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\footnote{Definition of substitution modulo $\alpha$-conversion avoids to explicitly deal with the variable capture case. Thus, for example $(\lambda x.y)\{y/x\} =_{\alpha} (\lambda z.y)\{y/x\} =_{\alpha f} \lambda z.y\{y/x\} = \lambda z.x.$} as follows:

- $x\{x/v\} := v$
- $y\{x/v\} := y$ if $x \neq y$
- $(u_1u_2)\{x/v\} := u_1\{x/v\}u_2\{x/v\}$
- $(\lambda y.u)\{x/v\} := \lambda y.u\{x/v\}$

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 of substitution rules.
of reduction rules (so that one still works modulo α-conversion). The following reduction system, known as λx [Lin86 Lin92 Ros92 BR95], is thus obtained.

\[
\begin{align*}
\lambda(x.t) & \rightarrow t[x/v] \\
x[x/v] & \rightarrow v \\
y[x/v] & \rightarrow y \\
(u_1 u_2)[x/v] & \rightarrow u_1[u_1][u_2][x/v] \\
(\lambda y.u)[x/v] & \rightarrow \lambda y.u[x/v]
\end{align*}
\]

The λx-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 \((zyx)[y/(xx)][x/v]\), which can be further reduced to \((zyv)[y/yy]\), a term equal to \((zyx)[y/xx]t\), where \([x/v]\) is the meta/implicit substitution that the explicit substitution \([x/v]\) is supposed to implement.

In these last twenty years there has been a growing interest in λ-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 λx-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 λσ [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 \(t[u/u]\); in other words, explicit substitution implements the implicit one. Indeed, taking again the previous example, \((zyx)[y/xx][x/v]\) reduces to \((zyx)[y/xx][x/v]\) = \((zyv)[y/yy]\). Many calculi such as λσ, λσ\(\_\) [HL89], λsub [Mil06], λlxr [KL05 KL07] and λes [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 λ-calculus is perfectly orthogonal (in particular does not have critical pairs), calculi with ES such as λx suffer at least from the following well-known diverging example:

\[
t[y/v][x/u][y/v] \rightarrow (\lambda x.t)[y/v][y/v] \rightarrow t[x/u][y/v]
\]

Different solutions were adopted in the literature to close this diagram. If no new rewriting rule is added to those of the minimal λx-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 \(y[x/u] \rightarrow y\) by the more general one \(t[x/u] \rightarrow t\) if \(x \not\in f(v(t))\).
incomplete programs and proofs). If liberal rules for composition are considered, as in \(\lambda \sigma, \lambda \sigma_\#_i\), or \(\lambda \sigma_e\) [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’ counterexample [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 calculus [LM99, Fort92], 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 \(\lambda_{lxr}\), 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 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 \(\lambda_{lxr}\)-calculus whose syntax is obtained by incorporating these new operators to the \(\lambda\)-terms. The reduction system of \(\lambda_{lxr}\) 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 \(\lambda_{lxr}\) is just conjectured but not still proved. Also, whereas \(\lambda_{lxr}\) 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 \(\lambda_{ex}\)-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 \(\lambda_{ex}\)-calculus.

The \(\lambda_{ex}\)-calculus is obtained by extending \(\lambda x\) 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 \(\lambda_{lxr}\) to guarantee PSN. The substitutions of \(\lambda_{ex}\) 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 \textit{sets} (given by commutation of independent unary substitutions) in \( \lambda \text{ex} \).

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 \( \lambda \text{ws} \) in [DCKP03] and \( \lambda \text{lxr} \) [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 \( \lambda \text{es} \), where PSN is obtained by two consecutive translations, one from \( \lambda \text{es} \) 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 \textit{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 \( \lambda \text{ex} \), 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 \( \lambda \text{ex} \) are the following. We first define a \textit{perpetual} reduction strategy for \( \lambda \text{ex} \): if \( t \) can be reduced to \( t' \) by the strategy, and \( t' \in SN_{\lambda \text{ex}} \), then \( t \in SN_{\lambda \text{ex}} \). In particular, since the perpetual strategy reduces \( t[x/u] \) to \( t\{x/u\} \), one has to show that normalisation of \textit{Implicit} substitution implies normalisation of \textit{Explicit} substitution. More precisely,

\[
\text{(IE)} \quad u \in SN_{\lambda \text{ex}} \& t\{x/u\} \in SN_{\lambda \text{ex}} \implies t[x/u] \in SN_{\lambda \text{ex}}.
\]

