\mathtt{App}_3}$	$t[x/v]\ u$	$(x \in \overline{t}\ \&\ x \notin \overline{u})$
	$(\lambda y.t)[x/v]$	$\rightarrow_{\mathtt{Lamb}}$	$\lambda y.t[x/v]$	
	$t[x/u][y/v]$	$\rightarrow_{\mathtt{Comp}_1}$	$t[y/v][x/u[y/v]]$	$(y \in \overline{u}\ \&\ y \in \overline{t})$
	$t[x/u][y/v]$	$\rightarrow_{\mathtt{Comp}_2}$	$t[x/u[y/v]]$	$(y \in \overline{u}\ \&\ y \notin \overline{t})$

It is appropriate to point out here that α-conversion is necessary in order to avoid capture of variables. Thus for example the left-hand side of the Lamb-rule $(\lambda y.t)[x/v]$ implicitly assumes $y \neq x$ and $y \notin \overline{v}$. See also Sections 4.2 and 6 for a a discussion about the minimality of the subset s w.r.t its number of rules.

The *higher-order rewriting system* containing the rules $\{\mathtt{B}\} \cup \mathtt{s}$ is called Bs. The *equivalence relation* generated by the conversions $\mathtt{E_s} = \{\alpha, \mathtt{C}\}$ is denoted by $=_{\mathtt{E_s}}$. The *reduction relation* generated by the *rewriting rules* s (resp. Bs) *modulo the equivalence relation* $=_{\mathtt{E_s}}$ is denoted by $\rightarrow_{\mathtt{es}}$ (resp. $\rightarrow_{\lambda\mathtt{es}}$), the e means equational and the s substitution. More precisely,

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

The notation $\rightarrow^*_{\lambda\mathtt{es}}$ (resp. $\rightarrow^+_{\lambda\mathtt{es}}$) is used for the reflexive and transitive (resp. transitive) closure of $\rightarrow_{\lambda\mathtt{es}}$.

Remark that any simultaneous (n-ary) substitution can now be thought as a sequence of consecutive *independent* unary substitutions representing the same mapping. Thus for example $[x/u, y/v]$ can be expressed as $[x/u][y/v]$ (or $[y/v][x/u]$) where $y \notin \overline{u}$ and $x \notin \overline{v}$. The use of the equation C to make a list of independent substitutions behave like a simultaneous one is essential. We leave to the reader the verification that composition of simultaneous substitution can be expressed within our λes-reduction relation.

The equivalence relation preserves free variables and the reduction relation either preserves or decreases them. Thus, $t \rightarrow_{\lambda\mathtt{es}} u$ implies $\overline{u} \subseteq \overline{t}$.

Also, the (sub)calculus es, which is intended to implement (meta-level) substitution, can be shown to be terminating by associating to each Λes-term t a measure which does not change by $\mathtt{E_s}$ but strictly decreases by $\rightarrow_s$ (details can be found in [28]).

We now address the property of full composition. For that, we extend the standard notion of (meta-level)substitution on λ-terms given in the introduction to all the Λes-terms by adding the new case $t[y/u]\{x/v\} := t\{x/v\}[y/u\{x/v\}]$, where we implicitly mean $x \neq y\ \&\ y \notin \overline{v}$. Remark that $t\{x/u\} = t$ if $x \notin \overline{t}$, thus we can prove:

Lemma 1 (Full Composition). *Let t and u be Λes-terms. Then* $t[x/u] \rightarrow^*_{\lambda\mathtt{es}} t\{x/u\}$.

We now establish basic connections between λ and $\lambda\mathtt{es}$-reduction. As expected, β-reduction can be implemented by the more atomic notion of $\lambda\mathtt{es}$-reduction while this one can be projected into β.

Lemma 2 (Simulating β-reduction). *Let t be a λ-term s.t. $t \rightarrow_{\beta} t'$. Then $t \rightarrow^{+}_{\lambda\mathtt{es}} t'$.*

Proof. By induction on β-reduction using Lemma 1.

$\Lambda\mathtt{es}$-terms are encoded into λ-terms as follows: $\mathtt{L}(x) := x$, $\mathtt{L}(\lambda x.t) := \lambda x.\mathtt{L}(t)$, $\mathtt{L}(t\ u) := \mathtt{L}(t)\ \mathtt{L}(u)$ and $\mathtt{L}(t[x/u]) := \mathtt{L}(t)\{x/\mathtt{L}(u)\}$. Thus, projection is obtained:

Lemma 3 (Projecting into β-reduction). *If $t \rightarrow_{\lambda\mathtt{es}} u$, then $\mathtt{L}(t) \rightarrow^{*}_{\beta} \mathtt{L}(u)$.*

Proof. First prove that $t =_{\mathtt{E_s}} u$ implies $\mathtt{L}(t) = \mathtt{L}(u)$ by the well-known substitution lemma [4] of λ-calculus. Remark that $t \rightarrow_{\mathtt{s}} u$ trivially implies $\mathtt{L}(t) = \mathtt{L}(u)$. Finally, prove that $t \rightarrow_{\mathtt{B}} u$ implies $\mathtt{L}(t) \rightarrow^{*}_{\beta} \mathtt{L}(u)$ by induction on the reduction step $t \rightarrow_{\mathtt{B}} u$.

