Logical Methods in Computer Science
Vol. 5 (3:1) 2009, pp. 1–29
www.lmcs-online.org

Submitted Dec. 8, 2008
Published Jul. 15, 2009

# A THEORY OF EXPLICIT SUBSTITUTIONS WITH SAFE AND FULL COMPOSITION

DELIA KESNER

PPS (Université Paris-Diderot and CNRS), France
*e-mail address*: kesner@pps.jussieu.fr

Abstract. Many different systems with explicit substitutions have been proposed to implement a large class of higher-order languages. Motivations and challenges that guided the development of such calculi in functional frameworks are surveyed in the first part of this paper. Then, very simple technology in named variable-style notation is used to establish a theory of explicit substitutions for the lambda-calculus which enjoys a whole set of useful properties such as full composition, simulation of one-step beta-reduction, preservation of beta-strong normalisation, strong normalisation of typed terms and confluence on metaterms. Normalisation of related calculi is also discussed.

## 1. Introduction

This paper is about *explicit substitutions* (ES), a formalism that - by decomposing the *implicit* substitution operation into more atomic steps - allows a better understanding of the execution models of higher-order 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 $\lambda$-calculus whose solely reduction rule is given by $(\lambda x.t)\ v \rightarrow_\beta t\{x/v\}$, where the operation $t\{x/v\}$ denotes the result of substituting all the *free* occurrences of $x$ in $t$ by $v$, a notion that can be formally defined *modulo $\alpha$-conversion*[1] as follows:

$$
\begin{array}{lcll}
x\{x/v\} & := & v & \\
y\{x/v\} & := & y & x \neq y \\
(u_1 u_2)\{x/v\} & := & u_1\{x/v\} u_2\{x/v\} & \\
(\lambda y.u)\{x/v\} & := & \lambda y.u\{x/v\} &
\end{array}
$$

The simplest way to specify a $\lambda$-calculus with ES is to incorporate substitution operators into the language, then to transform the equalities of the previous specification into a set

*1998 ACM Subject Classification:* F.3.2, D.1.1, F.4.1.
*Key words and phrases:* operational semantics, functional languages, lambda calculus.

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

of reduction rules (so that one still works modulo $\alpha$-conversion). The following reduction system, known as $\lambda\mathtt{x}$ [Lin86, Lin92, Ros92, BR95], is thus obtained.

$$\begin{array}{lcll}
(\lambda x.t)\ v & \to & t[x/v] & \\
x[x/v] & \to & v & \\
y[x/v] & \to & y & x \neq y \\
(u_1 u_2)[x/v] & \to & u_1[x/v]u_2[x/v] & \\
(\lambda y.u)[x/v] & \to & \lambda y.u[x/v] &
\end{array}$$

The $\lambda\mathtt{x}$-calculus corresponds to the minimal behaviour[2] that can be found among the calculi with ES appearing in the literature (equivalent minimal behaviours can be found, for example, in [Cur91, BBLRD96, KR98]). However, when using this simple operational semantics, outermost substitutions must be always delayed until the total execution of all the innermost substitutions appearing in the same environment. Thus for example, the propagation of the outermost substitution $[x/v]$ in the term $(zyx)[y/xx][x/v]$ must be delayed until $[y/xx]$ is first executed on $zyx$.

This restriction can be recovered by the use of more sophisticated interactions, known as *composition* of substitutions, which allow in particular the propagation of substitutions through other substitutions. Thus for example, $(zyx)[y/xx][x/v]$ can be reduced to $(zyx)[x/v][y/(xx)[x/v]]$, which can be further reduced to $(zyv)[y/vv]$, a term equal to $(zyx)[y/xx]\{x/v\}$, where $\{x/v\}$ is the *meta/implicit* substitution that the *explicit* substitution $[x/v]$ is supposed to implement.

In these twenty last years there has been a growing interest in $\lambda$-calculi with ES. They can be defined either with unary [Ros92, LRD94] or n-ary [ACCL91, HL89] substitutions, by using de Bruijn notation [dB72, dB78], or levels [LRD95], or nominal logic [GP99], or combinators [GL99], or director strings [SFM03], or ... simply by named variables as in the $\lambda\mathtt{x}$-calculus. Besides different notations, a calculus with ES can be also seen as a term notation for a logical system where the reduction rules behave like cut elimination transformations [Her94, DU01, KL08].

Composition rules for ES first appeared in $\lambda\sigma$ [ACCL91]. They turn out to be necessary to get confluence on open terms [HL89] in calculi implementing higher-order unification [DHK00] or functional abstract machines [LM99, HMP96]. They also guarantee a simple property, called *full composition*, that calculi without composition do not enjoy: any term of the form $t[x/u]$ can be reduced to $t\{x/u\}$; in other words, explicit substitution implements the implicit one. Indeed, taking again the previous example, $(zyx)[y/xx][x/v]$ reduces to $(zyx)[y/xx]\{x/v\} = (zyv)[y/vv]$. Many calculi such as $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$ [HL89], $\lambda_{sub}$ [Mil06], $\lambda\mathtt{lxr}$ [KL05, KL07] and $\lambda\mathtt{es}$ [Kes07] enjoy full composition.

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 $\lambda$-calculus is perfectly *orthogonal* (in particular does not have critical pairs), calculi with ES such as $\lambda\mathtt{x}$ suffer at least from the following well-known diverging example:

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

Different solutions were adopted in the literature to close this diagram. If no new rewriting rule is added to those of the minimal $\lambda\mathtt{x}$-calculus, then reduction turns out to be confluent on terms but not on *metaterms* (terms with metavariables used to represent

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

incomplete programs and proofs). If liberal rules for composition are considered, as in $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, or $\lambda s_e$ [KR97], then one recovers confluence on metaterms but loses preservation of $\beta$-strong normalisation (PSN) as not all the $\beta$-strongly normalising terms remain normalising in the corresponding ES version. This phenomenon, known as Melliès' counter-example [Mel95] (see also [BG99] for later counterexamples in named calculi), shows a flaw in the design of ES calculi since they are supposed to implement their underlying calculus (in our case the $\lambda$-calculus) without losing its good properties.

There are many ways to avoid Melliès' counter-example in order to recover the PSN property. One can forbid the substitution operators to cross $\lambda$-abstractions or avoid composition of substitutions. 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 [LM99, For02], not able to express *strong* beta-equality (used for example in implementations of proof-assistants). The second solution [BBLRD96] is drastic when composition of substitutions is needed for implementations of HO unification [DHK00] or functional abstract machines [LM99, HMP96]. The last one does not take advantage of the notion of ES because they can be neither composed nor even delayed.

Fortunately, confluence on metaterms and preservation of $\beta$-strong normalisation can live together, this is for example the case of $\lambda_{ws}$ [DG99, DG01] and $\lambda\mathtt{lxr}$, which both introduce a *controlled* notion of composition for substitutions. Syntax of $\lambda_{ws}$ is based on terms with explicit weakening constructors. Its operational semantics reveals [DCKP00] a natural understanding of ES in terms of Linear Logic's proof-nets [Gir87], which are a geometrical representation of linear logic sequent proofs that incorporate a clear mechanism to control *weakening* and *contraction*. Weakening, viewed as erasure, and contraction, viewed as duplication, are precisely the starting points of the $\lambda\mathtt{lxr}$-calculus whose syntax is obtained by incorporating these new operators to the $\lambda$-terms. The reduction system of $\lambda\mathtt{lxr}$ contains 6 equations and 19 rewriting rules, thus requiring a big number of cases when developing some 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 expected from a calculus with ES, there is no justified reason to think that they are also *necessary*.

We choose here to use simple syntax in named variable notation style to define a formalism with full and safe composition that we call $\lambda\mathtt{ex}$-calculus. Thus, we dissociate the operational semantics of the calculus from all the renaming details that are necessary to specify higher-order substitution on terms that are implemented by non-trivial technologies such as de Bruijn indices or nominal notation. Even if our choice implies the use of $\alpha$-equivalence, we think that this presentation is more appropriate to focus on the fundamental (operational) properties of full and safe composition. It is now perfectly well-understood in the literature how to translate terms with named variables into other notations, so that we expect these translations to be able to preserve all the properties of the $\lambda\mathtt{ex}$-calculus.

The $\lambda\mathtt{ex}$-calculus is obtained by extending $\lambda\mathtt{x}$ with one rewriting rule to specify composition of *dependent* substitutions and 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 anymore the complex manipulation of explicit operators for contraction and weakening used in $\lambda\mathtt{lxr}$ to guarantee PSN. The substitutions of $\lambda\mathtt{ex}$ are defined by means of *unary* constructors but have the same expressive power as *n-ary* substitutions. Indeed, while simultaneous substitutions are specified by *lists* (given by n-ary substitutions) in $\lambda\sigma$, they are modelled by *sets* (given by commutation of independent unary substitutions) in $\lambda$`ex`.

We thus achieve the definition of a concise language being easy to understand, and enjoying a useful set of properties: confluence on metaterms (and thus on terms), simulation of one-step $\beta$-reduction, full composition, preservation of $\beta$-strong normalisation and strong normalisation of typed terms (SN).

Most of the available SN proofs for calculi with composition are not really first-hand: either one simulates reduction by means of another well-founded relation, or SN is deduced from a sufficient property, as for example PSN. Proofs using the first technique are for example those for $\lambda_{ws}$ in [DCKP03] and $\lambda$`lxr` [KL07], based on the well-foundedness of the reduction relation for multiplicative exponential linear logic (MELL) proof-nets [Gir87]. An example of SN proof using the second technique is that for $\lambda$`es`, where PSN is obtained by two consecutive translations, one from $\lambda$`es` into a calculus with ES and weakening, the second one from this intermediate calculus into the Church-Klop's $\Lambda_I$-calculus [Klo80]. In both cases the resulting proofs are long, particularly because they make use of normalisation properties of other (related) calculi.

It is then desirable to provide more direct arguments to prove normalisation properties of full and safe composition, thus avoiding unnecessary *detours* through other complex theories. And this becomes even necessary when one realises that normalisation of a calculus which allows duplication of void substitutions, such as $\lambda$`ex`, cannot be understood in terms of calculi like MELL proof-nets where such behaviour is impossible.

The technical tools used in the paper to show PSN for $\lambda$`ex` are the following. We first define a *perpetual* reduction strategy for $\lambda$`ex`: if $t$ can be reduced to $t'$ by the strategy, and $t' \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$. In particular, since the perpetual strategy reduces $t[x/u]$ to $t\{x/u\}$, one has to show that normalisation of **I***mplicit* substitution implies normalisation of **E***xplicit* substitution. More precisely,

$$(\mathbf{IE})\ u \in \mathcal{SN}_{\lambda\mathtt{ex}}\ \&\ t\{x/u\} \in \mathcal{SN}_{\lambda\mathtt{ex}} \text{ imply } t[x/u] \in \mathcal{SN}_{\lambda\mathtt{ex}}.$$

In other words, explicit substitution implements implicit substitution but nothing more than that, otherwise one may get calculi such as $\lambda\sigma$ where $t[x/u]$ does much more than $t\{x/u\}$. A consequence of the **IE** property is that standard techniques to show SN based on *meta*-substitution can also be applied to calculi with ES, thus simplifying the reasoning considerably. Indeed, the perpetual strategy is used to give an inductive characterisation of the set $\mathcal{SN}_{\lambda\mathtt{ex}}$ by means of just four inference rules. This inductive characterisation is then used to show that *untyped* terms preserve $\beta$-strong normalisation and that *typed* terms are in $\mathcal{SN}_{\lambda\mathtt{ex}}$. At the end of the paper we also show how SN of other calculi with or without full composition can be obtained from SN of $\lambda$`ex`.

All our proofs are developed using simple logical tools: intuitionistic reasoning, induction, reasoning by cases on decidable predicates. All this gives a constructive (no use of classical logic) flavour to the whole development.

The proof technique used to show the **IE** property is mostly inspired from the PSN proofs used for the *non equational* systems $\lambda$`x` and $\lambda_{ws}$ in [LLD$^+$04] and [ABR00]. Current investigations carried out in [SvO07] show PSN for different calculi with (full or not) composition. The approach is based on the analysis of *minimal* non-terminating reduction sequences. The calculus proposed in [Sak] specifies commutation of independent substitutions by a *non-terminating* rewriting system (instead of an equation), thus leading to complicated notions and proofs.

This paper extends some ideas summarised in [Kes07, Kes08], particularly by the use of intersection types to characterise the set $\mathcal{SN}_{\lambda\mathtt{ex}}$ as well as the use of the Z-property of van Oostrom [vO] to show confluence. It is organised as follows. Section 2 introduces syntax and reduction rules for the $\lambda\mathtt{ex}$-calculus. The perpetual strategy for $\lambda\mathtt{ex}$ is introduced in Section 3 together with its corresponding Perpetuality Theorem. This fundamental theorem is proved thanks to a key property whose proof is left to Sections 4 and 5. The equivalence between intersection typed and $\beta$-strongly normalising terms is given in Section 6. In Section 7 we explain how to infer SN for other calculi with ES. In Section 8 we prove confluence for metaterms. Finally we conclude and give directions for further work in Section 9.

## 2. Syntax

The $\lambda\mathtt{ex}$-calculus can be viewed as a simple extension of the $\lambda\mathtt{x}$-calculus. The set of *terms* (meta-variables $s, t, u, v$) is defined by the following grammar.

$$\mathcal{T} ::= x \mid \mathcal{T}\mathcal{T} \mid \lambda x.\mathcal{T} \mid \mathcal{T}[x/\mathcal{T}]$$

*Free* and *bound* variables of $t$, written respectively $\mathtt{fv}(t)$ and $\mathtt{bv}(t)$, are defined by induction as follows:

$$\begin{array}{lcllcl}
\mathtt{fv}(x) & := & \{x\} & \mathtt{bv}(x) & := & \emptyset \\
\mathtt{fv}(\lambda x.u) & := & \mathtt{fv}(u) \setminus \{x\} & \mathtt{bv}(\lambda x.u) & := & \mathtt{bv}(u) \cup \{x\} \\
\mathtt{fv}(uv) & := & \mathtt{fv}(u) \cup \mathtt{fv}(v) & \mathtt{bv}(uv) & := & \mathtt{bv}(u) \cup \mathtt{bv}(v) \\
\mathtt{fv}(u[x/v]) & := & (\mathtt{fv}(u) \setminus \{x\}) \cup \mathtt{fv}(v) & \mathtt{bv}(u[x/v]) & := & \mathtt{bv}(u) \cup \{x\} \cup \mathtt{bv}(v)
\end{array}$$

Thus, $\lambda x.t$ and $t[x/u]$ bind the free occurrences of $x$ in $t$.

The congruence generated by renaming of bound variables is called *$\alpha$-conversion*. Thus for example $(\lambda y.x)[x/y] =_\alpha (\lambda z.x')[x'/y]$. Given a term of the form $t[x/u][y/v]$, the two outermost substitutions are said to be *independent* iff $y \notin \mathtt{fv}(u)$, and *dependent* iff $y \in \mathtt{fv}(u)$. Notice that in both cases we can always assume $x \notin \mathtt{fv}(v)$ by $\alpha$-conversion. We use the notation $\overline{t_n}$ for a list of $n$ $(n \geq 0)$ terms $t_1, \ldots, t_n$ and $u\overline{t_n}$ for $ut_1 \ldots t_n$, which is in turn an abbreviation of $(\ldots((ut_1)t_2)\ldots t_n)$.

*Meta-substitution* on terms is defined modulo $\alpha$-conversion in such a way that capture of variables is avoided. It is given by the following equations.

$$\begin{array}{lcl}
x\{x/v\} & := & v \\
y\{x/v\} & := & y \text{ if } y \neq x \\
(\lambda y.t)\{x/v\} & := & \lambda y.t\{x/v\} \\
(tu)\{x/v\} & := & t\{x/v\}u\{x/v\} \\
t[y/u]\{x/v\} & := & t\{x/v\}[y/u\{x/v\}]
\end{array}$$

Thus for example $(\lambda y.x)\{x/y\} = \lambda z.y$. Notice that $t\{x/u\} = t$ if $x \notin \mathtt{fv}(t)$.

Besides $\alpha$-conversion, we consider the equations and rewriting rules in Figure 1.

Notice that $\alpha$-conversion allows to assume that there is no capture of variables in the previous equations and rules. Thus for example we can assume $y \neq x$ and $y \notin \mathtt{fv}(v)$ in the rewriting rule `Lamb`. Same kind of assumptions are done for the rewriting rule `Comp` and the equation `C`.