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 \textit{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 \( SN_{\lambda \text{ex}} \) by means of just four inference rules. This inductive characterisation is then used to show that \textit{untyped} terms preserve \( \beta \)-strong normalisation and that \textit{typed} terms are in \( SN_{\lambda \text{ex}} \). 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 \( \lambda \text{ex} \).

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 \textit{IE} property is mostly inspired from the PSN proofs used for the \textit{non equational} systems \( \lambda \text{x} \) and \( \lambda \text{ws} \) 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 \textit{minimal} non-terminating reduction sequences. The calculus proposed in [Sak] specifies commutation of independent substitutions by a \textit{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 $SN_{\lambda ex}$ as well as the use of the Z-property of van Oostrom [Ov] to show confluence. It is organised as follows. Section 2 introduces syntax and reduction rules for the $\lambda ex$-calculus. The perpetual strategy for $\lambda ex$ 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 $\lambda ex$-calculus can be viewed as a simple extension of the $\lambda x$-calculus. The set of terms (meta-variables $s, t, u, v$) is defined by the following grammar:

$$T ::= x \mid TT \mid \lambda x.T \mid T[x/T]$$

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

- $fv(x) := \{x\}$
- $fv(\lambda x.u) := fv(u) \setminus \{x\}$
- $fv(uv) := fv(u) \cup fv(v)$
- $fv(u[x/v]) := (fv(u) \setminus \{x\}) \cup fv(v)$

Thus, $x$ 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 $y \notin fv(u)$, and dependent iff $y \in fv(u)$. Notice that in both cases we can always assume $x \notin fv(v)$ by $\alpha$-conversion. We use the notation $t^n_u$ for a list of $n \geq 0$ terms $t_1, \ldots, t_n$ and $ut^n_{1n}$ 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.

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

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

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 $y \notin fv(v)$ in the rewriting rule $\text{Lamb}$. Same kind of assumptions are done for the rewriting rule $\text{Comp}$ and the equation $C$.

The rewriting relation $\rightarrow_{Bx}$ is generated by all the rewriting rules in Figure 1 and $\rightarrow_{x}$ is only generated by the five last ones. The equivalence relation $=_{e}$ is generated by the conversions $\alpha$ and $C$. The reduction relations $\rightarrow_{ex}$ and $\rightarrow_{\lambda ex}$ are respectively generated by...
Lemma 2.1 (Basic Properties). Let $R \in \{ex, lex\}$ and let $t, t', u$ be terms.

- If $t \xrightarrow{R} t'$, then $fv(t') \subseteq fv(t)$.
- If $t \xrightarrow{R} t'$, then $u\{x/t\} \xrightarrow{R} u\{x/t'\}$ and $t\{x/u\} \xrightarrow{R} t'\{x/u\}$. Thus in particular $t\{x/u\} \in SN_R$ implies $t \in SN_R$.

As explained in Section I the composition rule $\text{Comp}$ and the equation $C$ guarantee the following property:

Lemma 2.2 (Full Composition for Terms). Let $t, u$ be terms. Then $t\{x/u\} \xrightarrow{ex} t\{x/u\}$.

Proof. By induction on $t$. Consider $t = s[y/v]$. If $x \in fv(v)$, then $s[y/v]\{x/u\} \xrightarrow{Comp} s[x/u][y/v]\{x/u\} \xrightarrow{ex} s\{x/u\}[y/v][x/u] = t\{x/u\}$. If $x \notin fv(v)$, then $s[y/v]\{x/u\} \xrightarrow{C} s[x/u][y/v] \xrightarrow{ex} s\{x/u\}[y/v] = t\{x/u\}$. All the other cases are straightforward. □
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 $t \rightarrow_\beta t'$, then $t \rightarrow_{\lambda_{\text{ex}}} t'$.