3 Confluence on metaterms

Metaterms are terms containing *metavariables* denoting *incomplete* programs/proofs in a higher-order unification framework [25]. Metavariables should come with a minimal amount of information to guarantee that some basic operations such as instantiation (replacement of metavariables by metaterms) are sound in a typing context. However, known formalisms in the literature for the specification of higher-order metaterms, such as Combinatory Reduction Systems (CRS) [30] or Expression Reduction Systems (ERS) [26], do not allow, at least in a simpler way, to specify the precise set of free variables which is expected from a (sound)instantiation. Thus for example, a CRS metaterm like $M(x, y)$ specifies that x and y *may* occur in the instantiation of M, but M can also be further instantiated by any other term not containing x and y at all. Another example is given by the (raw) ERS metaterm $t = \lambda y.y\ \mathbb{X}\ (\lambda z.\mathbb{X})$ because the instantiation of $\mathbb{X}$ by a term containing a free occurrence of z would be unsound (see [41, 15, 17] for details).

We thus propose to specify incomplete proofs as follows. We consider a countable set of *raw* metavariables $\mathbb{X}, \mathbb{Y}, \ldots$ associated to sets of variables $\Gamma, \Delta, \ldots$, thus yielding *decorated* metavariables denoted by $\mathbb{X}_{\Gamma}, \mathbb{Y}_{\Delta}$, etc. This decoration says nothing about the *structure* of the incomplete proof itself but is sufficient to guarantee that different occurrences of the same metavariable inside a metaterm are never instantiated by different metaterms.

The grammar for $\Lambda\mathtt{es}$-terms is extended to generate $\Lambda\mathtt{es}$-metaterms as follows:

$$t ::= x \mid \mathbb{X}_{\Delta} \mid t\ t \mid \lambda x.t \mid t[x/t]$$

We extend the notion of *free variables* to *metaterms* by $\overline{X_{\Delta}} = \Delta$.

Reduction on metaterms must be understood in the same way reduction on terms: the $\lambda\mathtt{es}$-relation is generated by the $\mathtt{Bs}$-relation on $\mathtt{E_s}$-equivalence classes of *metaterms*.

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

It is well-known that confluence on metaterms fails for calculi *without* composition for ES as for example the following critical pair in $\lambda\mathtt{x}$ shows

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

Indeed, while this diagram can be closed in $\lambda\mathtt{x}$ for terms *without metavariables* [10], there is no way to find a common reduct between s and s' whenever t is (or contains) metavariables: no $\lambda\mathtt{x}$-reduction rule is able to mimic composition on raw or decorated metavariables. This can be fortunately recovered in the case of the $\lambda\mathtt{es}$-calculus.

3.1 The confluence proof

This section develops a confluence proof for reduction on $\lambda\mathtt{es}$-metaterms based on Tait and Martin-Löf's technique: define a simultaneous reduction relation denoted $\Rrightarrow_{\mathtt{es}}$; prove that $\Rrightarrow^*_{\mathtt{es}}$ and $\rightarrow^*_{\mathtt{es}}$ are the same relation; show that $\Rrightarrow^*_{\mathtt{es}}$ is confluent; and finally conclude. While many steps in this proof are similar to those appearing in other proofs of confluence for the λ-calculus, some special considerations are to be used here in order to accommodate correctly the substitution calculus as well as the equational part of our notion of reduction (see in particular Lemma 6).

A first interesting property of the system $\mathtt{es}$ is that it can be used as a function on $\mathtt{E_s}$-equivalence classes:

Lemma 4. *The* $\mathtt{es}$*-normal forms of metaterms are unique modulo* $\mathtt{E_s}$ *so that* $t =_{\mathtt{E_s}} u$ *implies* $\mathtt{es}(t) =_{\mathtt{E_s}} \mathtt{es}(u)$.

The simultaneous reduction relation $\Rrightarrow_{\mathtt{es}}$ on $\mathtt{es}$-normal forms is now defined in terms of a simpler relation $\Rrightarrow$ working on $\mathtt{E_s}$-equivalence classes.