The *rewriting relation* $\rightarrow_{\mathtt{Bx}}$ is generated by all the rewriting rules in Figure 1 and $\rightarrow_{\mathtt{x}}$ is only generated by the five last ones. The *equivalence relation* $=_{\mathtt{e}}$ is generated by the conversions $\alpha$ and `C`. The *reduction relations* $\rightarrow_{\mathtt{ex}}$ and $\rightarrow_{\lambda\mathtt{ex}}$ are respectively generated by

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

Figure 1: The $\lambda\mathtt{ex}$-calculus

the *rewriting relations* $\to_{\mathtt{x}}$ and $\to_{\mathtt{Bx}}$ *modulo* $=_{\mathtt{e}}$ (thus specifying rewriting on $\mathtt{e}$-equivalence classes):

$$\begin{array}{lll} t \to_{\mathtt{ex}} t' & \text{iff} & \exists\, s, s' \text{ s.t. } \ t =_{\mathtt{e}} s \to_{\mathtt{x}} s' =_{\mathtt{e}} t' \\ t \to_{\lambda\mathtt{ex}} t' & \text{iff} & \exists\, s, s' \text{ s.t. } \ t =_{\mathtt{e}} s \to_{\mathtt{Bx}} s' =_{\mathtt{e}} t' \end{array}$$

Given any reduction relation $\mathcal{R}$, a term $t$ is said to be in $\mathcal{R}$-*normal form*, written $t \in \mathcal{NF}_{\mathcal{R}}$, if there is no $u$ such that $t \to_{\mathcal{R}} u$. As an example, an inductive definition of $\mathcal{NF}_{\lambda\mathtt{ex}}$ can be given by: $t_1, \ldots, t_n \in \mathcal{NF}_{\lambda\mathtt{ex}}$ imply $xt_1 \ldots t_n \in \mathcal{NF}_{\lambda\mathtt{ex}}$, and $t \in \mathcal{NF}_{\lambda\mathtt{ex}}$ implies $\lambda x.t \in \mathcal{NF}_{\lambda\mathtt{ex}}$.

Again for any reduction relation $\mathcal{R}$, a term $t$ is said to be $\mathcal{R}$-*strongly normalising*, written $t \in \mathcal{SN}_{\mathcal{R}}$, if there is no infinite $\mathcal{R}$-reduction sequence starting at $t$, in which case the notation $\eta_{\mathcal{R}}(t)$ means the *maximal length of a* $\mathcal{R}$*-reduction sequence starting at* $t$. An inductive definition of $\mathcal{SN}_{\mathcal{R}}$ is usually given by:

$$t \in \mathcal{SN}_{\mathcal{R}} \text{ iff } \forall s\ (t \to_{\mathcal{R}} s \text{ implies } s \in \mathcal{SN}_{\mathcal{R}})$$

The notation $\to^*_{\mathcal{R}}$ (resp. $\to^+_{\mathcal{R}}$) is used for the reflexive (resp. reflexive and transitive) closure of $\to_{\mathcal{R}}$. Thus in particular, if $t \to^*_{\lambda\mathtt{ex}} t'$ in 0 reduction steps, then $t =_{\mathtt{e}} t'$.

The following basic properties can be shown by a straightforward induction on the reduction relation.

**Lemma 2.1** (Basic Properties)**.** *Let* $\mathcal{R} \in \{\mathtt{ex}, \lambda\mathtt{ex}\}$ *and let* $t, t', u$ *be terms.*

- *If* $t \to_{\mathcal{R}} t'$, *then* $\mathtt{fv}(t') \subseteq \mathtt{fv}(t)$.
- *If* $t \to_{\mathcal{R}} t'$, *then* $u\{x/t\} \to^*_{\mathcal{R}} u\{x/t'\}$ *and* $t\{x/u\} \to_{\mathcal{R}} t'\{x/u\}$. *Thus in particular* $t\{x/u\} \in \mathcal{SN}_{\mathcal{R}}$ *implies* $t \in \mathcal{SN}_{\mathcal{R}}$.

As explained in Section 1 the composition rule Comp and the equation C guarantee the following property:

**Lemma 2.2** (Full Composition for Terms)**.** *Let* $t, u$ *be terms. Then* $t[x/u] \to^+_{\mathtt{ex}} t\{x/u\}$.

*Proof.* By induction on $t$. Consider $t = s[y/v]$. If $x \in \mathtt{fv}(v)$, then $s[y/v][x/u] \to_{\mathtt{Comp}} s[x/u][y/v[x/u]] \to^+_{\mathtt{ex}\ (i.h.)} s\{x/u\}[y/v\{x/u\}] = t\{x/u\}$. If $x \notin \mathtt{fv}(v)$, then $s[y/v][x/u] =_{\mathtt{C}} s[x/u][y/v] \to^+_{\mathtt{ex}\ (i.h.)} s\{x/u\}[y/v] = t\{x/u\}$. All the other cases are straightforward. $\square$

Simulation of one-step $\beta$-reduction is then a direct consequence of full composition.

**Lemma 2.3** (Simulating One-Step $\beta$-Reduction)**.** *Let $t, t'$ be $\lambda$-terms. If $t \to_\beta t'$, then $t \to^*_{\lambda\mathtt{ex}} t'$.*

## 3. Perpetuality and Preservation of Normalisation

A *perpetual* strategy gives an *infinite* reduction sequence for a term, if one exists, otherwise, it gives a finite reduction sequence leading to some normal form. Perpetual strategies, introduced in [BBKV76], can be seen as antonyms of normalising strategies, they are particularly used to obtain normalisation results. We refer the reader to [vRSSX99] for more details.

Perpetual strategies can be specified by one or many steps. In contrast to *one-step* strategies for ES given for example in [Bon01a], we now define a *many-step* strategy giving a reduct for any $t \notin \mathcal{NF}_{\lambda\mathtt{ex}}$. This is done according to the following cases. If $t = xt_1 \dots t_n$, rewrite the *left-most* $t_i$ which is reducible. If $t = \lambda x.u$, rewrite $u$. If $t = (\lambda x.s)u\overline{v_n}$, rewrite the head redex. If $t = s[x/u]\overline{v_n}$ and $u \notin \mathcal{SN}_{\lambda\mathtt{ex}}$, rewrite $u$. If $t = s[x/u]\overline{v_n}$ and $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$, apply full composition to the head redex $s[x/u]$ by using as many steps as necessary. Formally,

**Definition 3.1** (A Strategy for Terms)**.** The *strategy* $\rightsquigarrow$ on terms is given by an inductive definition.

$$\frac{\overline{u_n} \in \mathcal{NF}_{\lambda\mathtt{ex}} \quad t \rightsquigarrow t'}{x\overline{u_n}t\overline{v_m} \rightsquigarrow x\overline{u_n}t'\overline{v_m}}\ (\mathtt{p\text{-}var}) \qquad \frac{t \rightsquigarrow t'}{\lambda x.t \rightsquigarrow \lambda x.t'}\ (\mathtt{p\text{-}abs}) \qquad \frac{}{(\lambda x.t)u\overline{u_n} \rightsquigarrow t[x/u]\overline{u_n}}\ (\mathtt{p\text{-}B})$$

$$\frac{u \in \mathcal{SN}_{\lambda\mathtt{ex}}}{t[x/u]\overline{v_n} \rightsquigarrow t\{x/u\}\overline{v_n}}\ (\mathtt{p\text{-}subs1}) \qquad \frac{u \notin \mathcal{SN}_{\lambda\mathtt{ex}} \quad u \rightsquigarrow u'}{t[x/u]\overline{v_n} \rightsquigarrow t[x/u']\overline{v_n}}\ (\mathtt{p\text{-}subs2})$$

The strategy is deterministic so that $t \rightsquigarrow u$ and $t \rightsquigarrow v$ imply $u = v$. Moreover, the strategy is not necessarily leftmost-outermost or left-to-right because of the `(p-subs1)` rule: substitution propagation can be performed in any order. Notice that the syntactical details concerning the manipulation of substitutions are completely hidden in the definition of the strategy which is only based on the full composition property. This makes the results of this section to be abstract and modular. A basic property of the strategy is:

**Lemma 3.2.** *Let $t, t'$ be terms. If $t \rightsquigarrow t'$, then $t \to^+_{\lambda\mathtt{ex}} t'$.*

*Proof.* By induction on the definition of the strategy $\rightsquigarrow$ using Lemma 2.2. □

The strategy turns out to be perpetual, that is, terminating terms are stable by anti-reduction (also called expansion). The proof of this property is presented in a modular way, by leaving all the details concerning the particularities of the substitution calculus to one single statement, called *the* **IE** *property* (Lemma 5.9) and fully developed in the next section.

**Theorem 3.3** (Perpetuality Theorem)**.** *Let $t, t'$ be terms. If $t \rightsquigarrow t'$ and $t' \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

*Proof.* By induction on the definition of the strategy $\rightsquigarrow$.

- $t = (\lambda x.s)u\overline{u_n} \rightsquigarrow s[x/u]\overline{u_n} = t'$ by (p-B). If $s[x/u]\overline{u_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $s, u, \overline{u_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$. We show $(\lambda x.s)u\overline{u_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by induction on $\eta_{\lambda\mathtt{ex}}(s) + \eta_{\lambda\mathtt{ex}}(u) + \Sigma_{i\in 1\ldots n}\ \eta_{\lambda\mathtt{ex}}(u_i)$. For that, it is sufficient to show that every $\lambda\mathtt{ex}$-reduct of $(\lambda x.s)u\overline{u_n}$ is in $\mathcal{SN}_{\lambda\mathtt{ex}}$. If the reduction takes place in a subterm of $(\lambda x.s)u\overline{u_n}$, then the property holds by the i.h. Otherwise $(\lambda x.s)u\overline{u_n} \rightarrow_{\mathtt{B}} s[x/u]\overline{u_n}$ which is in $\mathcal{SN}_{\lambda\mathtt{ex}}$ by hypothesis. We thus conclude $(\lambda x.s)u\overline{u_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$.
- $t = s[x/u]\overline{v_n} \rightsquigarrow s[x/u']\overline{v_n} = t'$ by (p-subs2), so that $u \notin \mathcal{SN}_{\lambda\mathtt{ex}}$ and $u \rightsquigarrow u'$. If $s[x/u']\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then in particular $u' \in \mathcal{SN}_{\lambda\mathtt{ex}}$, thus $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by the i.h. From $u \notin \mathcal{SN}_{\mathtt{ex}}$ and $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$ we can get any proposition, so in particular $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$.
- $t = s[x/u]\overline{v_n} \rightsquigarrow s\{x/u\}\overline{v_n} = t'$ by (p-subs1) so that $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$. Then the **IE** property (Lemma 5.9 in Section 4) allows to conclude.

All the other cases are straightforward. □

An inductive syntactic characterisation of the set $\mathcal{SN}_{\lambda\mathtt{ex}}$ can be now given using the perpetual strategy. This kind of characterisation is usually useful when developing SN proofs. An inductive syntactic definition of SN terms for the $\lambda$-calculus is given for example in [vR96]. It was then extended in [LLD+04, Bon01b] for calculi with ES, but using many different inference rules to characterise SN terms of the form $t[x/u]$. We just give here one inference rule for each possible syntactical form.

**Definition 3.4** (Inductive Characterisation of $\mathcal{SN}_{\lambda\mathtt{ex}}$)**.** The *inductive set* $\mathcal{ISN}$ is defined as follows:

$$\frac{t_1,\ldots,t_n \in \mathcal{ISN} \quad n \geq 0}{x t_1 \ldots t_n \in \mathcal{ISN}}\ (\mathtt{var}) \qquad \frac{u[x/v]t_1 \ldots t_n \in \mathcal{ISN} \quad n \geq 0}{(\lambda x.u)v t_1 \ldots t_n \in \mathcal{ISN}}\ (\mathtt{app})$$

$$\frac{u\{x/v\}t_1 \ldots t_n \in \mathcal{ISN} \quad v \in \mathcal{ISN} \quad n \geq 0}{u[x/v]t_1 \ldots t_n \in \mathcal{ISN}}\ (\mathtt{subs}) \qquad \frac{u \in \mathcal{ISN}}{\lambda x.u \in \mathcal{ISN}}\ (\mathtt{abs})$$

**Proposition 3.5.** *$\mathcal{SN}_{\lambda\mathtt{ex}} = \mathcal{ISN}$.*

*Proof.* If $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $t \in \mathcal{ISN}$ is proved by induction on the lexicographic pair $\langle \eta_{\lambda\mathtt{ex}}(t), t\rangle$. If $t \in \mathcal{ISN}$, then $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$ is proved by induction on $t \in \mathcal{ISN}$ using Theorem 3.3. □

The PSN property received a lot of attention in calculi with explicit substitutions, starting from an unexpected result given by Melliès [Mel95] who has shown that there are $\beta$-strongly normalisable $\lambda$-terms that are not strongly normalisable in calculi with composition such as $\lambda\sigma$ [ACCL91]. Since then, many formalisms with and without composition have been shown to enjoy PSN. The proof technique used in this paper to show PSN is based on the Perpetuality Theorem and is mostly inspired from [ABR00, LLD+04, ABR00]. However, the use of two quite abstract concepts, namely, full composition and the **IE** property, makes our proof much more modular than the existing ones.

**Theorem 3.6** (PSN for $\lambda$-terms)**.** *If $t \in \mathcal{SN}_\beta$, then $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

*Proof.* By induction on the definition of $\mathcal{SN}_\beta$ [vR96] using the inductive Definition 3.4 and Proposition 3.5 (which holds by the Perpetuality Theorem 3.3).

If $t = x t_1 \ldots t_n$ with $t_i \in \mathcal{SN}_\beta$, then $t_i \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by the i.h. so that the (var) rule allows to conclude. The case $t = \lambda x.u$ is similar. If $t = (\lambda x.u)v t_1 \ldots t_n$, with $u\{x/v\}t_1 \ldots t_n \in \mathcal{SN}_\beta$

and $v \in \mathcal{SN}_\beta$, then both terms are in $\mathcal{SN}_{\lambda\texttt{ex}}$ by the i.h. so that the (subs) rule gives $u[x/v]t_1 \dots t_n \in \mathcal{SN}_{\lambda\texttt{ex}}$ and the (app) rule gives $(\lambda x.u)vt_1 \dots t_n \in \mathcal{SN}_{\lambda\texttt{ex}}$. □

*Alternative Proof.* By induction on the definition of $\mathcal{SN}_\beta$ [vR96] using the **IE** property (Lemma 5.9 in Section 4).

If $t = xt_1 \dots t_n$ with $t_i \in \mathcal{SN}_\beta$, then $t_i \in \mathcal{SN}_{\lambda\texttt{ex}}$ by the i.h. so that $t \in \mathcal{SN}_{\lambda\texttt{ex}}$ is straightforward. If $t = \lambda x.u$ with $u \in \mathcal{SN}_\beta$, then $u \in \mathcal{SN}_{\lambda\texttt{ex}}$ by the i.h. and thus $t \in \mathcal{SN}_{\lambda\texttt{ex}}$ is also straightforward. If $t = (\lambda x.u)vt_1 \dots t_n$, with $u\{x/v\}t_1 \dots t_n \in \mathcal{SN}_\beta$ and $v \in \mathcal{SN}_\beta$, then both terms are in $\mathcal{SN}_{\lambda\texttt{ex}}$ by the i.h. The **IE** property gives $t' = u[x/v]t_1 \dots t_n \in \mathcal{SN}_{\lambda\texttt{ex}}$ so that in particular $u, v, t_1 \dots, t_n \in \mathcal{SN}_{\lambda\texttt{ex}}$. We show $t = (\lambda x.u)vt_1 \dots t_n \in \mathcal{SN}_{\lambda\texttt{ex}}$ by induction on $\mu_{\lambda\texttt{ex}}(u) + \mu_{\lambda\texttt{ex}}(v) + \Sigma_i\ \mu_{\lambda\texttt{ex}}(t_i)$. For that, it is sufficient to show that every $\lambda\texttt{ex}$-reduct of $t$ is in $\mathcal{SN}_{\lambda\texttt{ex}}$. Now, if the $\lambda\texttt{ex}$-reduct of $t$ comes from an internal reduction, then conclude with the i.h. Otherwise, $t \rightarrow_{\lambda\texttt{ex}} t'$ which is already in $\mathcal{SN}_{\lambda\texttt{ex}}$. □

## 4. The Labelling Technique

This section develops the key technical tools used to guarantee that the strategy $\rightsquigarrow$ (Definition 3.1) is perpetual. More precisely, we want show that normalisation of **I***mplicit* substitution implies normalisation of **E***xplicit* substitution:

$$(\textbf{IE})\ u \in \mathcal{SN}_{\lambda\texttt{ex}}\ \&\ t\{x/u\}\overline{v_n} \in \mathcal{SN}_{\lambda\texttt{ex}} \text{ imply } t[x/u]\overline{v_n} \in \mathcal{SN}_{\lambda\texttt{ex}}$$

For that we adapt the labelling technique [DG01, ABR00, Bon01b] to the equational case. The technique can be summarised by the following steps:

(1) Use a labelling to mark some $\lambda\texttt{ex}$-*strongly normalising* terms used as substitutions. Thus for example $t[\![x/u]\!]$ indicates that $u \in \mathcal{T}\ \&\ u \in \mathcal{SN}_{\lambda\texttt{ex}}$.
(2) Enrich the original $\lambda\texttt{ex}$-reduction system with a relation $\underline{\texttt{ex}}$ used only to propagate *terminating labelled* substitutions. Let $\lambda\underline{\texttt{ex}}$ be the enriched calculus.
(3) Show that $u \in \mathcal{SN}_{\lambda\texttt{ex}}\ \&\ t\{x/u\}\overline{v_n} \in \mathcal{SN}_{\lambda\texttt{ex}}$ imply $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\texttt{ex}}}$.
(4) Show that $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\texttt{ex}}}$ implies $t[x/u]\overline{v_n} \in \mathcal{SN}_{\lambda\texttt{ex}}$.