### 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 $t \notin N_{\lambda_{\text{ex}}}$. This is done according to the following cases. If $t = x_1 \ldots x_n$, rewrite the left-most $x_i$ which is reducible. If $t = \lambda x . u$, rewrite $u$. If $t = (\lambda x . s) u$, rewrite the head redex. If $t = s[x/u]v_n$ and $u \notin S_{\lambda_{\text{ex}}}$, 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.

\[
\begin{align*}
&\frac{u_n \in N_{\lambda_{\text{ex}}}}{xu_mту_m' \rightsquigarrow xu_mту_m'} \quad (p\text{-var}) \\
&\frac{u \in S_{\lambda_{\text{ex}}}}{t[x/u]v_n \rightsquigarrow t[x/u]v_n} \quad (p\text{-subs1}) \\
&\frac{(\lambda x . t)u_{\overline{n}} \rightsquigarrow t[x/u]u_{\overline{n}}}{(\lambda x . t)u_{\overline{n}} \rightsquigarrow t[x/u]u_{\overline{n}}} \quad (p\text{-abs}) \\
&\frac{t \rightsquigarrow t'}{\lambda x . t \rightsquigarrow \lambda x . t'} \quad (p\text{-B}) \\
&\frac{t \rightsquigarrow t'}{t[x/u]v_n \rightsquigarrow t[x/u]v_n} \quad (p\text{-subs2})
\end{align*}
\]

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-sub1) 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 $t \rightarrow_{\lambda_{\text{ex}}} t'$.

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

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 $t' \in S_{\lambda_{\text{ex}}}$, then $t \in S_{\lambda_{\text{ex}}}$.

**Proof.** By induction on the definition of the strategy $\rightsquigarrow$. \qed
• \(t = (\lambda x.s)u \eta \rightarrow s[x/u] \eta = t'\) by \((p-B)\). If \(s[x/u] \eta \in SN_{\lambda e x}\), then \(s, u, \eta \in SN_{\lambda e x}\).

We show \((\lambda x.s)u \eta \in SN_{\lambda e x}\) by induction on \(\eta \in SN_{\lambda e x}\) for the \(\eta \lambda e x\) and \(\Sigma_{i \in 1..n} \eta \lambda e x(u_i)\). For that, it is sufficient to show that every \(\lambda e x\)-reduct of \((\lambda x.s)u \eta\) is in \(SN_{\lambda e x}\). If the reduction takes place in a subterm of \((\lambda x.s)u \eta\), then the property holds by the \(\eta \lambda e x\). Otherwise \((\lambda x.s)u \eta \rightarrow s[x/u] \eta\) which is in \(SN_{\lambda e x}\) by hypothesis. We thus conclude \((\lambda x.s)u \eta \in SN_{\lambda e x}\).

• \(t = s[x/u] \eta \rightarrow s[x/u'] \eta = t'\) by \((p \text{-subs} 2)\), so that \(u \notin SN_{\lambda e x}\) and \(u \rightarrow u'\). If \(s[x/u] \eta \in SN_{\lambda e x}\), then in particular \(u' \in SN_{\lambda e x}\), thus \(u \in SN_{\lambda e x}\) by the \(\eta \lambda e x\). From \(u \notin SN_{\lambda e x}\) and \(u \in SN_{\lambda e x}\) we can get any proposition, so in particular \(t \in SN_{\lambda e x}\).

• \(t = s[x/u] \eta \rightarrow s[x/u] \eta = t'\) by \((p \text{-subs} 1)\) so that \(u \in SN_{\lambda e x}\). Then the \(IE\) property (Lemma 5.9 in Section 4) allows to conclude. All the other cases are straightforward. \(\square\)

An inductive syntactic characterisation of the set \(SN_{\lambda e x}\) 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] 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 syntactic form.