Definition 1 (The relations $\Rrightarrow$ and $\Rrightarrow_{\mathtt{es}}$). *Simultaneous reduction is defined on metaterms in* $\mathtt{es}$*-normal form as follows:* $t \Rrightarrow_{\mathtt{es}} t'$ *iff* $\exists\, u, u'$ *s.t.* $t =_{\mathtt{E_s}} u \Rrightarrow u' =_{\mathtt{E_s}} t'$*, where*

- $x \Rrightarrow x$
- *If* $t \Rrightarrow t'$*, then* $\lambda x.t \Rrightarrow \lambda x.t'$
- *If* $t \Rrightarrow t'$ *and* $u \Rrightarrow u'$*, then* $t\ u \Rrightarrow t'\ u'$
- *If* $t \Rrightarrow t'$ *and* $u \Rrightarrow u'$*, then* $(\lambda x.t)\ u \Rrightarrow \mathtt{es}(t'[x/u'])$
- *If* $u_i \Rrightarrow u'_i$ *and* $x_i \notin \overline{u_j}$ *for all* $i, j \in [1, n]$*, then* $\mathbb{X}_\Delta[x_1/u_1]\ldots[x_n/u_n] \Rrightarrow \mathbb{X}_\Delta[x_1/u'_1]\ldots[x_n/u'_n]$

The simultaneous relation is stable in the following sense.

Lemma 5. *If* $t \Rrightarrow_{\mathtt{es}} t'$ *and* $u \Rrightarrow_{\mathtt{es}} u'$*, then* $\mathtt{es}(t[x/u]) \Rrightarrow_{\mathtt{es}} \mathtt{es}(t'[x/u'])$.

It can be now shown that the relation $\Rrightarrow_{\mathtt{es}}$ has the *diamond property*.

Lemma 6. *If* $t_1\ {}_{\mathtt{es}}\!\Lleftarrow t \Rrightarrow_{\mathtt{es}} t_2$*, then* $\exists t_3$ *s.t.* $t_1 \Rrightarrow_{\mathtt{es}} t_3\ {}_{\mathtt{es}}\!\Lleftarrow t_2$.

Proof. 1. First prove that $t \Lleftarrow u =_{\mathtt{E_s}} u'$ implies $t =_{\mathtt{E_s}} t' \Lleftarrow u'$ for some t' by induction on $t \Lleftarrow u$. Thus conclude that $v\ {}_{\mathtt{es}}\!\Lleftarrow v' =_{\mathtt{E_s}} u'$ implies $v =_{\mathtt{E_s}} t' \Lleftarrow u'$ for some t'.

2. Prove that $t_1 \Lleftarrow t \Rrightarrow t_2$ implies $t_1 \Rrightarrow_{\mathtt{es}} t_3 \ {}_{\mathtt{es}}\!\Lleftarrow t_2$ for some t_3 by induction on $\Rrightarrow$ using Lemma 5.
3. Finally prove the diamond property as follows. Let $t_1 \ {}_{\mathtt{es}}\!\Lleftarrow t =_{\mathtt{E_s}} u \Rrightarrow u' =_{\mathtt{E_s}} t_2$. By point (1) there is u_1 such that $t_1 =_{\mathtt{E_s}} u_1 \Lleftarrow u$ and by point (2) there is t_3 such that $u_1 \Rrightarrow_{\mathtt{es}} t_3 \ {}_{\mathtt{es}}\!\Lleftarrow u'$. Conclude $t_1 \Rrightarrow_{\mathtt{es}} t_3 \ {}_{\mathtt{es}}\!\Lleftarrow t_2$.

We thus obtain the main result of this section:

Corollary 1. *The reduction relation $\rightarrow^*_{\mathtt{es}}$ is confluent.*

Proof. The relation $\Rrightarrow^*_{\mathtt{es}}$ enjoys the diamond property (Lemma 6) so that it turns out to be confluent [9]. Since $\Rrightarrow^*_{\mathtt{es}}$ and $\rightarrow^*_{\lambda\mathtt{es}}$ can be shown (using Lemmas 4 and 5) to be the same relation, then conclude that $\rightarrow^*_{\lambda\mathtt{es}}$ is also confluent.

Although this confluence result guarantees that all the critical pairs in $\lambda\mathtt{es}$ can be closed, let us analyse a concrete example being the source of interesting diverging diagrams in calculi with ES (c.f. Section 1), giving by the following case:

$$\begin{array}{ccccc} s_3 & {}^{*}_{\lambda\mathtt{es}}\!\leftarrow & s_1 & \rightarrow_{\mathtt{B}} & s_2 \\ ? & & ((\lambda x.t)\ u)[y/v] & & t[x/u][y/v] \end{array}$$

The metaterm s_3 as well as the one used to close the diagram can be determined by the following four different cases:

$y \in \overline{t}$	$y \in \overline{u}$	s_3	Close the diagram by
Yes	Yes	$t[y/v][x/u[y/v]]$	$s_3 \ {}_{\mathtt{Comp_1}}\!\leftarrow s_2$
Yes	No	$t[y/v][x/u]$	$s_3 =_{\mathtt{E_s}} s_2$
No	Yes	$t[x/u[y/v]]$	$s_3 \ {}_{\mathtt{Comp_2}}\!\leftarrow s_2$
No	No	$(\lambda x.t)\ u$	$s_3 \rightarrow_{\mathtt{B}} t[x/u] \ {}_{\mathtt{Gc}}\!\leftarrow s_2$

4 Preservation of β-strong normalisation

Preservation of β-strong normalisation (PSN) in calculi with ES received a lot of attention (see for example [2, 6, 10, 32]), starting from an unexpected result given by Melliès [40] 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 for $\lambda\sigma$ [2] and $\lambda\sigma_{\Uparrow}$ [23].

Since then, different notions of safe composition where introduced, even if PSN becomes more difficult to prove ([8, 14, 1, 29, 31]). 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 $\lambda_{\mathtt{Z}}$ if every subterm v appearing in some substituted subterm $u[x/v]$ of t is $\lambda_{\mathtt{Z}}$-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 both $\lambda\mathtt{es}$-strongly normalising, but its $\mathtt{Comp_2}$-reduct $x[x/(y\ y)[y/\lambda w.w\ w]]$ is not.

This section proves that $\lambda\mathtt{es}$ preserves β-strong normalisation. For that, we use a simulation proof technique based on the following steps. We first define a calculus $\lambda\mathtt{esw}$ (Section 4.1). We then give a translation $\mathtt{K}$ from $\Lambda\mathtt{es}$-terms (and thus also from λ-terms) into $\lambda\mathtt{esw}$ s.t. $t \in \mathcal{SN}_{\beta}$ implies $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$ (Corollary 4) and $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$ implies $t \in \mathcal{SN}_{\lambda\mathtt{es}}$ (Corollary 2).

4.1 The λesw-calculus

A Λesw-term is inductively defined by x, $t\ u$, $\lambda x.t$, $t[x/u]$ or $\mathcal{W}_x(t)$ (an *explicit weakening*). We extend the notion of free variables to explicit weakenings by adding the case $\overline{\mathcal{W}_x(t)} = \{x\} \cup \overline{t}$. The notion of *strict* term will be essential: every subterm $\lambda x.t$ and $t[x/u]$ is such that $x \in \overline{t}$ and every subterm $\mathcal{W}_x(t)$ is such that $x \notin \overline{t}$.

Besides equations and rules in λes, those in the following table are also considered.

Additional Equations	**Additional Reduction Rules**
$\mathcal{W}_x(\mathcal{W}_y(t)) =_{\mathtt{WC}} \quad \mathcal{W}_y(\mathcal{W}_x(t))$	$\mathcal{W}_x(t)[x/u] \to \mathcal{W}_{\overline{u}\setminus\overline{t}}(t)$
$\mathcal{W}_y(t)[x/u] =_{\mathtt{Weak1}} \mathcal{W}_y(t[x/u]) \ (x \neq y \ \&\ y \notin \overline{u})$	$\mathcal{W}_y(t)\ u \to t\ u \quad (y \in \overline{u})$
$\mathcal{W}_y(\lambda x.t) =_{\mathtt{WAbs}} \lambda x.\mathcal{W}_y(t) \quad (x \neq y)$	$\mathcal{W}_y(t)\ u \to \mathcal{W}_y(t\ u) \quad (y \notin \overline{u})$
	$t\ \mathcal{W}_y(u) \to t\ u \quad (y \in \overline{t})$
	$t\ \mathcal{W}_y(u) \to \mathcal{W}_y(t\ u) \quad (y \notin \overline{t})$
	$\mathcal{W}_y(t)[x/u] \to t[x/u] \quad (y \in \overline{u})$
	$t[x/\mathcal{W}_y(u)] \to \mathcal{W}_y(t[x/u]) \ (y \notin \overline{t})$
	$t[x/\mathcal{W}_y(u)] \to t[x/u] \quad (y \in \overline{t})$

Given a set of variables $\Gamma = \{x_1, \ldots, x_n\}$, the use of the abbreviation $\mathcal{W}_\Gamma(t)$ for $\mathcal{W}_{x_1}(\ldots \mathcal{W}_{x_n}(t))$ in the first reduction rule is justified by the equation WC. In the particular case $\Gamma = \emptyset$, we define $\mathcal{W}_\emptyset(t) = t$. It is suitable again to recall that we work modulo α-conversion. Thus for example the terms $\mathcal{W}_y(\lambda x.t)$ and $t[x/\mathcal{W}_y(u)]$ have to be always understood as $x \neq y$. However, this is not the case for example for $\lambda x.\mathcal{W}_y(t)$ or $\mathcal{W}_y(t)[x/u]$ where the variables x and y may be equal or different, that's the reason to explicitly add the side-condition $x \neq y$ in some of the previous equations and rules.