We now develop the first and second points, leaving the two last ones to Section 5.

**Definition 4.1** (Labelled Terms)**.** Given a finite set of variables $\mathbb{S}$, the $\mathbb{S}$-*labelled terms* (or simply *labelled terms* if $\mathbb{S}$ is clear from the context), are defined by the following grammar:

$$\mathcal{L}_\mathbb{S} \quad ::= \quad x \mid \mathcal{L}_\mathbb{S}\mathcal{L}_\mathbb{S} \mid \lambda x.\mathcal{L}_\mathbb{S} \mid \mathcal{L}_\mathbb{S}[x/\mathcal{L}_\mathbb{S}] \mid \mathcal{L}_\mathbb{S}[\![x/v]\!]\ (v \in \mathcal{T} \cap \mathcal{SN}_{\lambda\texttt{ex}}\ \&\ \texttt{fv}(v) \subseteq \mathbb{S})$$

Thus, labelled substitutions can only contain *terms* so in particular they cannot contain other labelled substitutions. Notice that all the terms (as defined in Section 2) are labelled terms, but some terms with arbitrary labels are not. Labelled terms need not be confused with the *decent* terms of [Blo97] which do not have labels at all and are not stable by reduction.

We can always assume that subterms $\lambda x.u$, $u[x/v]$ and $u[\![x/v]\!]$ inside $t \in \mathcal{L}_\mathbb{S}$ are s.t. $x \notin \mathbb{S}$. Indeed, $\alpha$-conversion allows to choose names outside $\mathbb{S}$ for the bound variables of labelled terms. As a consequence, no substitution (labelled or not) can be used to affect the bodies of other labelled substitutions (whose free variables are all in $\mathbb{S}$). That means also that given a term $t$ having a subterm $u[\![x/v]\!]$, no free occurrence of $y$ in $v$ can be bound in the path leading to the root of $t$. In other words, the bodies of labelled

| | | | |
|---|---|---|---|
| **Equations** : | | | |
| $t[y/u][\![x/v]\!]$ | $=_{\underline{\mathtt{C}}}$ | $t[\![x/v]\!][y/u]$ | if $x \notin \mathtt{fv}(u)$ & $y \notin \mathtt{fv}(v)$ |
| $t[\![y/u]\!][\![x/v]\!]$ | $=_{\underline{\mathtt{C}}}$ | $t[\![x/v]\!][\![y/u]\!]$ | if $x \notin \mathtt{fv}(u)$ & $y \notin \mathtt{fv}(v)$ |
| **Rules** : | | | |
| $x[\![x/v]\!]$ | $\rightarrow_{\underline{\mathtt{Var}}}$ | $v$ | |
| $t[\![x/v]\!]$ | $\rightarrow_{\underline{\mathtt{Gc}}}$ | $t$ | if $x \notin \mathtt{fv}(t)$ |
| $(tu)[\![x/v]\!]$ | $\rightarrow_{\underline{\mathtt{App}}}$ | $t[\![x/v]\!]\ u[\![x/v]\!]$ | |
| $(\lambda y.t)[\![x/v]\!]$ | $\rightarrow_{\underline{\mathtt{Lamb}}}$ | $\lambda y.t[\![x/v]\!]$ | |
| $t[y/u][\![x/v]\!]$ | $\rightarrow_{\underline{\mathtt{Comp}}}$ | $t[\![x/v]\!][y/u[\![x/v]\!]]$ | if $x \in \mathtt{fv}(u)$ |

Figure 2: The $\underline{\mathtt{ex}}$-calculus

substitutions are safe since they are already normalising and cannot loose normalisation after reduction/substitution.

The idea behind the operational semantics of labelled terms, specified by the equations and reduction rules in Figure 2, is that labelled substitutions may commute/traverse ordinary substitutions but these last ones cannot traverse the labelled ones.

The *rewriting relation* $\rightarrow_{\underline{\mathtt{x}}}$ is generated by the rewriting rules in Figure 2 and the *equivalence relation* $=_{\underline{\mathtt{e}}}$ is generated by the conversions $\alpha$ and $\underline{\mathtt{C}}$. The *reduction relation* $\rightarrow_{\underline{\mathtt{ex}}}$ is generated by the rewriting relation $\rightarrow_{\underline{\mathtt{x}}}$ modulo $=_{\underline{\mathtt{e}}}$. In particular, both relations $\rightarrow_{\underline{\mathtt{x}}}$ and $\rightarrow_{\underline{\mathtt{ex}}}$ enjoy termination (see Lemma 4.7). An even richer reduction relation $\lambda\underline{\mathtt{ex}}$ can be defined on labelled terms by adding to $\underline{\mathtt{ex}}$ the old reduction relation $\lambda\mathtt{ex}$ but now on *labelled terms*. That is, $\rightarrow_{\lambda\underline{\mathtt{ex}}}$ is defined as the union of the rewriting relations $\rightarrow_{\mathtt{Bx}}$ and $\rightarrow_{\underline{\mathtt{x}}}$ on labelled terms modulo $\alpha \cup \mathtt{C} \cup \underline{\mathtt{C}}$-equivalence classes:

$$t \rightarrow_{\lambda\underline{\mathtt{ex}}} t' \text{ iff } \exists\, s, s' \text{ s.t. } t =_{\mathtt{e}\cup\underline{\mathtt{e}}} s \rightarrow_{\mathtt{Bx}\cup\underline{\mathtt{x}}} s' =_{\mathtt{e}\cup\underline{\mathtt{e}}} t'$$

In order to show that $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$ & $t\{x/u\}\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ imply $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\mathtt{ex}}}$ we first need to relate the $\lambda\underline{\mathtt{ex}}$-reduction relation to that of the $\lambda\mathtt{ex}$-calculus. For that, the reduction relation $\lambda\underline{\mathtt{ex}}$, which is defined on labelled terms, is split in two relations $\lambda\underline{\mathtt{ex}}^i$ and $\lambda\underline{\mathtt{ex}}^e$, on labelled terms as well, which will both be projected into $\lambda\mathtt{ex}$-reduction sequences. More precisely, $\lambda\underline{\mathtt{ex}}^i$ can be *weakly* projected (eventually empty steps) into $\lambda\mathtt{ex}$ while $\lambda\underline{\mathtt{ex}}^e$ can be *strongly* projected (at least one step) into $\lambda\underline{\mathtt{ex}}^e$ (details in the forthcoming Lemma 5.2).

**Definition 4.2** (Internal and External Reductions)**.** The *internal* reduction relation $\rightarrow_{\lambda\underline{\mathtt{ex}}^i}$ on labelled terms is given by adding to $\underline{\mathtt{ex}}$ the $\lambda\mathtt{ex}$-reduction relation in the bodies of labelled substitutions. Formally, $\rightarrow_{\lambda\underline{\mathtt{ex}}^i}$ is taken as the following reduction relation $\rightarrow_{\lambda\underline{\mathtt{x}}^i}$ on $\alpha \cup \mathtt{C} \cup \underline{\mathtt{C}}$-equivalence classes:

- If $u \rightarrow_{\mathtt{Bx}} u'$ and $u, u'$ are terms, then $t[\![x/u]\!] \rightarrow_{\lambda\underline{\mathtt{x}}^i} t[\![x/u']\!]$.
- If $t \rightarrow_{\underline{\mathtt{x}}} t'$, then $t \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'$.
- If $t \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'$, then $tu \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'u$, $ut \rightarrow_{\lambda\underline{\mathtt{x}}^i} ut'$, $\lambda x.t \rightarrow_{\lambda\underline{\mathtt{x}}^i} \lambda x.t'$, $t[x/u] \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'[x/u]$, $u[x/t] \rightarrow_{\lambda\underline{\mathtt{x}}^i} u[x/t']$, $t[\![x/u]\!] \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'[\![x/u]\!]$.

The *external* reduction relation $\rightarrow_{\lambda\underline{\mathtt{ex}}^e}$ on labelled terms is given by $\lambda\mathtt{ex}$-reduction on *labelled terms* everywhere except inside bodies of labelled substitutions. Formally, $\rightarrow_{\lambda\underline{\mathtt{ex}}^e}$ is taken as the following reduction relation $\rightarrow_{\lambda\underline{\mathtt{x}}^e}$ on $\alpha \cup \mathtt{C} \cup \underline{\mathtt{C}}$-equivalence classes:

- If $t \rightarrow_{\mathtt{Bx}} t'$ occurs outside a labelled substitution, then $t \rightarrow_{\lambda\underline{\mathtt{x}}^e} t'$.

- If $t \to_{\lambda\underline{\mathtt{x}}^e} t'$, then $tu \to_{\lambda\underline{\mathtt{x}}^e} t'u$, $ut \to_{\lambda\underline{\mathtt{x}}^e} ut'$, $\lambda x.t \to_{\lambda\underline{\mathtt{x}}^e} \lambda x.t'$, $t[x/u] \to_{\lambda\underline{\mathtt{x}}^e} t'[x/u]$, $u[x/t] \to_{\lambda\underline{\mathtt{x}}^e} u[x/t']$ and $t[\![x/u]\!] \to_{\lambda\underline{\mathtt{x}}^e} t'[\![x/u]\!]$.

**Lemma 4.3.** $\to_{\lambda\underline{\mathtt{ex}}} = \to_{\lambda\underline{\mathtt{ex}}^e} \cup \to_{\lambda\underline{\mathtt{ex}}^i}$.

*Proof.* Since we are working everywhere with $\alpha \cup \mathtt{C} \cup \underline{\mathtt{C}}$-equivalence classes, then it is sufficient to show $\to_{\mathtt{Bx}\cup\underline{\mathtt{x}}} = \to_{\lambda\underline{\mathtt{x}}^i} \cup \to_{\lambda\underline{\mathtt{x}}^e}$.

$\subseteq$: If $t \to_{\mathtt{Bx}} t'$ occurs inside a labelled substitution, then $t \to_{\lambda\underline{\mathtt{x}}^i} t'$, otherwise $t \to_{\lambda\underline{\mathtt{x}}^e} t'$. If $t \to_{\underline{\mathtt{x}}} t'$, then $t \to_{\lambda\underline{\mathtt{x}}^i} t'$.

$\supseteq$: By induction on the definitions of $\to_{\lambda\underline{\mathtt{x}}^e}$ and $\to_{\lambda\underline{\mathtt{x}}^i}$. □

Since $\lambda\underline{\mathtt{ex}}^i$-reduction will only be weakly projected into $\lambda\mathtt{ex}$, we need to guarantee that there are no infinite $\lambda\underline{\mathtt{ex}}^i$-reduction sequences starting at labelled term. This is exactly the goal of the final part of this section. We will then use this result in Section 5 to relate termination of $\lambda\mathtt{ex}$ to that of $\lambda\underline{\mathtt{ex}}$ (Corollary 5.4).

**Definition 4.4** (A Decreasing Measure for $\underline{\mathtt{Comp}}$)**.** For every variable $x \notin \mathbb{S}$, the function $\mathtt{af}_x(\_)$ counts the number of bodies of non-labelled substitutions having free occurrences of $x$. Formally, $\mathtt{af}_x(\_)$ is defined on labelled terms as follows.

$$\begin{array}{lcllcll}
\mathtt{af}_x(z) & := & 0 & \mathtt{af}_x(tu) & := & \mathtt{af}_x(t)+\mathtt{af}_x(u) & \\
\mathtt{af}_x(\lambda y.t) & := & \mathtt{af}_x(t) & \mathtt{af}_x(t[y/u]) & := & \mathtt{af}_x(t) & \text{if } x \notin \mathtt{fv}(u)\\
\mathtt{af}_x(t[\![y/u]\!]) & := & \mathtt{af}_x(t) & \mathtt{af}_x(t[y/u]) & := & \mathtt{af}_x(t)+1+\mathtt{af}_x(u) & \text{if } x \in \mathtt{fv}(u)
\end{array}$$

A second function $\mathtt{dep}(\_)$ counts the total number of $\mathtt{af}_x(\_)$ in a labelled term $t$, and this for all variables $x$ which are bound by some labelled substitution of $t$. Formally, $\mathtt{dep}(\_)$ is defined on labelled terms as follows.

$$\begin{array}{lcllcl}
\mathtt{dep}(x) & := & 0 & \mathtt{dep}(tu) & := & \mathtt{dep}(t)+\mathtt{dep}(u)\\
\mathtt{dep}(\lambda y.t) & := & \mathtt{dep}(t) & \mathtt{dep}(t[x/u]) & := & \mathtt{dep}(t)+\mathtt{dep}(u)\\
 & & & \mathtt{dep}(t[\![x/u]\!]) & := & \mathtt{dep}(t)+\mathtt{af}_x(t)
\end{array}$$

For example, given $v = w[w/(xx)[y/x]]$, we have $\mathtt{af}_x(v) = 2$ and $\mathtt{dep}(v[y/v][\![x/x_1]\!]) = 5$.

Notice that $\mathtt{af}_x(t) = 0$ if $x \notin \mathtt{fv}(t)$ and $\mathtt{dep}(t) = 0$ if $t$ does not have labelled substitutions. Notice also that $\mathtt{dep}(t[\![x/u]\!])$ is well-defined in terms of $\mathtt{af}_x$ since we can always assume $x \notin \mathbb{S}$ by $\alpha$-conversion.

**Definition 4.5** (A Decreasing Measure for $\underline{\mathtt{x}} \setminus \underline{\mathtt{Comp}}$)**.** We consider the following function $\mathtt{K}(\_)$ on terms:

$$\begin{array}{lcllcl}
\mathtt{K}(x) & := & 1 & \mathtt{K}(tu) & := & \mathtt{K}(t)+\mathtt{K}(u)+1\\
\mathtt{K}(\lambda x.t) & := & \mathtt{K}(t)+1 & \mathtt{K}(t[x/u]) & := & \mathtt{K}(t)\cdot\mathtt{K}(u)
\end{array}$$

In order to extend $\mathtt{K}(\_)$ on terms to $\mathbb{K}(\_)$ on labelled terms we define a special measure for $\lambda\mathtt{ex}$-strongly normalising terms. Thus, given $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$, let us consider

$$\phi(t) := 1 + \eta_{\lambda\mathtt{ex}}(t) + \mathtt{maxK}_{\lambda\mathtt{ex}}(t), \text{ where } \mathtt{maxK}_{\lambda\mathtt{ex}}(t) := \mathtt{max}\{\mathtt{K}(t') \mid t \to^*_{\lambda\mathtt{ex}} t'\}$$

Notice that $\phi$ is well-defined since $\lambda\mathtt{ex}$-strongly normalising terms have only a finite set of reducts. Notice also that $\phi(t) \geq 2$ for every term $t$. Moreover, $t \to_{\lambda\mathtt{ex}} t'$ implies $\eta_{\lambda\mathtt{ex}}(t) > \eta_{\lambda\mathtt{ex}}(t')$ and $\mathtt{maxK}_{\lambda\mathtt{ex}}(t) \geq \mathtt{maxK}_{\lambda\mathtt{ex}}(t')$ so that $\phi(t) > \phi(t')$.