**Definition 3.4** (Inductive Characterisation of \(SN_{\lambda e x}\)). The inductive set \(ISN\) is defined as follows:

\[
\begin{align*}
t_1, \ldots, t_n &\in ISN \quad n \geq 0 & u[x/v]t_1 \ldots t_n &\in ISN \quad n \geq 0 \\
x.t_1 \ldots t_n &\in ISN & (\text{var}) \\
u[x/v]t_1 \ldots t_n &\in ISN \quad v \in ISN \quad n \geq 0 & (\lambda x.u)v.t_1 \ldots t_n &\in ISN \quad (\text{app}) \\
u[x/v]t_1 \ldots t_n &\in ISN & (\text{subs}) \\
u &\in ISN \quad n \geq 0 & \lambda x.u &\in ISN \quad (\text{abs})
\end{align*}
\]

**Proposition 3.5.** \(SN_{\lambda e x} = ISN\).

**Proof.** If \(t \in SN_{\lambda e x}\), then \(t \in ISN\) is proved by induction on the lexicographic pair \(\langle \eta \lambda e x (t), t \rangle\). If \(t \in ISN\), then \(t \in SN_{\lambda e x}\) is proved by induction on \(t \in ISN\) using Theorem 3.3. \(\square\)

The PSN property received a lot of attention in calculi with explicit substitutions, starting from an unexpected result given by Melliès [Mc95] 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 \(t \in SN_{\lambda}\), then \(t \in SN_{\lambda e x}\).

**Proof.** By induction on the definition of \(SN_{\lambda}\) [VR96] using the inductive Definition 3.4 and Proposition 3.5 (which holds by the Perpetuality Theorem 3.3).

If \(t = xt_1 \ldots t_n\) with \(t_i \in SN_{\lambda}\), then \(t_i \in SN_{\lambda e x}\) by the \(\eta \lambda e x\). 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 \(u[x/v]t_1 \ldots t_n \in SN_{\lambda}\)
and \( v \in SN_\beta \), then both terms are in \( SN_{\lambda \text{ex}} \) by the i.h. so that the \((\text{subs})\) rule gives \( u[x/v]t_1 \ldots t_n \in SN_{\lambda \text{ex}} \) and the \((\text{app})\) rule gives \((\lambda x.u)vt_1 \ldots t_n \in SN_{\lambda \text{ex}}\).

Alternative Proof. By induction on the definition of \( SN_\beta \) [vR96] using the IE property (Lemma 5.9 in Section 4).

If \( t = xt_1 \ldots t_n \) with \( t_i \in SN_\beta \), then \( t_i \in SN_{\lambda \text{ex}} \) by the i.h. so that \( t \in SN_{\lambda \text{ex}} \) is straightforward. If \( t = \lambda x.u \) with \( u \in SN_\beta \), then \( u \in SN_{\lambda \text{ex}} \) by the i.h. and thus \( t \in SN_{\lambda \text{ex}} \) is also straightforward. If \( t = (\lambda x.u)vt_1 \ldots t_n \), with \( u[x/v]t_1 \ldots t_n \in SN_\beta \) and \( v \in SN_\beta \), then both terms are in \( SN_{\lambda \text{ex}} \) by the i.h. The IE property gives \( t' = u[x/v]t_1 \ldots t_n \in SN_{\lambda \text{ex}} \) so that in particular \( u, v, t_1, \ldots, t_n \in SN_{\lambda \text{ex}} \). We show \( t = (\lambda x.u)vt_1 \ldots t_n \in SN_{\lambda \text{ex}} \) by induction on \( \mu_{\lambda \text{ex}}(u) + \mu_{\lambda \text{ex}}(v) + \Sigma_1 \mu_{\lambda \text{ex}}(t_i) \). For that, it is sufficient to show that every \( \lambda \text{ex}\)-reduct of \( t \) is in \( SN_{\lambda \text{ex}} \). Now, if the \( \lambda \text{ex}\)-reduct of \( t \) comes from an internal reduction, then conclude with the i.h. Otherwise, \( t \rightarrow_{\lambda \text{ex}} t' \) which is already in \( SN_{\lambda \text{ex}} \).