The rewriting system containing all the reduction rules in the previous table plus those in system s is called sw. The notation Bsw is used for the system $\{\mathtt{B}\} \cup \mathtt{sw}$. The equivalence relation generated by all the equations in the previous table plus those in $\mathtt{E_s}$ is denoted by $=_{\mathtt{E_{sw}}}$. The relation generated by the reduction rules sw (resp. Bsw) modulo the equivalence relation $=_{\mathtt{E_{sw}}}$ is denoted by $\to_{\mathtt{esw}}$ (resp. $\to_{\lambda\mathtt{esw}}$). More precisely,

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

From now on, we only work with strict terms, a choice that is justified by the fact that λesw-reduction relation preserves strict terms.

In order to infer normalisation of λes from that of λesw, a relation between both notions of reduction is needed. For that, a translation K from Λes-terms (and thus also from λ-terms) to (strict) Λesw-terms is defined as follows:

$$\begin{array}{lll|lll} \mathtt{K}(x) & := x & & \mathtt{K}(u\ v) & := \mathtt{K}(u)\ \mathtt{K}(v) & \\ \mathtt{K}(\lambda x.t) & := \lambda x.\mathtt{K}(t) & \text{If } x \in \overline{t} & \mathtt{K}(\lambda x.t) & := \lambda x.\mathcal{W}_x(\mathtt{K}(t)) & \text{If } x \notin \overline{t} \\ \mathtt{K}(u[x/v]) & := \mathtt{K}(u)[x/\mathtt{K}(v)] & \text{If } x \in \overline{t} & \mathtt{K}(u[x/v]) & := \mathcal{W}_x(\mathtt{K}(u))[x/\mathtt{K}(v)] & \text{If } x \notin \overline{t} \end{array}$$

Remark that $\overline{\mathtt{K}(t)} = \overline{t}$. Also, λesw-reduction can be used to push out useless weakening constructors as follows:

Lemma 7. *If* $u \to_{\lambda\mathtt{es}} v$, *then* $\mathtt{K}(u) \to^+_{\lambda\mathtt{esw}} \mathcal{W}_{\overline{u}\setminus\overline{v}}(\mathtt{K}(v))$.

Proof. The proof is by induction on $\to_{\lambda\mathtt{es}}$ and it accurately puts in evidence the fact that `Weak1` and `WAbs` are needed as equations and not as rewriting rules.

The previous lemma allows us to conclude with the following preservation result:

Corollary 2. *If* $\mathtt{K}(t) \in \mathcal{SN}_{\lambda\mathtt{esw}}$, *then* $t \in \mathcal{SN}_{\lambda\mathtt{es}}$.

4.2 The Λ_I-calculus

The Λ_I-calculus is another intermediate language used as technical tool to prove PSN. The set of Λ_I-terms [30] is defined by the grammar:

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

We consider the extended notions of free variables and (meta)level substitution on Λ_I-terms. We restrict again the syntax to *strict* terms (every subterm $\lambda x.M$ satisfies $x \in \overline{M}$). The following two reduction rules will be used:

$$\boxed{\begin{array}{l} (\lambda x.M)\ N \to_\beta M\{x/N\} \\ [M, N]\ L \quad \to_\pi [M\ L, N] \end{array}}$$

Strict Λ_I-terms turn out to be stable by reduction since they do not lose free variables during reduction.

A binary relation (and not a function) $\mathcal{I}$ is used to relate $\lambda\mathtt{esw}$ and Λ_I-terms, this because $\Lambda\mathtt{esw}$-terms are translated into Λ_I-syntax by adding some *garbage* information which is not uniquely determined. Thus, each $\Lambda\mathtt{esw}$-term can be projected into different Λ_I-terms, and this will be essential in the simulation property (Theorem 1).

Definition 2. *The relation* $\mathcal{I}$ *between* strict $\Lambda\mathtt{esw}$*-terms and* strict Λ_I*-terms is inductively given by the following rules:*

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

$$\frac{t\ \mathcal{I}\ T\ \&\ M \textit{ strict}}{t\ \mathcal{I}\ [T, M]} \qquad \frac{t\ \mathcal{I}\ T\ \&\ x \in \overline{T}}{\mathcal{W}_x(t)\ \mathcal{I}\ T}$$

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

Lemma 8. *Let* $t\ \mathcal{I}\ M$. *Then* $\overline{t} \subseteq \overline{M}$, $M \in \Lambda_I$ *and* $x \notin \overline{t}\ \&\ N \in \Lambda_I$ *implies* $t\ \mathcal{I}\ M\{x/N\}$.