We can now consider the following function $\mathbb{K}(\_)$ on labelled terms.

$$\begin{array}{lcllcl}
\mathbb{K}(x) & := & 1 & \mathbb{K}(tu) & := & \mathbb{K}(t)+\mathbb{K}(u)+1\\
\mathbb{K}(\lambda x.t) & := & \mathbb{K}(t)+1 & \mathbb{K}(t[x/u]) & := & \mathbb{K}(t)\cdot\mathbb{K}(u)\\
 & & & \mathbb{K}(t[\![x/u]\!]) & := & \mathbb{K}(t)\cdot\phi(u)
\end{array}$$

**Lemma 4.6.** *Let $t, u$ be $\mathbb{S}$-labelled terms and let $z \notin \mathbb{S}$. Then,*
(1) $t =_{\alpha,\mathtt{C},\underline{\mathtt{C}}} u$ *implies* $\mathtt{af}_z(t) = \mathtt{af}_z(u)$, $\mathtt{dep}(t) = \mathtt{dep}(u)$ *and* $\mathbb{K}(t) = \mathbb{K}(u)$.
(2) $t \rightarrow_{\underline{\mathtt{Comp}}} u$ *implies* $\mathtt{af}_z(t) = \mathtt{af}_z(u)$ *and* $\mathtt{dep}(t) > \mathtt{dep}(u)$.
(3) $t \rightarrow_{\underline{\mathtt{x}}\setminus\underline{\mathtt{Comp}}} u$ *implies* $\mathtt{af}_z(t) \geq \mathtt{af}_z(u)$, $\mathtt{dep}(t) \geq \mathtt{dep}(u)$ *and* $\mathbb{K}(t) > \mathbb{K}(u)$.

*Proof.* By induction on reduction. Notice that $\mathtt{af}_z(t) > \mathtt{af}_z(u)$ holds for example for $t = t_1[x/u_1] \rightarrow_{\underline{\mathtt{Gc}}} t_1[x/u'_1] = u$, where $u_1 \rightarrow_{\underline{\mathtt{Gc}}} u'_1$, $z \in \mathtt{fv}(u_1)$ and $z \notin \mathtt{fv}(u'_1)$. Similarly, $\mathtt{dep}(t) = \mathtt{dep}(u)$ holds for example for $t \rightarrow_{\underline{\mathtt{Var}}} u$, and $\mathtt{dep}(t) > \mathtt{dep}(u)$ holds for example for $t = t_2[\![z/u_2]\!] \rightarrow_{\underline{\mathtt{Gc}}} t'_2[\![z/u_2]\!] = u$, where $t_2 \rightarrow_{\underline{\mathtt{Gc}}} t'_2$ and $\mathtt{af}_z(t_2) > \mathtt{af}_z(t'_2)$. □

**Lemma 4.7.** *The reduction relation* $\underline{\mathtt{ex}}$ *(and thus also* $\underline{\mathtt{x}}$*) is terminating.*

*Proof.* Since $t \rightarrow_{\underline{\mathtt{ex}}} u$ implies $\langle \mathtt{dep}(t), \mathbb{K}(t)\rangle >_{lex} \langle \mathtt{dep}(u), \mathbb{K}(u)\rangle$ by Lemma 4.6 and $>_{lex}$ is a well-founded relation, then $\underline{\mathtt{ex}}$ terminates. □

**Lemma 4.8.** *The reduction relation* $\lambda\underline{\mathtt{ex}}^i$ *is terminating.*

*Proof.* Lemma 4.6(1) guarantees that $t =_{\mathtt{e}\cup\underline{\mathtt{e}}} t'$ implies $\langle \mathtt{dep}(t), \mathbb{K}(t)\rangle = \langle \mathtt{dep}(t'), \mathbb{K}(t')\rangle$. We now show that $t \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'$ implies $\mathtt{af}_z(t) \geq \mathtt{af}_z(t')$ for $z \notin \mathbb{S}$ and $\langle \mathtt{dep}(t), \mathbb{K}(t)\rangle >_{lex} \langle \mathtt{dep}(t'), \mathbb{K}(t')\rangle$. We proceed by induction on $\rightarrow_{\lambda\underline{\mathtt{x}}^i}$.

- If $t = u[\![x/v]\!] \rightarrow_{\lambda\underline{\mathtt{x}}^i} u[\![x/v']\!] = t'$ comes from $v \rightarrow_{\mathtt{Bx}} v'$, then $\mathtt{af}_z(t) = \mathtt{af}_z(u) = \mathtt{af}_z(t')$, $\mathtt{dep}(t) = \mathtt{dep}(u) + \mathtt{af}_x(u) = \mathtt{dep}(t')$ and $\mathbb{K}(t) = \mathbb{K}(u)\cdot\phi(v) > \mathbb{K}(u)\cdot\phi(v') = \mathbb{K}(t')$.
- If $t \rightarrow_{\lambda\underline{\mathtt{x}}^i} t'$ comes from $t \rightarrow_{\underline{\mathtt{x}}} t'$, then conclude using Lemma 4.6.
- If $t = u[\![x/v]\!] \rightarrow_{\lambda\underline{\mathtt{x}}^i} u'[\![x/v]\!] = t'$ or $t = u[x/v] \rightarrow_{\lambda\underline{\mathtt{x}}^i} u'[x/v] = t'$ or $t = v[x/u] \rightarrow_{\lambda\underline{\mathtt{x}}^i} v[x/u'] = t'$ or $t = uv \rightarrow_{\lambda\underline{\mathtt{x}}^i} u'v = t'$ or $t = vu \rightarrow_{\lambda\underline{\mathtt{x}}^i} vu' = t'$ or $t = \lambda x.u \rightarrow_{\lambda\underline{\mathtt{x}}^i} \lambda x.u' = t'$ comes from $u \rightarrow_{\lambda\underline{\mathtt{x}}^i} u'$, then the property trivially holds by the i.h. □

## 5. The **IE** Property

This section is devoted to show the **IE** Property, this is done by using the labelled terms introduced in Section 4 as an intermediate formalism between $t\{x/u\}\overline{v_n}$ and $t[x/u]\overline{v_n}$. More precisely, we split the **IE** Property in two different steps:

- Show that $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$ & $t\{x/u\}\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ imply $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\mathtt{ex}}}$.
- Show that $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\mathtt{ex}}}$ implies $t[x/u]\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$.

In order to relate reduction steps in $\lambda\underline{\mathtt{ex}}$ to reduction steps in $\lambda\mathtt{ex}$ we use a function $\mathtt{xc}$ from labelled terms to terms which computes all the labelled substitutions as follows:

| | | |
|---|---|---|
| $\mathtt{xc}(x)$ | $:=$ | $x$ |
| $\mathtt{xc}(tu)$ | $:=$ | $\mathtt{xc}(t)\mathtt{xc}(u)$ |
| $\mathtt{xc}(\lambda y.t)$ | $:=$ | $\lambda y.\mathtt{xc}(t)$ |
| $\mathtt{xc}(t[x/u])$ | $:=$ | $\mathtt{xc}(t)[x/\mathtt{xc}(u)]$ |
| $\mathtt{xc}(t[\![x/v]\!])$ | $:=$ | $\mathtt{xc}(t)\{x/v\}$ |

Notice that $\mathtt{xc}(t) = t$ if $t$ is a term.

**Lemma 5.1.** *Let $t, t'$ be labelled terms. If $t \rightarrow_{\underline{\mathtt{ex}}} t'$, then $\mathtt{xc}(t) = \mathtt{xc}(t')$.*

*Proof.* By induction on $t \rightarrow_{\underline{\mathtt{ex}}} t'$. The interesting case is $t = s[x/u][\![y/v]\!] =_{\underline{\mathtt{C}}} s[\![y/v]\!][x/u] = t'$, with $y \notin \mathtt{fv}(u)$ & $x \notin \mathtt{fv}(v)$. The term $\mathtt{xc}(t)$ is equal to $\mathtt{xc}(s)[x/\mathtt{xc}(u)]\{y/v\} = \mathtt{xc}(s)\{y/v\}[x/\mathtt{xc}(u)] = \mathtt{xc}(t')$. □

**Lemma 5.2** (Projecting $\lambda\underline{\texttt{ex}}$)**.** *Let $t, t'$ be labelled terms. Then,*
(1) $t =_{\alpha,\texttt{C},\underline{\texttt{C}}} t'$ *implies* $\texttt{xc}(t) = \texttt{xc}(t')$.
(2) $t \to_{\lambda\underline{\texttt{x}}^i} t'$ *implies* $\texttt{xc}(t) \to^*_{\lambda\texttt{ex}} \texttt{xc}(t')$.
(3) $t \to_{\lambda\underline{\texttt{x}}^e} t'$ *implies* $\texttt{xc}(t) \to^+_{\lambda\texttt{ex}} \texttt{xc}(t')$.

*Proof.*
(1) By induction on the conversion relation.
(2) Internal reduction:
- If $u[\![x/v]\!] \to_{\lambda\underline{\texttt{x}}^i} u[\![x/v']\!]$ comes from $v \to_{\texttt{Bx}} v'$, then
$\texttt{xc}(u[\![x/v]\!]) = \texttt{xc}(u)\{x/v\} \to^*_{\lambda\texttt{ex}\ (L.\ 2.1)} \texttt{xc}(u)\{x/v'\} = \texttt{xc}(u[\![x/v']\!])$.
- If $t \to_{\lambda\underline{\texttt{x}}^i} t'$ comes from $t \to_{\underline{\texttt{x}}} t'$ (so that also $t \to_{\underline{\texttt{ex}}} t'$), then Lemma 5.1 gives $\texttt{xc}(t) = \texttt{xc}(t')$.
- If $uv \to_{\lambda\underline{\texttt{x}}^i} u'v$ where $u \to_{\lambda\underline{\texttt{x}}^i} u'$, then
$\texttt{xc}(uv) = \texttt{xc}(u)\texttt{xc}(v) \to^*_{\lambda\texttt{ex}\ (i.h.)} \texttt{xc}(u')\texttt{xc}(v) = \texttt{xc}(u'v)$.
- If $u[\![x/v]\!] \to_{\lambda\underline{\texttt{x}}^i} u'[\![x/v]\!]$ where $u \to_{\lambda\underline{\texttt{x}}^i} u'$, then
$\texttt{xc}(u[\![x/v]\!]) = \texttt{xc}(u)\{x/v\} \to^*_{\lambda\texttt{ex}\ (i.h.\ \&\ L.\ 2.1)} \texttt{xc}(u')\{x/v\} = \texttt{xc}(u'[\![x/v]\!])$.
- The other cases are similar since $\texttt{xc}$ does not alter application, lambda and substitution.

(3) External reduction:
- If $t \to_{\lambda\underline{\texttt{x}}^e} t'$ comes from a reduction $t \to_{\texttt{Bx}} t'$ which occurs outside a labelled substitution, then $\texttt{xc}(t) \to^+_{\lambda\texttt{ex}} \texttt{xc}(t')$ can be shown by induction on $t \to_{\texttt{Bx}} t'$ using Lemma 2.1.
- If $tu \to_{\lambda\underline{\texttt{x}}^e} t'u$, $ut \to_{\lambda\underline{\texttt{x}}^e} ut'$, $\lambda x.t \to_{\lambda\underline{\texttt{x}}^e} \lambda x.t'$, $t[x/u] \to_{\lambda\underline{\texttt{x}}^e} t'[x/u]$ or $u[x/t] \to_{\lambda\underline{\texttt{x}}^e} u[x/t']$ comes from $t \to_{\lambda\underline{\texttt{x}}^e} t'$, then $\texttt{xc}(t) \to^+_{\lambda\texttt{ex}} \texttt{xc}(t')$ by the i.h. and thus the property holds by definition of $\texttt{xc}$ and the fact that $\texttt{xc}$ does not alter application, lambda and substitution.
- If $t[\![x/u]\!] \to_{\lambda\underline{\texttt{x}}^e} t'[\![x/u]\!]$ comes from $t \to_{\lambda\underline{\texttt{x}}^e} t'$, then
$\texttt{xc}(t[\![x/u]\!]) = \texttt{xc}(t)\{x/u\} \to^+_{\lambda\texttt{ex}\ (i.h.\ \&\ L.\ 2.1)} \texttt{xc}(t')\{x/u\} = \texttt{xc}(t'[\![x/u]\!])$. □

**Lemma 5.3.** *Let $t$ be a labelled term. If $\texttt{xc}(t) \in \mathcal{SN}_{\lambda\texttt{ex}}$, then $t \in \mathcal{SN}_{\lambda\underline{\texttt{ex}}}$.*

*Proof.* We apply the Abstract Theorem A.2 in the Appendix A by taking $\mathcal{A}_1 = \lambda\underline{\texttt{ex}}^i$, $\mathcal{A}_2 = \lambda\underline{\texttt{ex}}^e$, $\mathcal{A} = \lambda\texttt{ex}$ and $u\ \mathcal{R}\ U$ iff $\texttt{xc}(u) = U$. Lemma 5.2 guarantees properties **P1** and **P2** and Lemma 4.8 guarantees property **P3**. We then get that $\texttt{xc}(t) \in \mathcal{SN}_{\lambda\texttt{ex}}$ implies $t \in \mathcal{SN}_{\lambda\underline{\texttt{ex}}^i \cup \lambda\underline{\texttt{ex}}^e}$, which is exactly $\mathcal{SN}_{\lambda\underline{\texttt{ex}}}$ by Lemma 4.3. We thus conclude. □

**Corollary 5.4.** *Let $t, u, \overline{v_n}$ be terms. If $u \in \mathcal{SN}_{\lambda\texttt{ex}}$ & $t\{x/u\}\overline{v_n} \in \mathcal{SN}_{\lambda\texttt{ex}}$, then $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\texttt{ex}}}$.*

*Proof.* Take $\mathbb{S} = \texttt{fv}(u)$. The hypothesis $u \in \mathcal{SN}_{\lambda\texttt{ex}}$ allows us to construct the $\mathbb{S}$-labelled term $t[\![x/u]\!]\overline{v_n}$. Moreover, $\texttt{xc}(t) = t$ so that $\texttt{xc}(t[\![x/u]\!]\overline{v_n}) = t\{x/u\}\overline{v_n}$ and we thus conclude by Lemma 5.3. □

Labelled terms can be unlabelled in such a way that $\lambda\texttt{ex}$-reduction on unlabelled labelled terms can be simulated by $\lambda\underline{\texttt{ex}}$-reduction.

**Definition 5.5** (Unlabelling)**.** *Unlabelling* of labelled terms is defined by induction.

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

Notice that $\mathtt{fv}(t) = \mathtt{fv}(\mathtt{U}(t))$.

**Lemma 5.6.** *Let $t \in \mathcal{L}_{\mathbb{S}}$ s.t. $\mathtt{U}(t) \to_{\lambda\mathtt{ex}} t'_1$. Then $\exists\ t_1 \in \mathcal{L}_{\mathbb{S}}$ s.t. $t \to_{\lambda\underline{\mathtt{ex}}} t_1$ and $\mathtt{U}(t_1) = t'_1$.*

*Proof.* By induction on $\to_{\lambda\mathtt{ex}}$ and case analysis. The interesting cases are the following.