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 implicit substitution implies normalisation of explicit substitution:

\[
\text{(IE) } u \in SN_{\lambda \text{ex}} \land t[x/u] \overline{v_n} \in SN_{\lambda \text{ex}} \text{ imply } t[x/u] \overline{v_n} \in SN_{\lambda \text{ex}}
\]

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 \( \lambda \text{ex}\)-strongly normalising terms used as substitutions.
   Thus for example \( t[x/u] \) indicates that \( u \in T \) & \( u \in SN_{\lambda \text{ex}} \).
2. Enrich the original \( \lambda \text{ex}\)-reduction system with a relation \( \lambda \text{ex} \) used only to propagate terminating labelled substitutions. Let \( \lambda \text{ex} \) be the enriched calculus.
3. Show that \( u \in SN_{\lambda \text{ex}} \land t[x/u] \overline{v_n} \in SN_{\lambda \text{ex}} \) imply \( t[x/u] \overline{v_n} \in SN_{\lambda \text{ex}} \).
4. Show that \( t[x/u] \overline{v_n} \in SN_{\lambda \text{ex}} \) implies \( t[x/u] \overline{v_n} \in SN_{\lambda \text{ex}} \).

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

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

\[
L_S := x \mid L_S L_S \mid \lambda x.L_S \mid L_S[x/L_S] \mid L_S[x/v] \quad (v \in T \cap SN_{\lambda \text{ex}} \land \text{fv}(v) \subseteq S)
\]

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 \( t \in L_S \) are s.t. \( x \notin S \). Indeed, \( \alpha\)-conversion allows to choose names outside \( S \) 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 \( S \)). 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
is defined as the union of the rewriting relations the old reduction relation is generated by the rewriting rules in Figure 2 and the labelled substitutions may commute/traverse the labelled ones.

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

\[
t \xrightarrow{\lambda} t' \iff \exists s, s' \text{ s.t. } t =_{\text{euc}} s \xrightarrow{Bx} \vec{a} s' =_{\text{euc}} t'
\]

In order to show that \( u \in SN_{\lambda \cdot \vec{a}} \) and \( t[x/u][x/u] \in SN_{\lambda \cdot \vec{a}} \) imply \( t \xrightarrow{\vec{a}} t' \) we first need to relate the \( \lambda \cdot \vec{a} \)-reduction relation to that of the \( \lambda \cdot \vec{a} \)-calculus. For that, the reduction relation \( \lambda \cdot \vec{a} \), which is defined on labelled terms, is split in two relations \( \lambda \cdot \vec{a} \)- and \( \lambda \cdot \vec{a} \)- on labelled terms as well, which will both be projected into \( \lambda \cdot \vec{a} \)-reduction sequences. More precisely, \( \lambda \cdot \vec{a} \)- can be weakly projected (eventually empty steps) into \( \lambda \cdot \vec{a} \) while \( \lambda \cdot \vec{a} \)- can be strongly projected (at least one step) into \( \lambda \cdot \vec{a} \)- (details in the forthcoming Lemma 5.2).

**Definition 4.2 (Internal and External Reductions).** The internal reduction relation \( \xrightarrow{\lambda} \) on labelled terms is given by adding to \( \vec{a} \) the \( \lambda \cdot \vec{a} \)-reduction relation in the bodies of labelled substitutions. Formally, \( \xrightarrow{\lambda} \) is taken as the following reduction relation \( \xrightarrow{\lambda} \) on \( \alpha \cup C \cup \vec{C} \)-equivalence classes:

- If \( u \xrightarrow{Bx} u' \) and \( u, u' \) are terms, then \( t[x/u] \xrightarrow{\lambda} t[x/u'] \).
- If \( t \xrightarrow{\vec{a}} t' \), then \( t \xrightarrow{\lambda} t' \).
- If \( t \xrightarrow{\lambda} t', \) then \( tu \xrightarrow{\lambda} tu', ut \xrightarrow{\lambda} ut', \lambda x.t \xrightarrow{\lambda} \lambda x.t', t[x/u] \xrightarrow{\lambda} \lambda t'[x/u], u[x/t] \xrightarrow{\lambda} u[x/t'], t[x/u] \xrightarrow{\lambda} \lambda t'[x/u].