Remark however that $t\ \mathcal{I}\ M$ implies $\overline{t} \subseteq \overline{M}$ only on *strict* terms. This can be seen as a proof technical argument to exclude from our calculus rewriting rules not preserving strict terms like

$$\begin{array}{lll} (\mathtt{App}) & (t\ u)[x/v] & \to t[x/v]\ u[x/v] \\ (\mathtt{Comp}) & t[x/u][y/v] & \to t[y/v][x/u[y/v]] \quad (y \in \overline{u}) \end{array}$$

Reduction in $\lambda\texttt{esw}$ can be related to reduction in Λ_I by means of the following simulation property (proved by induction on the reduction/equivalence step).

Theorem 1. *Let $s \in \Lambda\texttt{esw}$ and $S \in \Lambda_I$.*

1. *If $s\ \mathcal{I}\ S$ and $s =_{\texttt{E}_{\texttt{sw}}} t$, then $t\ \mathcal{I}\ S$.*
2. *If $s\ \mathcal{I}\ S$ and $s \rightarrow_{\texttt{sw}} t$, then $t\ \mathcal{I}\ S$.*
3. *If $s\ \mathcal{I}\ S$ and $s \rightarrow_{\texttt{B}} t$, then there is $T \in \Lambda_I$ s.t. $t\ \mathcal{I}\ T$ and $S \rightarrow^{+}_{\beta\pi} T$.*

The second preservation result can be now stated as follows:

Corollary 3. *If $s\ \mathcal{I}\ S$ and $S \in \mathcal{SN}_{\beta\pi}$, then $s \in \mathcal{SN}_{\lambda\texttt{esw}}$.*

Proof. Suppose $s \notin \mathcal{SN}_{\lambda\texttt{esw}}$. As $\rightarrow_{\texttt{esw}}$ can easily be show to be well-founded (see [28] for details), then an infinite $\lambda\texttt{esw}$-reduction sequence starting at s is necessarily projected by the previous Theorem into an infinite $\beta\pi$-reduction sequence starting at S. This leads to a contradiction with the hypothesis.

4.3 Solving the puzzle

All the parts of the puzzle together give a PSN argument for $\lambda\texttt{es}$. The starting point is the following encoding from λ to Λ_I-terms:

$$\begin{array}{llll}
\texttt{I}(x) := x & & \texttt{I}(\lambda x.t) := \lambda x.\texttt{I}(t) & x \in \overline{t} \\
\texttt{I}(t\ u) := \texttt{I}(t)\ \texttt{I}(u) & & \texttt{I}(\lambda x.t) := \lambda x.[\texttt{I}(t), x] & x \notin \overline{t}
\end{array}$$

Now, starting from a λ-term u, which is also a $\Lambda\texttt{es}$-term, one computes its $\texttt{K}$-image - a $\lambda\texttt{esw}$-term - so that some Λ_I-term will be in $\mathcal{I}$-relation with it. More precisely, a straightforward induction on u gives:

Theorem 2. *For any λ-term u, $\texttt{K}(u)\ \mathcal{I}\ \texttt{I}(u)$.*

Preservation of β-strong-normalisation, which is one of the main results of the paper, can be finally stated:

Corollary 4 (PSN). *If $t \in \mathcal{SN}_{\beta}$, then $t \in \mathcal{SN}_{\lambda\texttt{es}}$.*

Proof. If $t \in \mathcal{SN}_{\beta}$, then $\texttt{I}(t) \in WN_{\beta\pi}$ [34] and thus $\texttt{I}(t) \in SN_{\beta\pi}$ [42]. As $\texttt{K}(t)\ \mathcal{I}\ \texttt{I}(t)$ by Theorem 2, then $\texttt{K}(t) \in \mathcal{SN}_{\lambda\texttt{esw}}$ by Corollary 3 so that $t \in \mathcal{SN}_{\lambda\texttt{es}}$ by Corollary 2.

5 The typed $\lambda\texttt{es}$-calculus

Simply types are built over a countable set of atomic symbols (base types) and the type constructor $\rightarrow$ (functional types). 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 $A = B$. The *union of compatible contexts* is written $\Gamma \uplus \Delta$. Thus for example $(x : A, y : B) \uplus (x : A, z : C) = (x : A, y : B, z : C)$. The following properties on compatible environments will be used:

Lemma 9.

1. *If $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$, then $\Gamma \uplus \Delta \subseteq \Gamma' \uplus \Delta'$.*
2. *$\Gamma \uplus (\Delta \uplus \Pi) = (\Gamma \uplus \Delta) \uplus \Pi$.*

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, written $\Gamma \vdash_{\lambda \mathtt{es}} t : A$, can be obtained by application of the (*multiplicative*) rules in the following table.