- $t = u[x/v][\![y/w]\!]$ where $y \in \mathtt{fv}(v)$, and
$$\begin{array}{lll}
\mathtt{U}(u[x/v][\![y/w]\!]) & = & \\
\mathtt{U}(u)[x/\mathtt{U}(v)][y/w] & \to_{\mathtt{Comp}} & \mathtt{U}(u)[y/w][x/\mathtt{U}(v)[y/w]] \quad = t'_1
\end{array}$$
We then let $t_1 = u[\![y/w]\!][x/v[\![y/w]\!]]$ so that $\mathtt{U}(t_1) = t'_1$ and $t \to_{\underline{\mathtt{Comp}}} t_1$.
- $t = u[x/v][\![y/w]\!]$ where $y \notin \mathtt{fv}(v)$, and
$$\begin{array}{lll}
\mathtt{U}(u[x/v][\![y/w]\!]) & = & \\
\mathtt{U}(u)[x/\mathtt{U}(v)][y/w] & =_{\mathtt{C}} & \mathtt{U}(u)[y/w][x/\mathtt{U}(v)] \quad = t'_1
\end{array}$$
We then let $t_1 = u[\![y/w]\!][x/v]$ so that $\mathtt{U}(t_1) = t'_1$ and $t =_{\underline{\mathtt{C}}} t_1$.
- $t = u[\![y/w]\!][x/v]$. By $\alpha$-conversion we can always choose $x \notin \mathbb{S}$, which is a fixed set of variables, so that we necessarily have $x \notin \mathtt{fv}(w)$ since $\mathtt{fv}(w) \subseteq \mathbb{S}$ by construction. Now, consider
$$\begin{array}{lll}
\mathtt{U}(u[\![y/w]\!][x/v]) & = & \\
\mathtt{U}(u)[y/w][x/\mathtt{U}(v)] & =_{\mathtt{C}} & \mathtt{U}(u)[x/\mathtt{U}(v)][y/w] \quad = t'_1
\end{array}$$
We then let $t_1 = u[x/v][\![y/w]\!]$ so that $\mathtt{U}(t_1) = t'_1$ and $t =_{\underline{\mathtt{C}}} t_1$.
- $t = u[\![x_1/v_1]\!][\![x_2/v_2]\!]$. Again, by $\alpha$-conversion we can assume $x_i \notin \mathbb{S}$ so that $x_i \notin \mathtt{fv}(v_j)$ since $\mathtt{fv}(v_i) \subseteq \mathbb{S}$ by construction. Now, consider
$$\begin{array}{llll}
\mathtt{U}(u[\![x_1/v_1]\!][\![x_2/v_2]\!]) & = & & \\
\mathtt{U}(u)[x_1/v_1][x_2/v_2] & =_{\mathtt{C}} & \mathtt{U}(u)[x_2/v_2][x_1/v_1] & = \\
& & \mathtt{U}(u[\![x_2/v_2]\!][\![x_1/v_1]\!]) & = t'_1
\end{array}$$
We then let $t_1 = u[\![x_2/v_2]\!][\![x_1/v_1]\!]$ so that $\mathtt{U}(t_1) = t'_1$ and $t =_{\underline{\mathtt{C}}} t_1$.

All the other cases are straightforward. □

**Lemma 5.7.** *Let $t \in \mathcal{L}_{\mathbb{S}}$. If $t \in \mathcal{SN}_{\lambda\underline{\mathtt{ex}}}$, then $\mathtt{U}(t) \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

*Proof.* We prove $\mathtt{U}(t) \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by induction on $\eta_{\lambda\underline{\mathtt{ex}}}(t)$. This is done by considering all the $\lambda\mathtt{ex}$-reducts of $\mathtt{U}(t)$ and using Lemma 5.6. □

Taking $\mathbb{S} = \mathtt{fv}(u)$ and transforming the term $s[x/u]\overline{u_n}$ into the $\mathbb{S}$-labelled term $s[\![x/u]\!]\overline{u_n}$ we have the following special case.

**Corollary 5.8.** *If $t[\![x/u]\!]\overline{v_n} \in \mathcal{SN}_{\lambda\underline{\mathtt{ex}}}$, then $t[x/u]\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

We can now conclude with the main property required in the proof of the Perpetuality Theorem:

**Lemma 5.9** (**IE** Property)**.** *Let $t, u, \overline{v_n}$ be terms. If $u \in \mathcal{SN}_{\lambda\mathtt{ex}}$ & $t\{x/u\}\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $t[x/u]\overline{v_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

*Proof.* By Corollaries 5.4 and 5.8. □

## 6. Intersection Types

The simply typed calculus is a typed lambda calculus whose only type connective is the function type. This makes it canonical, simple, and decidable [Tai67]. The simply typed lambda calculus enjoys the *$\beta$-strong normalisation* property stating that every $\beta$-reduction sequence starting with a typed $\lambda$-term terminates.

However, some intersection type disciplines [CDC78, CDC80] are more expressive and flexible than simple type systems in the sense that not only are typed $\lambda$-terms $\beta$-strongly normalising, but the converse also holds, thus giving a characterisation of the set of $\beta$-strongly normalising $\lambda$-terms.

Intersection types for calculi with explicit substitutions have been studied in [LLD+04, Kik07, KC]. Here, we apply this technique to the $\lambda\texttt{ex}$-calculus, and obtain a characterisation of the set of $\lambda\texttt{ex}$-strongly normalising terms by means of an intersection type system.

*Types* are built over a countable set of atomic symbols as follows:

$$A ::= \sigma \text{ (atomic)} \mid A \to A \mid A \cap A$$

An *environment* is a finite set of pairs of the form $x : A$. *Typing judgements* have the form $\Gamma \vdash t : A$ where $t$ is a term, $A$ is a type and $\Gamma$ is an environment. The *intersection type* system, called *System $\cap$*, is defined by means of the set of *typing rules* in Figure 3.

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\texttt{ax}) \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}\ (\texttt{app})$$

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

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B}\ (\cap\, \texttt{I}) \qquad \frac{\Gamma \vdash t : A_1 \cap A_2}{\Gamma \vdash t : A_i}\ (\cap\, \texttt{E})$$

Figure 3: System $\cap$: an intersection type discipline for terms

A *derivation* of a typing judgement $\Gamma \vdash t : A$, written $\Gamma \vdash_\cap t : A$, is a tree obtained by successive applications of the typing rules of the system $\cap$. A term $t$ is said to be $\cap$-*typable*, iff there is an environment $\Gamma$ and a type $A$ s.t. $\Gamma \vdash_\cap t : A$. Notice that every $\lambda$-term is $\cap$-*typable* iff there is an environment $\Gamma$ and a type $A$ s.t. $\Gamma \vdash_\cap t : A$ holds in the system which only contains the typing rules $\{\texttt{ax}, \texttt{abs}, \texttt{app}, \cap\,\texttt{I}, \cap\,\texttt{E}\}$ in Figure 3.

The well-known characterisation of the set of $\beta$-strongly normalising $\lambda$-terms reads now as follows:

**Theorem 6.1** ([Pot80])**.** *Let $t$ be a $\lambda$-term. Then $t$ is* $\cap$-typable *iff $t \in \mathcal{SN}_\beta$.*

A subtyping relation on intersection types is now specified by means of a preorder. This will be used to establish a Generation Lemma transforming any type derivation into a specific derivation depending only on the form of the term (and not on the type). Thus, the Generation Lemma turns out to be extremely useful to reason by induction on type derivations.

**Definition 6.2.** The relation $\ll$ on types is defined by the following axioms and rules

(1) $A \ll A$
(2) $A \cap B \ll A$ and $A \cap B \ll B$
(3) $A \ll B$ & $B \ll C$ implies $A \ll C$
(4) $A \ll B$ & $A \ll C$ implies $A \ll B \cap C$

**Lemma 6.3.** *If $\Gamma \vdash_\cap t : B$ and $B \ll A$, then $\Gamma \vdash_\cap t : A$.*

*Proof.* Let $\Gamma \vdash_\cap t : B$. We reason by induction on the definition of $B \ll A$.

**Case $B = A \ll A$:** Trivial.

**Case $B = A \cap C \ll A$ and $B = C \cap A \ll A$:** Use $\cap$ `E`.

**Case $B \ll C, C \ll A$:** Use (twice) the i.h. to get successively $\Gamma \vdash_\cap t : C$ and then $\Gamma \vdash_\cap t : A$.

**Case $B \ll B_1, B \ll B_2, A = B_1 \cap B_2$:** Use (twice) the i.h. to get $\Gamma \vdash_\cap t : B_1$ and $\Gamma \vdash_\cap t : B_2$, then apply $\cap$ `I`. □

We use the notation $\underline{n}$ for $\{1 \ldots n\}$ and $\cap_n A_i$ for $A_1 \cap \ldots \cap A_n$.

**Lemma 6.4.** *Let $\cap_n A_i \ll \cap_m B_j$, where none of the $A_i$ and $B_j$ is an intersection. Then for each $B_j$ there is $A_i$ s.t. $B_j = A_i$.*

*Proof.* By induction on the definition of $\cap_n A_i \ll \cap_m B_j$. Let $\cap_p C_k$ be some type where none of the $C_k$ is an intersection type.

**Case $\cap_n A_i \ll \cap_n A_i$:** Trivial.

**Case $\cap_m B_j \cap \cap_p C_k \ll \cap_m B_j$ and $\cap_p C_k \cap \cap_m B_j \ll \cap_m B_j$:** Trivial.

**Case $\cap_n A_i \ll \cap_p C_k, \cap_p C_k \ll \cap_m B_j$:** Applying the i.h. a first time we have for each $B_j$ a $C_k$ s.t. $B_j = C_k$. Applying the i.h. again we have for each $C_k$ a $A_i$ s.t. $C_k = A_i$. Thus we can conclude.

**Case $\cap_n A_i \ll B_1 \cap \ldots \cap B_k, \cap_n A_i \ll B_{k+1} \cap \ldots \cap B_m$:** By the i.h. we have for each $B_j, 1 \leq j \leq k$ a type $A_i$ s.t. $B_j = A_i$ and for each $B_j, k+1 \leq j \leq m$ a type $A_i$ s.t. $B_j = A_i$. Thus we can conclude. □

**Lemma 6.5** (Generation Lemma)**.**

(1) *$\Gamma \vdash_\cap x : A$ iff there is $x : B \in \Gamma$ and $B \ll A$.*
(2) *$\Gamma \vdash_\cap t[x/u] : A$ iff there exist $A_i, B_i$ ($i \in \underline{n}$) s.t. $\cap_n A_i \ll A$ and $\forall i \in \underline{n}, \Gamma \vdash_\cap u : B_i$ and $\Gamma, x : B_i \vdash_\cap t : A_i$.*
(3) *$\Gamma \vdash_\cap tu : A$ iff there exist $A_i, B_i$ ($i \in \underline{n}$) s.t. $\cap_n A_i \ll A$ and $\forall i \in \underline{n}, \Gamma \vdash_\cap t : B_i \to A_i$ and $\Gamma \vdash_\cap u : B_i$.*
(4) *$\Gamma \vdash_\cap \lambda x.t : A$ iff there exist $A_i, B_i$ ($i \in \underline{n}$) s.t. $\cap_n (A_i \to B_i) \ll A$ and $\forall i \in \underline{n}, \Gamma, x : A_i \vdash_\cap t : B_i$.*
(5) *$\Gamma \vdash_\cap \lambda x.t : B \to C$ iff $\Gamma, x : B \vdash_\cap t : C$.*

*Proof.* The right to left implications follow from the typing rules of the intersection type system $\cap$ and Lemma 6.3.

The left to right implication of the first four points are shown by induction on the typing derivation of the left part. We only show the two first points as the other ones are similar.

(1) Consider $\Gamma \vdash_\cap x : A$.
- Suppose the derivation is (`ax`) so that $x : A \in \Gamma$, then $B = A$.

- Suppose $A = C_1 \cap C_2$ and the root of the derivation is

$$\frac{\Gamma \vdash x : C_1 \quad \Gamma \vdash x : C_2}{\Gamma \vdash x : C_1 \cap C_2} (\cap\, \mathtt{I})$$

  By the i.h. there is $B_1 \ll C_1$ and $B_2 \ll C_2$ s.t. $x : B_1, x : B_2 \in \Gamma$, thus $B_1 = B_2$ and $B_1 \ll C_1 \cap C_2$ concludes the proof of this case.
- Suppose the root of the derivation is

$$\frac{\Gamma \vdash x : A \cap A'}{\Gamma \vdash x : A} (\cap\, \mathtt{E})$$

  By the i.h. there is $B \ll A \cap A'$ s.t. $x : B \in \Gamma$. By transitivity $B \ll A$ which concludes the proof of this case.
- There is no other possible case.

(2) Consider $\Gamma \vdash_\cap t[x/u] : A$.

- Suppose the root of the derivation is

$$\frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x/u] : A} (\mathtt{subs})$$

  then the property immediately holds by taking $n = 1$, $B_1 = B$ and $A_1 = A$.
- Suppose $A = C_1 \cap C_2$ and the root of the derivation is

$$\frac{\Gamma \vdash t[x/u] : C_1 \quad \Gamma \vdash t[x/u] : C_2}{\Gamma \vdash t[x/u] : C_1 \cap C_2} (\cap\, \mathtt{I})$$

  By the i.h. there are $A_i, B_i$ ($i \in \underline{n}$) s.t. $\cap_n A_i \ll C_1$ and $\Gamma \vdash_\cap u : B_i$ and $\Gamma, x : B_i \vdash_\cap t : A_i$ for all $i \in \underline{n}$. Also there are $A'_i, B'_i$ ($i \in \underline{n'}$) s.t. $\cap_{n'} A'_i \ll C_2$ and $\Gamma \vdash_\cap u : B'_i$ and $\Gamma, x : B'_i \vdash_\cap t : A'_i$ for all $i \in \underline{n'}$. Since $\cap_n A_i \cap \cap_{n'} A'_i \ll C_1 \cap C_2$, this concludes this case.
- Suppose the root of the derivation is

$$\frac{\Gamma \vdash t[x/u] : A \cap B}{\Gamma \vdash t[x/u] : A} (\cap\, \mathtt{E})$$

  By the i.h. there are $A_i, B_i$ ($i \in \underline{n}$) s.t. $\cap_n A_i \ll A \cap B$ and $\Gamma \vdash u : B_i$ and $\Gamma, x : B_i \vdash t : A_i$ for all $i \in \underline{n}$. Since $\cap_n A_i \ll A$, this concludes this case.

The left to right implication of point 5 follows from point 4 and Lemma 6.4. Indeed, if $\Gamma \vdash_\cap \lambda x.t : B \to C$, then point 4 gives $\Gamma, x : B_i \vdash_\cap t : C_i$ for $\cap_n(B_i \to C_i) \ll B \to C$. Lemma 6.4 gives $B \to C = B_j \to C_j$ for some $j \in \underline{n}$, thus $\Gamma, x : B \vdash_\cap t : C$. □

The rest of the section is now devoted to establish some connections between typable and strongly normalisable terms in the $\lambda\mathtt{ex}$-calculus.

**Definition 6.6.** The function $\mathtt{V}(\_)$ from terms to $\lambda$-terms is defined by induction as follows:

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

This function is compositional with respect to substitution:

**Lemma 6.7.** *Let $t, u$ be terms. Then* $\mathtt{V}(t)\{x/\mathtt{V}(u)\} = \mathtt{V}(t\{x/u\})$.

*Proof.* By induction on $t$. □

The function $\mathtt{V}(\_)$ does not modify typability.

**Lemma 6.8.** *Let $t$ be a term. Then $\Gamma \vdash_\cap \mathtt{V}(t) : A$ iff $\Gamma \vdash_\cap t : A$.*

*Proof.* By induction on $t$ using the Generation Lemma 6.5. □

**Theorem 6.9** (Typable Terms are SN)**.** *If $t$ is $\cap$-typable, then $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

*Proof.* By Lemma 6.8 the $\lambda$-term $\mathtt{V}(t)$ is also $\cap$-typable so that the left to right implication of Theorem 6.1 gives $\mathtt{V}(t) \in \mathcal{SN}_\beta$ and then the PSN Property (Theorem 3.6) gives $\mathtt{V}(t) \in \mathcal{SN}_{\lambda\mathtt{ex}}$. Since $\mathtt{V}(t) \to^+_{\mathtt{B}} t$ (a straightforward induction on $t$), then $t$ is necessarily in $\mathcal{SN}_{\lambda\mathtt{ex}}$. □

We now complete the picture by showing that the intersection type discipline for terms gives a characterisation of $\lambda\mathtt{ex}$-strongly normalising terms.

**Lemma 6.10.** *Let $t$ be a term s.t. $\mathtt{V}(t) \to_\beta t'_1$. Then, $\exists\ t_1$ s.t. $t \to^+_{\lambda\mathtt{ex}} t_1$ and $t'_1 = \mathtt{V}(t_1)$.*

*Proof.* By induction on the reduction step $\mathtt{V}(t) \to_\beta t'_1$.