The external reduction relation \( \xrightarrow{\lambda} \) on labelled terms is given by \( \lambda \cdot \vec{a} \)-reduction on labelled terms everywhere except inside bodies of labelled substitutions. Formally, \( \xrightarrow{\lambda} \) is taken as the following reduction relation \( \xrightarrow{\lambda} \) on \( \alpha \cup C \cup \vec{C} \)-equivalence classes:

- If \( t \xrightarrow{Bx} t' \) occurs outside a labelled substitution, then \( t \xrightarrow{\lambda} t' \).
(Corollary 5.4). On labelled terms we define a special measure for
\( \lambda \) on labelled terms as follows. \(#\) counts the total number of
\( \lambda \) on terms to
\( \lambda \) in a labelled term
\( \lambda \) on terms:

\[ \phi = \lambda \eta \lambda x.t \rightarrow \lambda x.t', t[x/u] \rightarrow \lambda x.t' [x/u] \],
\( u[x/t] \rightarrow \lambda x.u[x/t] \) and \( t[x/u] \rightarrow \lambda x.t'[x/u] \).

**Lemma 4.3.** \( \lambda \text{ex} \subseteq \lambda \text{ex} \cup \lambda \text{ex} \).

**Proof.** Since we are working everywhere with \( \alpha \cup \mathcal{C} \)-equivalence classes, then it is sufficient
to show \( \lambda \cup \mathcal{C} \subseteq \lambda \text{ex} \cup \lambda \text{ex} \).
\( \subseteq \): If \( \lambda \text{ex} \) occurs inside a labelled substitution, then \( \lambda \text{ex} \), otherwise \( \lambda \text{ex} \).
\( \supseteq \): By induction on the definitions of \( \lambda \text{ex} \) and \( \lambda \text{ex} \).

Since \( \lambda \text{ex} \)-reduction will only be weakly projected into \( \lambda \text{ex} \), we need to guarantee that
there are no infinite \( \lambda \text{ex} \)-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 \( \lambda \text{ex} \) to that of \( \lambda \text{ex} \) (Corollary 5.4).

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

\[
\begin{align*}
\text{af}_x(z) & := 0 \\
\text{af}_x(\lambda y.t) & := \text{af}_x(t) + \text{af}_x(u) \\
\text{af}_x(t[x/u]) & := \text{af}_x(t) + \text{af}_x(u) \\
\text{af}_x(t[y/u]) & := \text{af}_x(t) + \text{af}_x(u)
\end{align*}
\]

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

\[
\begin{align*}
\text{dep}(x) & := 0 \\
\text{dep}(\lambda y.t) & := \text{dep}(t) \\
\text{dep}(t[x/u]) & := \text{dep}(t) + \text{af}_x(u) \\
\text{dep}(t[y/u]) & := \text{dep}(t) + \text{af}_x(u)
\end{align*}
\]

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

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

**Definition 4.5** (A Decreasing Measure for \( \mathcal{X} \setminus \text{Comp} \)). We consider the following function
\( \mathcal{K}(\lambda) \) on terms:

\[
\begin{align*}
\mathcal{K}(x) & := 1 \\
\mathcal{K}(\lambda x.t) & := \mathcal{K}(t) + \mathcal{K}(u) + 1
\end{align*}
\]

In order to extend \( \mathcal{K}(\lambda) \) on terms to \( \mathcal{K}(\lambda) \) on labelled terms we define a special measure for
\( \lambda \text{ex} \)-strongly normalising terms. Thus, given \( u \in SN_{\lambda \text{ex}} \), let us consider
\( \phi(t) := 1 + \eta_{\lambda \text{ex}}(t) + \max K_{\lambda \text{ex}}(t) \),
where \( \max K_{\lambda \text{ex}}(t) := \max \{ K(t') | t \rightarrow^*_{\lambda \text{ex}} t' \} \)

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

We can now consider the following function \( \mathcal{K}(\lambda) \) on labelled terms.

\[
\begin{align*}
\mathcal{K}(x) & := 1 \\
\mathcal{K}(\lambda x.t) & := \mathcal{K}(t) + \mathcal{K}(u) + 1
\end{align*}
\]

In order to extend \( \mathcal{K}(\lambda) \) on terms to \( \mathcal{K}(\lambda) \) on labelled terms we define a special measure for
\( \lambda \text{ex} \)-strongly normalising terms. Thus, given \( u \in SN_{\lambda \text{ex}} \), let us consider
\( \phi(t) := 1 + \eta_{\lambda \text{ex}}(t) + \max K_{\lambda \text{ex}}(t) \), where \( \max K_{\lambda \text{ex}}(t) := \max \{ K(t') | t \rightarrow^*_{\lambda \text{ex}} t' \} \)