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

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

The axiom rule types a variable in a minimal environment but variables not appearing free may be introduced by binder symbols by means of the rules abs and subs. Thus for example starting from the derivable typing judgement $x : B \vdash x : B$ one can derive judgements like $\vdash \lambda x.x : B \to B$ or $x : B \vdash \lambda z.x : A \to B$. Remark that when $\Gamma \uplus \Delta$ appears in the conclusion of some rule, then by definition, Γ and Δ are compatible.

The typing rules for $\lambda\mathtt{es}$ ensure that every environment Γ contains *exactly* the set of free variables of the term t. Thus, $\Gamma \vdash_{\lambda \mathtt{es}} t : A$ implies $\Gamma = \overline{t}$.

The typed calculus enjoys *local* subject reduction in the sense that no meta-theorem stating *weakening* or *thinning* is needed to show preservation of types.

Lemma 10 (Subject Reduction). *Let $\Gamma \vdash_{\lambda \mathtt{es}} s : A$. Then $s =_{\mathtt{E_s}} s'$ implies $\Gamma \vdash_{\lambda \mathtt{es}} s' : A$ and $s \to_{\lambda \mathtt{es}} s'$ implies $\Pi' \vdash_{\lambda \mathtt{es}} s' : A$ for some $\Pi' \subseteq \Pi$.*

The connexion between *typed* derivations in λ-calculus (written $\vdash_\lambda$) and *typed* derivations in $\lambda\mathtt{es}$-calculus is stated as follows, where $\Gamma|_{\mathcal{S}}$ denotes the environment Γ restricted to the set of variables $\mathcal{S}$.

Lemma 11. *If $\Gamma \vdash_\lambda t : A$, then $\Gamma|_{\overline{t}} \vdash_{\lambda \mathtt{es}} t : A$ and if $\Gamma \vdash_{\lambda \mathtt{es}} t : A$, then $\Gamma \vdash_\lambda \mathtt{L}(t) : A$.*

We now prove strong-normalisation for $\lambda\mathtt{es}$-typed terms by using PSN. Another proof of strong-normalisation based on a translation of typed $\lambda\mathtt{es}$-terms into Linear Logic's proof-nets is also developed in [28].

Theorem 3 (Strong Normalisation). *Every typable $\Lambda\mathtt{es}$-term M is in $SN_{\lambda \mathtt{es}}$.*

Proof. First define a translation $\mathtt{C}$ from $\lambda\mathtt{es}$ to λ as follows: $\mathtt{C}(x) := x$, $\mathtt{C}(t\ u) := \mathtt{C}(t)\ \mathtt{C}(u)$, $\mathtt{C}(\lambda x.t) := \lambda x.\mathtt{C}(t)$ and $\mathtt{C}(t[x/u]) := (\lambda x.\mathtt{C}(t))\ \mathtt{C}(u)$. 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 $\mathtt{C}(t) \to^*_{\lambda \mathtt{es}} t$. Also, when t is typable in $\lambda\mathtt{es}$, then also $\mathtt{C}(t)$ is typable in $\lambda\mathtt{es}$ (just change the use of subs by abs followed by app). By Lemma 11 the term $\mathtt{L}(\mathtt{C}(t)) = \mathtt{C}(t)$ is also typable in simply typed λ-calculus and thus $\mathtt{C}(t) \in SN_\beta$ [5]. We get $\mathtt{C}(t) \in SN_{\lambda \mathtt{es}}$ by Corollary 4 so that $t \in SN_{\lambda \mathtt{es}}$.

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 (de Bruijn) calculi [43, 3].

6 Conclusion

In this paper we survey some properties concerning ES calculi and we describe work done in the domain during these last 15 years. We propose simple syntax and simple equations and rewriting rules to model a formalism enjoying 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 might be proved directly without using translations of $\lambda\mathtt{es}$ to other formalisms. We leave this for future work.

Another interesting issue is the extension of Pure Type Systems (PTS) with ES in order to improve the understanding of logical systems used in theorem-provers. Work done in this direction is based on sequent calculi [33] or natural deduction [41]. The main contribution of $\lambda\mathtt{es}$ w.r.t the formalisms previously mentioned would be our *safe* notion of composition.

It is also legitimate to ask whether $\lambda\mathtt{es}$ is minimal w.r.t. the number of rewriting rules. Indeed, it is really tempted to gather the rules $\{\mathtt{App}_1, \mathtt{App}_2, \mathtt{App}_3\}$ (resp. $\{\mathtt{Comp}_1, \mathtt{Comp}_2\}$) into the single rule $\mathtt{App}$ for application (resp. $\mathtt{Comp}$ for composition) given just after Lemma 8. While this change seems to be sound w.r.t. the properties of the calculus [3], the translation of $\Lambda\mathtt{es}$-terms into Λ_I-terms (c.f. Section 4.2), respectively into proof-nets (c.f. [28]), does not work anymore. We thus leave this question as an open problem. Note however 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 $\mathtt{App}$ followed by $\mathtt{Gc}$.