- If $\mathtt{V}((\lambda x.u)\ v) = (\lambda x.\mathtt{V}(u))\mathtt{V}(v) \to_\beta \mathtt{V}(u)\{x/\mathtt{V}(v)\}$, then let $t_1 = u\{x/v\}$. We have $(\lambda x.u)\ v \to_{\mathtt{B}} u[x/v] \to^+_{\lambda\mathtt{ex}\ (L.\ 2.2)} u\{x/v\}$ and we conclude by Lemma 6.7.
- If $\mathtt{V}(u[x/v]) = (\lambda x.\mathtt{V}(u))\mathtt{V}(v) \to_\beta \mathtt{V}(u)\{x/\mathtt{V}(v)\}$, then again we conclude by letting $t_1 = u\{x/v\}$.
- If $\mathtt{V}(u[x/v]) = (\lambda x.\mathtt{V}(u))\mathtt{V}(v) \to_\beta (\lambda x.u'_1)\mathtt{V}(v)$, where $\mathtt{V}(u) \to_\beta u'_1$ then the i.h. gives $u_1$ s.t. $u'_1 = \mathtt{V}(u_1)$ and $u \to^+_{\lambda\mathtt{ex}} u_1$. Let $t_1 = u_1[x/v]$. We have $u[x/v] \to^+_{\lambda\mathtt{ex}} u_1[x/v]$ and $(\lambda x.u'_1)\ \mathtt{V}(v) = \mathtt{V}(u_1[x/v])$.
- If $\mathtt{V}(u[x/v]) = (\lambda x.\mathtt{V}(u))\mathtt{V}(v) \to_\beta (\lambda x.\mathtt{V}(u))v'_1$, where $\mathtt{V}(v) \to_\beta v'_1$, then proceed as in the previous one.
- All the other cases are straightforward. □

**Theorem 6.11** (SN Terms are Typable)**.** *If $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $t$ is $\cap$-typable.*

*Proof.* Let $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$. One first shows that $\mathtt{V}(t) \in \mathcal{SN}_\beta$ by induction on $\eta_{\lambda\mathtt{ex}}(t)$. This is done by considering all the $\beta$-reducts of $\mathtt{V}(t)$ and using Lemma 6.10.

Now, $\mathtt{V}(t) \in \mathcal{SN}_\beta$ implies that $\mathtt{V}(t)$ is $\cap$-typable by the right to left implication of Theorem 6.1. Finally, Lemma 6.8 allows to conclude that $t$ is $\cap$-typable. □

**Corollary 6.12.** *Let $t$ be a term. Then $t$ is $\cap$-typable iff $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

We conclude this section by focusing on the particular case of the *simply typed $\lambda\mathtt{ex}$-calculus* : types are only built over atomic symbols and functional types so that the type system only contains the typing rules $\{\mathtt{ax}, \mathtt{abs}, \mathtt{app}, \mathtt{subs}\}$ in Figure 3. Since every simply typed $\lambda$-term is $\beta$-strongly normalising (this is the restriction of the left to right implication of Theorem 6.1 to simple types), then in particular:

**Corollary 6.13** (Simply Typed Terms are SN - First Proof)**.** *Simply typed $\lambda\mathtt{ex}$-calculus is $\lambda\mathtt{ex}$-strongly normalising.*

This proof depends however on previous results by [Pot80]. Another self-contained argument can be given by means of the arithmetical technique [vD77], and is extremely short.

**Lemma 6.14.** *If $t^A, u^B \in \mathcal{SN}_{\lambda\mathtt{ex}}$, then $t\{x^B/u^B\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$.*

*Proof.* By induction on the lexicographic triple $\langle B, \eta_{\lambda\mathtt{ex}}(t), t\rangle$.

- $t = x$. Then $x\{x/u\} = u \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by the hypothesis.
- $t = y\overline{v_n}$ with $x \neq y$ and $n \geq 0$. The i.h. gives $v_i\{x/u\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ since $\eta_{\lambda\mathtt{ex}}(v_i)$ decreases and $v_i$ is strictly smaller than $t$. Then we conclude by Definition 3.4 and Proposition 3.5.
- $t = xv\overline{v_n}$. The i.h. gives $V = v\{x/u\}$ and $V_i = v_i\{x/u\}$ in $\mathcal{SN}_{\lambda\mathtt{ex}}$. We show $t\{x/u\} = uV\overline{V_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by induction on $\eta_{\lambda\mathtt{ex}}(u) + \eta_{\lambda\mathtt{ex}}(V) + \Sigma_{i\in 1...n}\ \eta_{\lambda\mathtt{ex}}(V_i)$. For that, it is sufficient to show that all its reducts are in $\mathcal{SN}_{\lambda\mathtt{ex}}$. If the reduction takes place in a subterm of $u, V, \overline{V_n}$, then we conclude by the i.h. Otherwise, suppose $u = \lambda y.U$ and $(\lambda y.U)V\overline{V_n} \to U[y/V]\overline{V_n}$. Then $\mathtt{type}(V) = \mathtt{type}(v) < \mathtt{type}(u) = \mathtt{type}(x)$ so that $U\{y/V\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by the i.h. Let us write $U\{y/V\}\overline{V_n} = (z\overline{V_n})\{z/U\{y/V\}\}$. We have $\mathtt{type}(U\{y/V\}) = \mathtt{type}(U) < \mathtt{type}(u)$ so that again by the i.h. we get $U\{y/V\}\overline{V_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$. We conclude $U[y/V]\overline{V_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by Definition 3.4 and Proposition 3.5.
- $t = \lambda y.v$. Then $v\{x/u\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by the i.h. and thus $t\{x/u\} = \lambda x.v\{x/u\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ follows from Definition 3.4 and Proposition 3.5.
- $t = (\lambda y.s)v\overline{v_n}$. The i.h. gives $S = s\{x/u\}$, $V = v\{x/u\}$ and $V_i = v_i\{x/u\}$ in $\mathcal{SN}_{\lambda\mathtt{ex}}$. To show $t\{x/u\} = (\lambda y.S)V\overline{V_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ we reason by induction on $\eta_{\lambda\mathtt{ex}}(S) + \eta_{\lambda\mathtt{ex}}(V) + \Sigma_{i\in 1...n}\ \eta_{\lambda\mathtt{ex}}(V_i)$. For that, it is sufficient to show that all its reducts are in $\mathcal{SN}_{\lambda\mathtt{ex}}$. If the reduction takes place in a subterm of $(\lambda y.S), V, \overline{V_n}$, we conclude by the i.h. Otherwise suppose $(\lambda y.S)V\overline{V_n} \to S[y/V]\overline{V_n}$. Take $T = s[y/v]\overline{v_n}$. Since $\eta_{\lambda\mathtt{ex}}(T) < \eta_{\lambda\mathtt{ex}}(t)$, then the i.h. gives $T\{x/u\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$. But $S[y/V]\overline{V_n} = T\{x/u\}$ so that $S[y/V]\overline{V_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$.
- $t = s[y/v]\overline{v_n}$. The i.h. gives $S = s\{x/u\}$ and $V = v\{x/u\}$ and $V_i = v_i\{x/u\}$ are in $\mathcal{SN}_{\lambda\mathtt{ex}}$. They are also typed. We claim $t\{x/u\} = S[y/V]\overline{V_n} \in \mathcal{SN}_{\lambda\mathtt{ex}}$. The perpetual strategy gives
$$t\{x/u\} = S[y/V]\overline{V_n} \rightsquigarrow S\{y/V\}\overline{V_n}$$
This last term can be written as $T\{x/u\}$ where $T = s\{y/v\}\ \overline{v_n}$. Since $\eta_{\lambda\mathtt{ex}}(T) < \eta_{\lambda\mathtt{ex}}(t)$, then the i.h. gives $T\{x/u\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$ and thus Theorem 3.3 gives $S[y/V]\overline{V_n}$ in $\mathcal{SN}_{\lambda\mathtt{ex}}$. □

**Corollary 6.15** (Simply Typed Terms are SN - Second Proof)**.** *Simply typed $\lambda\mathtt{ex}$-calculus is $\lambda\mathtt{ex}$-strongly normalising.*

*Proof.* Let $t$ be a simply typed term. We reason by induction on the structure of $t$. The cases $t = x$ and $t = \lambda x.u$ are straightforward. If $t = uv$, then $u, v$ are typed so that $u, v \in \mathcal{SN}_{\lambda\mathtt{ex}}$ by the i.h. We write $t = (zv)\{z/u\}$, where $zv$ is $\mathcal{SN}_{\lambda\mathtt{ex}}$ by Definition 3.4. The term $zv$ is also appropriately typed. Lemma 6.14 then gives $t \in \mathcal{SN}_{\lambda\mathtt{ex}}$. If $t = u[x/v]$, then $u, v$ are typed and by the i.h. $u, v \in \mathcal{SN}_{\lambda\mathtt{ex}}$ so that Lemma 6.14 gives $u\{x/v\} \in \mathcal{SN}_{\lambda\mathtt{ex}}$. Definition 3.4 and Proposition 3.5 allow us to conclude $u[x/v] \in \mathcal{SN}_{\lambda\mathtt{ex}}$. □

## 7. Deriving Strong Normalisation for Other Related Calculi

We now informally discuss how strong normalisation of other calculi with ES (having or not safe composition) can be derived from strong normalisation of $\lambda\mathtt{ex}$.

- The $\lambda\mathtt{x}$-calculus [Lin86, Lin92, Ros92] is just a sub-calculus of $\lambda\mathtt{ex}$, with no equation and no composition rule. Thus, the fact that $t \to_{\lambda\mathtt{x}} t'$ implies $t \to^+_{\lambda\mathtt{ex}} t'$ is straightforward. Since simply typed terms in both calculi are the same, we thus deduce that typed terms are $\lambda\mathtt{x}$-strongly normalising.
- The $\lambda\mathtt{es}$-calculus [Kes07] can be seen as a refinement of $\lambda\mathtt{ex}$, where propagation of substitution with respect to application and substitution is done in a controlled way. We refer the reader to [Kes07] for details on the rules. The fact that $t \to_{\lambda\mathtt{es}} t'$ implies $t \to^+_{\lambda\mathtt{ex}} t'$ is

straightforward. Simply typed terms in both calculi are the same, we thus deduce that typed terms are $\lambda\texttt{es}$-strongly normalising.

- Milner's calculus with explicit *partial* substitution [Mil06], called $\lambda_{sub}$, is able to encode $\lambda$-calculus in terms of a bigraphical reactive system. The operational semantics of $\lambda_{sub}$ is given by reduction rules which only propagate a substitution of the form $[x/u]$ on one occurrence of the variable $x$ at a time (see for example [Mil06] for details). In [KC] it is shown that there exists a translation $\texttt{T}$ from terms to terms such that $t \to_{\lambda_{sub}} t'$ implies $\texttt{T}(t) \to^+_{\lambda\texttt{es}} \texttt{T}(t')$. Since simply typed terms in both calculi are the same, we conclude that typed terms are $\lambda_{sub}$-strongly normalising from the previous point.
- A $\lambda$-calculus with implicit *partial* $\beta$-reduction, written here $\lambda_{\beta_p}$, appears in [dB87]. Its syntax is the one of the pure $\lambda$-calculus (so that there is no explicit substitution operator) and its semantics is similar to that of $\lambda_{sub}$ since arguments are consumed on only one occurrence at a time. Similarly to [KC] one can define a translation $\texttt{T}$ from $\lambda$-terms to terms such that one-step reduction in $\lambda_{\beta_p}$ is projected into at least one-step reduction in $\lambda_{sub}$. Since simply typed $\lambda$-terms translate to simply typed terms, then typed $\lambda$-terms are $\lambda_{\beta_p}$-strongly normalising from the previous point.
- David and Guillaume [DG01] defined a calculus with *labels*, called $\lambda_{ws}$, which allows *controlled* composition of ES without losing PSN. The calculus $\lambda_{ws}$ has a strong form of composition which is safe but not full. Its simply typed named notation can be translated into simply typed terms in such a way that one-step reduction in $\lambda_{ws}$ implies at least one-step reduction in $\lambda\texttt{ex}$. Thus, SN for typed terms in $\lambda_{ws}$ is a consequence of SN for typed $\lambda\texttt{ex}$.
- A calculus with a safe notion of composition in director string notation is defined in [SFM03]. The named version of this calculus can be understood as the $\lambda\texttt{x}$-calculus together with a composition rule of the form:

  $$t[x/u][y/v] \to t[x/u[y/v]] \text{ if } y \in \texttt{fv}(u) \ \&\ y \notin \texttt{fv}(t)$$

  This composition rule can be easily simulated by the rules $\texttt{Comp}$ and $\texttt{Gc}$ of the $\lambda\texttt{ex}$-calculus so that the whole calculus can be simulated by $\lambda\texttt{ex}$. As a consequence, simply typed terms turn out to be strongly normalising.
- The $\lambda\texttt{esw}$-calculus [Kes07] was used as a technical tool to show that $\lambda\texttt{es}$ enjoys PSN. The syntax extends terms with weakening constructors so that it is straightforward to define a translation $\texttt{T}$ from $\lambda\texttt{esw}$-terms to terms which forgets these weakening operators. The reduction relation $\lambda\texttt{esw}$ can be split into an equational system $\mathcal{E}$ and two rewriting relations $\mathcal{L}_1$ and $\mathcal{L}_2$ s.t.
  (1) If $t =_{\mathcal{E}} t'$ or $t \to_{\mathcal{L}_1} t'$ then $\texttt{T}(t) =_{\texttt{C}} \texttt{T}(t')$
  (2) If $t \to_{\mathcal{L}_2} t'$ then $\texttt{T}(t) \to^+_{\lambda\texttt{ex}} \texttt{T}(t')$

  The reduction relation generated by the rules $\mathcal{L}_1$ modulo the equations $\mathcal{E}$ can be easily shown to be terminating. Also, simply typed $\lambda\texttt{esw}$-terms trivially translate via $\texttt{T}$ to simply typed terms. Thus, the Abstract Theorem given in the Appendix A allows us to conclude that typed $\lambda\texttt{esw}$-terms are $\lambda\texttt{esw}$-strongly normalising.

## 8. Confluence

In this section we study confluence of the $\lambda\texttt{ex}$-calculus. More precisely, we show confluence of the relation $\to_{\lambda\texttt{ex}}$ on *metaterms*, which are terms containing *metavariables* denoting *incomplete* programs/proofs in a higher-order framework [Hue76]. 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. We thus specify metavariables as follows. We consider a countable set of *raw* metavariables, denoted $\mathbb{X}, \mathbb{Y}, \ldots$. To each raw metariable $\mathbb{X}$, we associate a set of variables $\Delta$, thus yielding a *decorated* metavariable denoted by $\mathbb{X}_\Delta$. Thus for example $\mathbb{X}_{x,y,z}$ and $\mathbb{Y}_{x,z}$ are decorated metavariables. This decoration says nothing about the *structure* of the incomplete proof itself but is sufficient to guarantee that different occurrences of the *same* metavariable are never instantiated by different metaterms.

The set of *metaterms* is defined by the following grammar.

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

Notice that terms are in particular metaterms.

We extend the notion of *free variables* to *metaterms* by $\mathtt{fv}(\mathbb{X}_\Delta) := \Delta$. Thus, $\alpha$-conversion turns out to be perfectly well-defined on metaterms by extending the renaming of bound variables to the decoration sets. Thus for example $\lambda x.\mathbb{Y}_x\mathbb{X}_{x,y} =_\alpha \lambda z.\mathbb{Y}_z\mathbb{X}_{z,y}$.

*Meta-substitution* on *metaterms* extends that on terms by adding two new cases:

$$\begin{array}{lcll} \mathbb{X}_\Delta\{x/v\} & := & \mathbb{X}_\Delta & \text{if } x \notin \Delta \\ \mathbb{X}_\Delta\{x/v\} & := & \mathbb{X}_\Delta[x/v] & \text{if } x \in \Delta \end{array}$$

**Lemma 8.1.** *Let $t, u$ be metaterms. Then $t\{x/u\} = t$ if $x \notin \mathtt{fv}(t)$.*

*Proof.* By induction on $t$. □

The following property holds for metaterms.

**Lemma 8.2** (Composition Lemma)**.** *Let $t, u, v$ be metaterms and let $x, y$ s.t. $x \neq y$ and $x \notin \mathtt{fv}(v)$. Then $t\{x/u\}\{y/v\} =_{\mathtt{e}} t\{y/v\}\{x/u\{y/v\}\}$.*

*Proof.* By induction on metaterms using Lemma 8.1. Notice that $=_{\mathtt{e}}$ is needed for the case where $t$ is a metavariable. □

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

Reduction on terms and metaterms enjoys stability by substitution and full composition.