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

We can now consider the following function \( \mathcal{K}(\lambda) \) on labelled terms.

\[
\begin{align*}
\mathcal{K}(x) & := 1 \\
\mathcal{K}(\lambda x.t) & := \mathcal{K}(t) + \mathcal{K}(u) + 1
\end{align*}
\]
Lemma 4.6. Let $t, u$ be $S$-labelled terms and let $z \notin S$. Then,

1. $t =_{a, c} u$ implies $af_z(t) = af_z(u), \text{dep}(t) = \text{dep}(u)$ and $K(t) = K(u)$.
2. $t \rightarrow^{{\complement}} u$ implies $af_z(t) = af_z(u)$ and $\text{dep}(t) > \text{dep}(u)$.
3. $t \rightarrow^{{\complement}} u$ implies $af_z(t) \geq af_z(u), \text{dep}(t) \geq \text{dep}(u)$ and $K(t) > K(u)$.

Proof. By induction on reduction. Notice that $af_z(t) > af_z(u)$ holds for example for $t = t_1[x/u_1] \rightarrow^c t_1[x/u_1] = u$, where $u_1 \rightarrow^c u_1', z \in \text{fv}(u_1)$ and $z \notin \text{fv}(u_1)$. Similarly, $\text{dep}(t) = \text{dep}(u)$ holds for example for $t \rightarrow^{\text{var}} u$, and $\text{dep}(t) > \text{dep}(u)$ holds for example for $t = t_2[z/u_2] \rightarrow^c t_2'[z/u_2] = u$, where $t_2 \rightarrow^c t_2'$. \>

Lemma 4.7. The reduction relation $e_x$ (and thus also $x$) is terminating.

Proof. Since $t \rightarrow e_x u$ implies $\langle \text{dep}(t), K(t) \rangle \succ_{lex} \langle \text{dep}(u), K(u) \rangle$ by Lemma 4.6 and $\succ_{lex}$ is a well-founded relation, then $e_x$ terminates.

Lemma 4.8. The reduction relation $\lambda e_x$ is terminating.

Proof. Lemma 4.6 guarantees that $t =_{e_x} t'$ implies $\langle \text{dep}(t), K(t) \rangle = \langle \text{dep}(t'), K(t') \rangle$. We now show that $t \rightarrow^{\lambda e_x} t'$ implies $af_z(t) \geq af_z(t')$ for $z \notin S$ and $\langle \text{dep}(t), K(t) \rangle \succ_{lex} \langle \text{dep}(t'), K(t') \rangle$. We proceed by induction on $\lambda e_x$.

- If $t = u[x/v] \rightarrow^{\lambda e_x} u'[x/v'] = t'$ comes from $v \rightarrow_{e_x} v'$, then $af_z(t) = af_z(u) = af_z(t')$, $\text{dep}(t) = \text{dep}(u) + af_z(u) = \text{dep}(t')$ and $K(t) = K(u) \cdot \phi(v) > K(u) \cdot \phi(v') = K(t')$.
- If $t \rightarrow_{\lambda e_x} t'$ comes from $t \rightarrow t'$, then conclude using Lemma 4.6.
- If $t = u[x/v'] \rightarrow^{\lambda e_x} u'[x/v'] = t'$ or $t = u[x/v] \rightarrow^{\lambda e_x} u'[x/v] = t'$ or $t = v[x/u] \rightarrow^{\lambda e_x} v'[x/u'] = t'$ or $t = uv \rightarrow^{\lambda e_x} v'u' = t'$ or $t = vu \rightarrow^{\lambda e_x} vu' = t'$ or $t = \lambda x.u \rightarrow^{\lambda e_x} \lambda x.u' = t'$ comes from $u \rightarrow_{\lambda e_x} u'$, 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 $t[x/u]v_n$ and $t[x/u]v_n$. More precisely, we split the IE Property in two different steps:

- Show that $u \in SN_{\lambda e_x} \& t[x/u]v_n \in SN_{\lambda e_x}$ imply $t[x/u]v_n \in SN_{\lambda e_x}$.
- Show that $t[x/u]v_n \in SN_{\lambda e_x}$ implies $t[x/u]v_n \in SN_{\lambda e_x}$.