As far as implementation is concerned, it would be preferable from a practical point of view to avoid the systematic use of the equivalence classes generated by the axioms α and $\mathtt{C}$. In other words, it would be more efficient to work with a pure rewriting system (without equations) verifying the same properties than $\lambda\mathtt{es}$. We believe that simultaneous substitutions will be needed to avoid axiom $\mathtt{C}$ while some technology like de Bruijn notation will be needed to avoid axiom α (as in the $\lambda_{\sigma_\Uparrow}$-calculus). We leave this topic for future investigations, but we refer the interested reader to [28] for a concrete proposition of such a calculus.

Acknowledgements

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

References

[1] A. Arbiser, E. Bonelli, and A. Ríos. Perpetuality in a lambda calculus with explicit substitutions and composition. *WAIT* 2000.

[3] While the weaker rule for composition given by $t[x/u][y/v] \to t[x/u[y/v]]$ $(y \notin \overline{t})$, is well-known [7] to affect strong normalisation and preservation of β-strong normalisation.

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

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

[4] H. Barendregt. The Lambda Calculus: Its Syntax and Semantics. North-Holland, 1984.

[5] H. Barendregt. Lambda calculus with types. In *Handbook of Logic in Computer Science*, volume 2, 1992.

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

[7] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. *TCS*, 6(5):699–722, 1999.

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

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

[10] 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* 1995.

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

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

[13] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. In *LNCS* 1784, *FOSSACS* 2000.

[14] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitution. *MSCS*, 11:169–206, 2001.

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

[16] R. Dyckhoff and C. Urban. Strong normalisation of Herbelin's explicit substitution calculus with substitution propagation. WESTAPP 2001.

[17] F. L. C. de Moura and M. Ayala-Rincón and F. Kamareddine. Higher-Order Unification: A structural relation between Huet's method and the one based on explicit substitutions. Journal of Applied Logic, 6(1):72-108, 2008.

[18] J. Forest. A weak calculus with explicit operators for pattern matching and substitution. In *LNCS* 2378, *RTA* 2002.

[19] J.-Y. Girard. Linear logic. *TCS*, 50(1):1–101, 1987.

[20] J. Goubault-Larrecq. Conjunctive types and SKInT. In *LNCS* 1657, *Types for Proofs and Programs*, 1999.

[21] 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.

[22] H. Herbelin. A λ-calculus structure isomorphic to sequent calculus structure. In *LNCS* 933, *CSL* 1994.

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

[24] T. Hardin, L. Maranget, and B. Pagano. Functional back-ends within the lambda-sigma calculus. In *ICFP* 1996.

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

[26] Zurab Khasidashvili. Expression reduction systems. In *Proceedings of IN Vekua Institute of Applied Mathematics*, volume 36, Tbilisi, 1990.

[27] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions. In *LNCS* 1103, *RTA* 1996.

[28] D. Kesner. The theory of calculi with explicit substitutions revisited, 2006. Available as `http://hal.archives-ouvertes.fr/hal-00111285/`.

[29] D. Kesner and S. Lengrand. Extending the explicit substitution paradigm. In *LNCS* 3467, *RTA* 2005.

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

[31] Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Uniform Normalization Beyond Orthogonality. In *LNCS* 2051, *RTA* 2001.

[32] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *LNCS* 982, *PLILP* 1995.

[33] S. Lengrand, R. Dyckhoff, and J. McKinna. A sequent calculus for type theory. In *LNCS* 4207, *CSL* 2006.

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

[35] P. Lescanne. From λ_σ to λ_υ, a journey through calculi of explicit substitutions. In *POPL* 1994.

[36] R. Lins. A new formula for the execution of categorical combinators. In *LNCS* 230, *CADE* 1986.

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

[38] J.-J. Lévy and L. Maranget. Explicit substitutions and programming languages. In *LNCS* 1738, *FSTTCS* 1999.

[39] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn levels. In *LNCS* 914, *RTA* 1995.

[40] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In *LNCS* 902, *TLCA* 1995.

[41] C. Muñoz. Un calcul de substitutions pour la représentation de preuves partielles en théorie de types. PhD thesis, Université Paris 7, 1997.

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

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

[44] K. Rose. Explicit cyclic substitutions. In *LNCS* 656, *RTA* 1992.

[45] T. Sakurai. Strong normalizability of calculus of explicit substitutions with composition. Available on `http://www.math.s.chiba-u.ac.jp/~sakurai/papers.html`.

[46] F.-R. Sinot, M. Fernández, and I. Mackie. Efficient reductions with director strings. In *LNCS* 2706 , *RTA* 2003.

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