**Lemma 8.3** (Stability of Reduction of Metaterms by Substitution)**.** *Let $t, u$ be metaterms. For $\mathcal{R} \in \{\mathtt{x}, \mathtt{ex}, \lambda\mathtt{x}, \lambda\mathtt{ex}\}$, if $t \to_{\mathcal{R}} t'$, then $u\{x/t\} \to^*_{\mathcal{R}} u\{x/t'\}$ and $t\{x/u\} \to_{\mathcal{R}} t'\{x/u\}$. Thus in particular $t\{x/u\} \in \mathcal{SN}_{\mathcal{R}}$ implies $t \in \mathcal{SN}_{\mathcal{R}}$.*

*Proof.* By induction on $t \to t'$. □

**Lemma 8.4** (Full Composition for Metaterms)**.** *Let $t, u$ be metaterms. Then $t[x/u] \to^*_{\mathtt{ex}} t\{x/u\}$.*

*Proof.* The proof can be done by induction on $t$ using Lemma 8.1. In contrast to full composition on terms (Lemma 2.2), the property holds with an equality for the base case $t = \mathbb{X}_\Delta$ with $x \in \Delta$ since $\mathbb{X}_\Delta[x/u] = \mathbb{X}_\Delta\{x/u\}$. □

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

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

Indeed, while this diagram can be closed in $\lambda\mathtt{x}$ for terms *without metavariables* [BR95], 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/decorated metavariables. Fortunately, this diagram can be closed in the $\lambda\mathtt{ex}$-calculus as follows. If $y \in \mathtt{fv}(u)$, then $s \to_{\mathtt{Comp}} s'$, otherwise $s' \to^*_{\mathtt{ex}\ (L.\ 8.4)} t[y/v][x/u\{y/v\}] =_{(L.\ 8.1)} t[y/v][x/u] =_{\mathtt{C}} s'$.

We now develop a confluence proof for metaterms which is based on the existence of a mapping allowing to verify the Z-property as stated by van Oostrom [vO].

**Definition 8.5** (Z-Property)**.** A map $^\circ$ from terms to terms satisfies the *Z-property* for a reduction relation $\to_{\mathcal{R}}$ iff $t \to_{\mathcal{R}} u$ implies $u \to^*_{\mathcal{R}} t^\circ$ and $t^\circ \to^*_{\mathcal{R}} u^\circ$. A reduction relation $\to_{\mathcal{R}}$ has the *Z-property* if there is a map which satisfies the *Z-property* for $\to_{\mathcal{R}}$.

It turns out [vO] that $\to_{\mathcal{R}}$ is confluent if it has the Z-property (see Theorem A.1 in the Appendix A), so to show confluence of $\lambda\mathtt{ex}$ it is then sufficient to define a map on metaterms satisfaying the Z-property. Such a map can be defined in terms of the superdevelopment function for the $\lambda$-calculus [Acz78, vR93].

**Definition 8.6** (Superdevelopment Function)**.** The function $\_^\circ$ on metaterms is defined by induction as follows:

$$\begin{array}{lll@{\qquad}lll@{\qquad}l}
\mathbb{X}_\Delta^\circ & := & \mathbb{X}_\Delta & (tu)^\circ & := & t^\circ u^\circ & \text{if } t^\circ \text{ is not an abstraction}\\
x^\circ & := & x & (tu)^\circ & := & v\{x/u^\circ\} & \text{if } t^\circ = \lambda x.v\\
(\lambda x.t)^\circ & := & \lambda x.t^\circ & t[x/u]^\circ & := & t^\circ\{x/u^\circ\} &
\end{array}$$

Notice that $\mathtt{fv}(t^\circ) \subseteq \mathtt{fv}(t)$.

**Lemma 8.7.** *Let $t, u$ be metaterms. Then $t^\circ u^\circ \to^*_{\lambda\mathtt{ex}} (tu)^\circ$.*

*Proof.* If $t^\circ$ is not an abstraction, then $t^\circ u^\circ = (tu)^\circ$. If $t^\circ = \lambda y.s$, then $t^\circ u^\circ = (\lambda y.s)u^\circ \to_{\mathtt{B}} s[y/u^\circ] \to^*_{\mathtt{ex}\ (L.\,8.4)} s\{y/u^\circ\} = (tu)^\circ$. ☐

**Lemma 8.8.** *Let $t, u$ be metaterms. Then $t^\circ\{x/u^\circ\} \to^*_{\lambda\mathtt{ex}} t\{x/u\}^\circ$.*

*Proof.* The proof is by induction on $t$. Suppose $t = vw$.

- If $v^\circ$ is not an abstraction, then

$$\begin{array}{lll}
(vw)^\circ\{x/u^\circ\} = & & \\
v^\circ\{x/u^\circ\}w^\circ\{x/u^\circ\} & \to^*_{\lambda\mathtt{ex}\ (i.h.)} v\{x/u\}^\circ w\{x/u\}^\circ & \to^*_{\lambda\mathtt{ex}\ (L.\,8.7)} (vw)\{x/u\}^\circ
\end{array}$$

- If $v^\circ = \lambda z.r$, then the i.h. gives $v^\circ\{x/u^\circ\} = (\lambda z.r)\{x/u^\circ\} \to^*_{\lambda\mathtt{ex}} v\{x/u\}^\circ$ so that $v\{x/u\}^\circ = \lambda z.s$ where $r\{x/u^\circ\} \to^*_{\lambda\mathtt{ex}} s$. As a consequence,

$$\begin{array}{lll}
(vw)^\circ\{x/u^\circ\} = & & \\
r\{z/w^\circ\}\{x/u^\circ\} =_{\mathtt{e}\ (L.\,8.2)} & & \\
r\{x/u^\circ\}\{z/w^\circ\{x/u^\circ\}\} & \to^*_{\lambda\mathtt{ex}} & s\{z/w^\circ\{x/u^\circ\}\}\\
 & \to^*_{\lambda\mathtt{ex}\ (i.h.\ \&\ L.\,8.3)} & s\{z/w\{x/u\}^\circ\}\\
 & & = (v\{x/u\}w\{x/u\})^\circ\\
 & & = (vw)\{x/u\}^\circ
\end{array}$$

The case $t = v[y/w]$ also uses the i.h. and Lemma 8.2. All the other cases are straightforward. $\square$

**Lemma 8.9.** *Let $t$ be a metaterm. Then $t \to^*_{\lambda\mathtt{ex}} t^\circ$.*

*Proof.* By induction on $t$. The interesting cases are the following ones.

- $t = uv$: Then $uv \to^*_{\lambda\mathtt{ex}\ (i.h.)} u^\circ v^\circ \to^*_{\lambda\mathtt{ex}\ (L.8.7)} (uv)^\circ = t^\circ$.
- $t = u[x/v]$: Then $u[x/v] \to^*_{\lambda\mathtt{ex}\ (i.h.)} u^\circ[x/v^\circ] \to^*_{\mathtt{ex}\ (L.8.4)} u^\circ\{x/v^\circ\} \to^*_{\lambda\mathtt{ex}\ (L.8.8)} u\{x/v\}^\circ$.

All the other cases are straightforward. $\square$

**Lemma 8.10** (Towards the Z-Property)**.** *Let $t, u$ be metaterms. If $t \to_{\mathtt{Bx}} u$, then $u \to^*_{\lambda\mathtt{ex}} t^\circ \to^*_{\lambda\mathtt{ex}} u^\circ$.*

*Proof.* By induction on $t \to_{\mathtt{Bx}} u$.

- If $t = \lambda x.r \to_{\mathtt{Bx}} \lambda x.s = u$, where $r \to_{\mathtt{Bx}} s$, then the property holds by the i.h.
- If $t = r[x/v] \to_{\mathtt{Bx}} s[x/v] = u$, where $r \to_{\mathtt{Bx}} s$, then

$$\begin{array}{llllll} u = s[x/v] & \to^*_{\lambda\mathtt{ex}\ (i.h.)} & r^\circ[x/v] & & & \\ & \to^*_{\lambda\mathtt{ex}\ (L.8.9)} & r^\circ[x/v^\circ] & & & \\ & \to^*_{\mathtt{ex}\ (L.8.4)} & r^\circ\{x/v^\circ\} = t^\circ & \to^*_{\lambda\mathtt{ex}\ (i.h.\ \&\ L\,8.3)} & s^\circ\{x/v^\circ\} & = \\ & & & & s[x/v]^\circ & = u^\circ \end{array}$$

- If $t = v[x/r] \to_{\mathtt{Bx}} v[x/s] = u$, where $r \to_{\mathtt{Bx}} s$, then proceed as in the previous case.
- If $t = rv \to_{\mathtt{Bx}} sv = u$, where $r \to_{\mathtt{Bx}} s$, then $sv \to^*_{\lambda\mathtt{ex}\ (i.h.)} r^\circ v \to^*_{\lambda\mathtt{ex}\ (L.8.9)} r^\circ v^\circ \to^*_{\lambda\mathtt{ex}\ (L.8.7)} (rv)^\circ$. For the second part of the statement there are two cases:
  - If $r^\circ$ is not an abstraction, then $(rv)^\circ = r^\circ v^\circ \to^*_{\lambda\mathtt{ex}\ (i.h.)} s^\circ v^\circ \to^*_{\lambda\mathtt{ex}\ (L.8.7)} (sv)^\circ$.
  - If $r^\circ = \lambda z.w$, then the i.h. $r^\circ \to^*_{\lambda\mathtt{ex}} s^\circ$ implies $s^\circ = \lambda z.q$, where $w \to^*_{\lambda\mathtt{ex}} q$. We conclude with $(rv)^\circ = w\{z/v^\circ\} \to^*_{\lambda\mathtt{ex}\ (L.8.3)} q\{z/v^\circ\} = (sv)^\circ$.
- If $t = vr \to_{\mathtt{Bx}} vs = u$, where $r \to_{\mathtt{Bx}} s$, then $vs \to^*_{\lambda\mathtt{ex}\ (i.h.)} vr^\circ \to^*_{\lambda\mathtt{ex}\ (L.8.9)} v^\circ r^\circ \to^*_{\lambda\mathtt{ex}\ (L.8.7)} (vr)^\circ$. For the second part of the statement there are two cases:
  - If $v^\circ$ is not an abstraction, then $(vr)^\circ = v^\circ r^\circ \to^*_{\lambda\mathtt{ex}\ (i.h.)} v^\circ s^\circ = (vs)^\circ$.
  - If $v^\circ = \lambda y.w$, then $(vr)^\circ = w\{y/r^\circ\} \to^*_{\lambda\mathtt{ex}\ (i.h.\ \&\ L.8.3)} w\{y/s^\circ\} = (vs)^\circ$.
- If $t = x[x/v] \to_{\mathtt{Var}} v = u$, then $x[x/v]^\circ = x\{x/v^\circ\} = v^\circ$. We conclude since $v \to^*_{\lambda\mathtt{ex}} v^\circ$ holds by Lemma 8.9.
- If $t = r[x/v] \to_{\mathtt{Gc}} r = u$, then $r[x/v]^\circ = r^\circ\{x/v^\circ\} =_{(L.8.1)} r^\circ$. We conclude since $r \to^*_{\lambda\mathtt{ex}} r^\circ$ holds by Lemma 8.9.
- If $t = (rs)[x/v] \to_{\mathtt{App}} r[x/v]s[x/v] = u$, then

$$\begin{array}{llllll} u & \to^*_{\lambda\mathtt{ex}\ (L.8.9)} & r^\circ[x/v^\circ]s^\circ[x/v^\circ] & & & \\ & \to^*_{\mathtt{ex}\ (L.8.4)} & r^\circ\{x/v^\circ\}s^\circ\{x/v^\circ\} & = & & \\ & & (r^\circ s^\circ)\{x/v^\circ\} & \to^*_{\lambda\mathtt{ex}\ (L.8.3\,\&\,8.7)} & (rs)^\circ\{x/v^\circ\} & = \\ & & & & (rs)[x/v]^\circ & = \ t^\circ \end{array}$$

  For the second part there are two cases.
  - If $r^\circ$ is not an abstraction, then

$$t^\circ = r^\circ\{x/v^\circ\}s^\circ\{x/v^\circ\} = r[x/v]^\circ s[x/v]^\circ \to^*_{\lambda\mathtt{ex}\ (L.8.7)} (r[x/v]s[x/v])^\circ = u^\circ$$

– If $r^\circ = \lambda y.q$, then $r[x/v]^\circ = \lambda y.q\{x/v^\circ\}$, so that

$$\begin{array}{lllll} t^\circ & = (rs)[x/v]^\circ & & & \\ & = (rs)^\circ\{x/v^\circ\} & & & \\ & = q\{y/s^\circ\}\{x/v^\circ\} & =_{\mathtt{e}\ (L.\,8.2)} & q\{x/v^\circ\}\{y/s^\circ\{x/v^\circ\}\} & = \\ & & & q\{x/v^\circ\}\{y/s[x/v]^\circ\} & = \\ & & & (r[x/v]s[x/v])^\circ & = \ u^\circ \end{array}$$

• If $t = (\lambda y.r)[x/v] \rightarrow_{\mathtt{Lamb}} \lambda y.r[x/v] = u$, then $(\lambda y.r)[x/v]^\circ = \lambda y.r^\circ\{x/v^\circ\}$. We have

$$u = \lambda y.r[x/v] \rightarrow^*_{\lambda\mathtt{ex}\ (L.\,8.9)} \lambda y.r^\circ[x/v^\circ] \rightarrow^*_{\mathtt{ex}\ (L.\,8.4)} \lambda y.r^\circ\{x/v^\circ\} = t^\circ = u^\circ$$

• If $t = r[x/v][y/w] \rightarrow_{\mathtt{Comp}} r[y/w][x/v[y/w]] = u$, then

$$\begin{array}{ll} u = r[y/w][x/v[y/w]] & \rightarrow^*_{\lambda\mathtt{ex}\ (L.\,8.9)} \\ r^\circ[y/w^\circ][x/v^\circ[y/w^\circ]] & \rightarrow^*_{\lambda\mathtt{ex}\ (L.\,8.4\ \&\ 8.3)} \\ r^\circ\{y/w^\circ\}\{x/v^\circ\{y/w^\circ\}\} & =_{\mathtt{e}\ (L.\,8.2)} r^\circ\{x/v^\circ\}\{y/w^\circ\} = t^\circ \end{array}$$

Since $u^\circ = r^\circ\{y/w^\circ\}\{x/v^\circ\{y/w^\circ\}\}$, then we have $t^\circ \rightarrow^*_{\lambda\mathtt{ex}} u^\circ$ as well. □

**Lemma 8.11.** *Let $t, u$ be metaterms s.t. $t =_{\mathtt{e}} u$. Then,*

- *If $r =_{\mathtt{e}} s$, then $t\{x/r\} =_{\mathtt{e}} u\{x/s\}$.*
- *$t^\circ =_{\mathtt{e}} u^\circ$.*

*Proof.* Suppose $t =_{\mathtt{e}} u$ holds in $n$ steps. Both properties can be simultaneously proved by induction on the lexicographic pair $\langle n, t\rangle$. □

**Corollary 8.12** (Z-Property)**.** *Let $t, u$ be metaterms. If $t \rightarrow_{\lambda\mathtt{ex}} u$, then $u \rightarrow^*_{\lambda\mathtt{ex}} t^\circ \rightarrow^*_{\lambda\mathtt{ex}} u^\circ$.*

*Proof.* Let $t =_{\mathtt{e}} r \rightarrow_{\mathtt{Bx}} s =_{\mathtt{e}} u$. By Lemma 8.10 $r \rightarrow^*_{\lambda\mathtt{ex}} s^\circ \rightarrow^*_{\lambda\mathtt{ex}} r^\circ$ and by Lemma. 8.11 $t^\circ =_{\mathtt{e}} r^\circ$ and $s^\circ =_{\mathtt{e}} u^\circ$. We thus conclude $t \rightarrow^*_{\lambda\mathtt{ex}} u^\circ \rightarrow^*_{\lambda\mathtt{ex}} t^\circ$. □

**Corollary 8.13** (Confluence)**.** *The reduction relation $\rightarrow_{\lambda\mathtt{ex}}$ is confluent on metaterms.*

*Proof.* Corollary 8.12 guarantees the Z-property. We conclude by Theorem A.1 in the Appendix A. □

## 9. Conclusion

We propose simple syntax in named variable notation to model a calculus with explicit substitutions enjoying good properties, specially confluence on metaterms, preservation of $\beta$-strong normalisation, strong normalisation of typed terms and implementation of full composition.

A simple perpetual strategy is defined for calculi with ES enjoying full composition in a modular way. This strategy is used to provide an inductive definition of SN terms which is then used to prove that untyped terms enjoy PSN. The inductive characterisation of SN terms and the PSN theorem are really modular with respect to other proofs in the literature [LLD+04, Bon01b], especially because we make an intensive use of two abstract properties: full composition and the **IE** property. Last but not least, our development is direct, since it is not based on similar properties for other related calculi, and has a constructive style, since no classical axiom seems to be needed.

Some remarks about the application of this modular method to other calculi with ES might be interesting. On one hand, the technology presented in this paper has been successfully applied to other calculi with explicit substitutions enjoying full composition [KR09, AG09]. On the other hand, full composition alone is not sufficient to achieve the SN proof, otherwise the $\lambda\sigma$-calculus [ACCL91], which is known to *not* being strongly normalising [Mel95], could be treated. Indeed, our strategy $\rightsquigarrow$ is not perpetual for $\lambda\sigma$: Melliès' counter-example is based on an infinite $\lambda\sigma$-reduction sequence starting from a simply typed term which is not reached by our perpetual strategy. In other words, $\rightsquigarrow$ is incomplete for $\lambda\sigma$. The definition of a perpetual strategy for $\lambda\sigma$ remains open.

We believe that a de Bruijn or nominal version of $\lambda\mathtt{ex}$ could be useful in real implementations. In the first case, this could be achieved by using for example $\lambda\sigma_{\Uparrow}$ technology (so that equation $\mathtt{C}$ can be eliminated) together with some control of composition needed to guarantee strong normalisation.

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 [LDM06] or natural deduction [Muñ01]. The main contribution of $\lambda\mathtt{ex}$ with respect to the formalisms previously mentioned would be the *safe* notion of full composition.

## References


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

[ACCL91] Martín Abadi, Luca Cardelli, Pierre Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[Acz78] Peter Aczel. A general church-rosser theorem, 1978. Unpublished note, University of Manchester.

[AG09] Beniamino Accattoli and Stefano Guerrini. Jumping Boxes. Representing lambda-calculus boxes by jumps. In *18th EACSL Annual Conference on Computer Science Logic (CSL)*, *Lecture Notes in Computer Science*, September 2009.

[BBKV76] Henk Barendregt, Jan Bergstra, Jan-Willem Klop, and Henri Volken. Degress, reductions and representability in the lambda calculus. Technical Report 22, Utrecht University, 1976.

[BBLRD96] Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, and Jocelyne Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

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

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

[Bon01a] Eduardo Bonelli. Perpetuality in a named lambda calculus with explicit substitutions. *Mathematical Structures in Computer Science*, 11(1):47–90, 2001.

[Bon01b] Eduardo Bonelli. *Substitutions explicites et réécriture de termes*. Thèse de doctorat, Université Paris XI, Orsay, November 2001.

[BR95] Roel Bloo and Kristoffer 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.

[CDC78] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archive for Mathematical Logic*, 19:139–156, 1978.

[CDC80] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the $\lambda$-calculus. *Notre Dame Journal of Formal Logic*, 4:685–693, 1980.

[Cur91] Pierre-Louis Curien. An abstract frame work for environment machines. *Theoretical Computer Science*, 82(2):389–402, 1991.

[dB72] Nicolaas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indag. Mathematicae*, 5(35):381–392, 1972.

[dB78] Nicolaas G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Technical Report 78-WSK-03, Eindhoven University of Technology, 1978.

[dB87] Nicolaas G. de Bruijn. Generalizing Automath by Means of a Lambda-Typed Lambda Calculus. In *Mathematical Logic and Theoretical Computer Science*, number 106 in Lecture Notes in Pure and Applied Mathematics, 1987.

[DCKP00] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. In Jerzy 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] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409–450, 2003.

[DG99] René David and Bruno Guillaume. The $\lambda_l$-calculus. In Delia Kesner, editor, *Proceedings of the 2nd Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, pages 2–13, July 1999.

[DG01] René David and Bruno Guillaume. A $\lambda$-calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11:169–206, 2001.

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

[DU01] Roy Dyckhoff and Christian Urban. Strong normalisation of Herbelin's explicit substitution calculus with substitution propagation. In Pierre Lescanne, editor, *Proceedings of the 3rd Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, pages 26–45, June 2001.

[For02] Julien Forest. A weak calculus with explicit operators for pattern matching and substitution. In Sophie 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.

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

[GL99] Jean Goubault-Larrecq. Conjunctive types and SKInT. In Thorsten Altenkirch, Wolfgang Naraschewski, and Bernhard 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.

[GP99] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In Giuseppe Longo, editor, *14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 214–224. IEEE Computer Society Press, July 1999.

[Her94] Hugo Herbelin. A $\lambda$-calculus structure isomorphic to sequent calculus structure. In Leszek Pacholski and Jerzy 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] Thérèse Hardin and Jean-Jacques Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[HMP96] Thérèse Hardin, Luc Maranget, and Bruno Pagano. Functional back-ends within the lambda-sigma calculus. In R. Kent Dybvig, editor, *Proceedings of the ACM International Conference on Functional Programming*, pages 25–33. ACM Press, May 1996.

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

[KC] Delia Kesner and Shane Ó Conchúir. Milner's lambda calculus with partial substitutions. Available on `http://www.pps.jussieu.fr/~kesner/papers/`.

[Kes07] Delia Kesner. The theory of calculi with explicit substitutions revisited. In Jacques Duparc and Thomas Henzinger, editors, *Proceedings of the 16th Annual Conference of the European

*Association for Computer Science Logic (CSL)*, volume 4646 of *Lecture Notes in Computer Science*, pages 238–252. Springer-Verlag, September 2007.

[Kes08] Delia Kesner. Perpetuality for full and safe composition (in a constructive setting). In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5126 of *Lecture Notes in Computer Science*, pages 311–322. Springer-Verlag, July 2008.

[Kik07] Kentaro Kikuchi. Simple proofs of characterizing strong normalization for explicit substitution calculi. In Franz Baader, editor, *18th International Conference on Rewriting Techniques and Applications (RTA)*, volume 4533 of *Lecture Notes in Computer Science*, pages 257–272. Springer-Verlag, September 2007.

[KL05] Delia Kesner and Stéphane Lengrand. Extending the explicit substitution paradigm. In Jürgen 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.

[KL07] Delia Kesner and Stéphane Lengrand. Resource operators for lambda-calculus. *Information and Computation*, 205(4):419–473, 2007.

[KL08] Kentaro Kikuchi and Stéphane Lengrand. Strong normalisation of cut-elimination that simulates $\beta$-reduction. In Roberto Amadio, editor, *Foundations of Software Science and Computation Structures*, volume 4962 of *Lecture Notes in Computer Science*, pages 380–394. Springer-Verlag, March 2008.

[KR09] Delia Kesner and Fabien Renaud. The prismoid of resources. In *34st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, *Lecture Notes in Computer Science*, Springer-Verlag, August 2009.

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

[KR97] Fairouz Kamareddine and Alejandro Ríos. Extending a $\lambda$-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, 1997.

[KR98] Fairouz Kamareddine and Alejandro Ríos. Bridging de Bruijn indices and variable names in explicit substitutions calculi. *Logic Journal of the Interest Group of Pure and Applied Logic*, 6(6):843–874, 1998.

[LDM06] Stéphane Lengrand, Roy Dyckhoff, and James McKinna. A sequent calculus for type theory. In Zoltan 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.

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

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

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

[LLD+04] Stéphane Lengrand, Pierre Lescanne, Dan Dougherty, Mariangiola Dezani-Ciancaglini, and Steffen van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

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

[LRD94] Pierre Lescanne and Jocelyne Rouyer-Degli. The calculus of explicit substitutions $\lambda\upsilon$. Technical report, INRIA, Lorraine, 1994.

[LRD95] Pierre Lescanne and Jocelyne Rouyer-Degli. Explicit substitutions with de Bruijn levels. In Jieh 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.

[Mel95] Paul-André Melliès. Typed $\lambda$-calculi with explicit substitutions may not terminate. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Proceedings of the 2nd International Conference on Typed Lambda Calculus and Applications (TLCA)*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334. Springer-Verlag, April 1995.

[Mil06] Robin Milner. Local bigraphs and confluence: two conjectures. In Roberto Amadio and Iain Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency (EXPRESS)*, volume 175. Electronic Notes in Theoretical Computer Science, 2006.

[Muñ01] César Muñoz. Dependent types and explicit substitutions: a meta-theoretical development. *Mathematical Structures in Computer Science*, 11(1), 2001.

[Pot80] Garrell Pottinger. A type assignment for the strongly normalizable $\lambda$-terms. In Roger Hindley and Jonathan P. Seldin, editors, *To Haskell Brooks Curry: Essays in Combinatory Logic, Lambda Calculus and formalism*, pages 561–577. Academic Press, 1980.

[Ros92] Kristoffer Rose. Explicit cyclic substitutions. In Michaël Rusinowitch and Jean-Luc Rémy, 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] Takafumi Sakurai. Strong normalizability of calculus of explicit substitutions with composition. Available on `http://www.math.s.chiba-u.ac.jp/~sakurai/papers.html`.

[SFM03] Francois-Régis Sinot, Maribel Fernández, and Ian Mackie. Efficient reductions with director strings. In Robert 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.

[SvO07] François-Régis Sinot and Vincent van Oostrom. Preserving termination of the $\lambda$-calculus or not, 2007. Unpublished note.

[Tai67] William Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32, 1967.

[vD77] Diederik Ton van Daalen. *The language theory of automath*. PhD thesis, Technische Hogeschool Eindhoven, 1977.

[vO] Vincent van Oostrom. Z. See `http://www.phil.uu.nl/~oostrom/publication/rewriting.html` for slides.

[vR93] Femke van Raamsdonk. Confluence and superdevelopments. In Claude Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications (RTA)*, volume 690 of *Lecture Notes in Computer Science*, pages 168–182. Springer-Verlag, June 1993.

[vR96] Femke van Raamsdonk. *Confluence and Normalization for Higher-Order Rewriting*. PhD thesis, Amsterdam University, Netherlands, 1996.

[vRSSX99] Femke van Raamsdonk, Paula Severi, Morten Heine Sorensen, and Hongwei Xi. Perpetual reductions in $\lambda$-calculus. *Information and Computation*, 149(2), 1999.

## Appendix A. Abstract Reduction Results

**Theorem A.1** (Z implies Confluence)**.** *If $\to_{\mathcal{R}}$ has the Z-property, then $\to_{\mathcal{R}}$ is confluent.*

*Proof.* We give a proof following the picture appearing in [vO] which proceeds in many steps. Suppose that $\_^\circ$ is some map satisfying the Z-property for $\mathcal{R}$.

(1) Define $a^\bullet := a$ if $a$ is in $\mathcal{R}$-normal form, $a^\bullet := a^\circ$ otherwise.

(2) Prove that $\_^\bullet$ also satisfies the Z-property for $\to_{\mathcal{R}}$.

*Proof.* If $a \to_{\mathcal{R}} b$, then $b \to^*_{\mathcal{R}} a^\circ \to^*_{\mathcal{R}} b^\circ$ by the hypothesis and $a^\bullet = a^\circ$ by Point (1) so that $b \to^*_{\mathcal{R}} a^\bullet$. If $b$ is an $\mathcal{R}$-normal form, then $b^\bullet = b = a^\circ = a^\bullet$ so that $a^\bullet \to^*_{\mathcal{R}} b^\bullet$. If $b$ is not an $\mathcal{R}$-normal form, then $b^\bullet = b^\circ$ so that also $a^\bullet = a^\circ \to^*_{\mathcal{R}} b^\circ = b^\bullet$.

(3) Prove that $a \to^*_{\mathcal{R}} a^\bullet$.

*Proof.* If $a$ is an $\mathcal{R}$-normal form, then $a^\bullet = a$ so we are done. Otherwise, there is $b$ such that $a \to_{\mathcal{R}} b$, so that Point (2) gives $b \to^*_{\mathcal{R}} a^\bullet$ and thus $a \to^*_{\mathcal{R}} a^\bullet$.

(4) Prove that $a \to^*_{\mathcal{R}} b$ implies $a^\bullet \to^*_{\mathcal{R}} b^\bullet$.

*Proof.* By induction on the number $n$ of steps from $a$ to $b$. If $n = 0$, then $a = b$ and $a^\bullet = b^\bullet$. If $n > 0$, then $a \to_{\mathcal{R}} c \to^*_{\mathcal{R}} b$, where $c \to^*_{\mathcal{R}} b$ holds in $n-1$ steps. Point (2) and the i.h. give $a^\bullet \to^*_{\mathcal{R}} c^\bullet \to^*_{\mathcal{R}} b^\bullet$.

(5) Conclude confluence of $\to_{\mathcal{R}}$.

*Proof.* Let $t \to^*_{\mathcal{R}} t_1$ and $t \to^*_{\mathcal{R}} t_2$. We want to show that there is $t_3$ such that $t_1 \to^*_{\mathcal{R}} t_3$ and $t_2 \to^*_{\mathcal{R}} t_3$. We proceed by induction on the number $n$ of steps from $t$ to $t_2$. If $n = 0$, then $t = t_2$ and we take $t_3 = t_1$ so we are done. If $n > 0$, then $t \to_{\mathcal{R}} u \to^*_{\mathcal{R}} t_2$, with $n-1$ steps from $u$ to $t_2$. By Point (2) $u \to^*_{\mathcal{R}} t^\bullet$ and by Point (4) $t^\bullet \to^*_{\mathcal{R}} t_1^\bullet$ so that $u \to^*_{\mathcal{R}} t_1^\bullet$. By Point (3) $t_1 \to^*_{\mathcal{R}} t_1^\bullet$. Now, $u \to^*_{\mathcal{R}} t_1^\bullet$ and $u \to^*_{\mathcal{R}} t_2$ holds in $n-1$ steps so we close the diagram by the i.h.. ∎

**Theorem A.2** (Modular Strong Normalisation)**.** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two reduction relations on* $\mathtt{s}$ *and let $\mathcal{A}$ be a reduction relation on* $\mathtt{S}$. *Let $\mathcal{R} \subseteq \mathtt{s} \times \mathtt{S}$. Suppose*

**P1:** *For every $u, v, U$ ($u \mathrel{\mathcal{R}} U$ & $u \mathrel{\mathcal{A}_1} v$ imply $\exists V$ s.t. $v \mathrel{R} V$ and $U \mathrel{\mathcal{A}^*} V$).*

**P1:** *For every $u, v, U$ ($u \mathrel{\mathcal{R}} U$ & $u \mathrel{\mathcal{A}_2} v$ imply $\exists V$ s.t. $v \mathrel{\mathcal{R}} V$ and $U \mathrel{\mathcal{A}^+} V$).*

**P1:** *The relation $\mathcal{A}_1$ is well-founded.*

*Then, $t \mathrel{\mathcal{R}} T$ & $T \in \mathcal{SN}_{\mathcal{A}}$ imply $t \in \mathcal{SN}_{\mathcal{A}_1 \cup \mathcal{A}_2}$.*

*Proof.* A constructive proof of this theorem can be found as Corollary 26 of [Len06]. A proof by contradiction can be easily done as follows. Suppose $t \notin \mathcal{SN}_{\mathcal{A}_1 \cup \mathcal{A}_2}$. Then, there is an infinite $\mathcal{A}_1 \cup \mathcal{A}_2$-reduction sequence starting at $t$, and since $\mathcal{A}_1$ is a well-founded relation by **P3**, this reduction sequence has necessarily the form

$$t \to^*_{\mathcal{A}_1} t_1 \to^+_{\mathcal{A}_2} t_2 \to^*_{\mathcal{A}_1} t_3 \to^+_{\mathcal{A}_2} \dots \infty$$

and can be projected by **P1** and **P2** into an infinite $\mathcal{A}$-reduction sequence as follows:

$$\begin{array}{ccccccccc}
t & \to^*_{\mathcal{A}_1} & t_1 & \to^+_{\mathcal{A}_2} & t_2 & \to^*_{\mathcal{A}_1} & t_3 & \to^+_{\mathcal{A}_2} & \dots\infty \\
\mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \\
T & \to^*_{\mathcal{A}} & T_1 & \to^+_{\mathcal{A}} & T_2 & \to^*_{\mathcal{A}} & T_3 & \to^+_{\mathcal{A}} & \dots\infty
\end{array}$$

We thus get a contradiction with the fact the $T \in \mathcal{SN}_{\mathcal{A}}$. ∎