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

\[
\begin{align*}
xc(x) &:= x \\
xc(tu) &:= xc(t)xc(u) \\
xc(\lambda y.t) &:= \lambda y.xc(t) \\
xc(t[x/u]) &:= xc(t)[x/xc(u)] \\
xc(t[x/v]) &:= xc(t)[x/v]
\end{align*}
\]

Notice that $xc(t) = t$ if $t$ is a term.

Lemma 5.1. Let $t, t'$ be labelled terms. If $t \rightarrow e_x t'$, then $xc(t) = xc(t')$.

Proof. By induction on $t \rightarrow e_x t'$. The interesting case is $t = s[x/u][y/v] = s[y/v][x/u] = t'$, with $y \notin \text{fv}(u)$ & $x \notin \text{fv}(v)$. The term $xc(t)$ is equal to $xc(s)[x/xc(u)]{y/v} = xc(s){y/v}[x/xc(u)] = xc(t')$.
Lemma 5.2 (Projecting $\lambda_{\text{ex}}$). Let $t, t'$ be labelled terms. Then,

1. $t =_{\alpha, \beta, \gamma} t'$ implies $xc(t) = xc(t')$.
2. $t \rightarrow_{\lambda_{\text{ex}}} t'$ implies $xc(t) \rightarrow_{\lambda_{\text{ex}}} xc(t')$.
3. $t \rightarrow_{\lambda_{\text{ex}}} t'$ implies $xc(t) \rightarrow_{t_{\lambda_{\text{ex}}}} xc(t')$.

Proof.

(1) By induction on the conversion relation.

(2) Internal reduction:
- If $u[x/v] \rightarrow_{\lambda_{\text{ex}}} u[x/v']$ comes from $v \rightarrow_{\text{ex}} v'$, then $xc(u[x/v]) = xc(u[x/v']) = xc(u[x/v'])$.
- If $t \rightarrow_{\lambda_{\text{ex}}} t'$ comes from $t \rightarrow_{\text{ex}} t'$ (so that also $t \rightarrow_{\text{ex}} t'$), then Lemma 5.1 gives $xc(t) = xc(t')$.
- If $uv \rightarrow_{\lambda_{\text{ex}}} u'v$ where $u \rightarrow_{\lambda_{\text{ex}}} u'$, then $xc(u) = xc(u') = xc(u')$.
- If $u[x/v] \rightarrow_{\lambda_{\text{ex}}} u'[x/v]$ where $u \rightarrow_{\lambda_{\text{ex}}} u'$, then $xc(u[x/v]) = xc(u'[x/v]) = xc(u'[x/v])$.
- The other cases are similar since $xc$ does not alter application, lambda and substitution.

(3) External reduction:
- If $t \rightarrow_{\lambda_{\text{ex}}} t'$ comes from a reduction $t \rightarrow_{\text{ex}} t'$ which occurs outside a labelled substitution, then $xc(t) \rightarrow_{t_{\lambda_{\text{ex}}}} xc(t')$ can be shown by induction on $t \rightarrow_{\text{ex}} t'$ using Lemma 2.1.
- If $tu \rightarrow_{\lambda_{\text{ex}}} tu'$, $ut \rightarrow_{\lambda_{\text{ex}}} ut'$, $\lambda x.t \rightarrow_{\lambda_{\text{ex}}} \lambda x.t'$, $t[x/u] \rightarrow_{\lambda_{\text{ex}}} t'[x/u]$ or $u[x/t] \rightarrow_{\lambda_{\text{ex}}} u'[x/t']$ comes from $t \rightarrow_{\lambda_{\text{ex}}} t'$, then $xc(t) \rightarrow_{t_{\lambda_{\text{ex}}}} xc(t')$ by the i.h. and thus the property holds by definition of $xc$ and the fact that $xc$ does not alter application, lambda and substitution.
- If $t[x/u] \rightarrow_{\lambda_{\text{ex}}} t'[x/u]$ comes from $t \rightarrow_{\lambda_{\text{ex}}} t'$, then $xc(t[x/u]) = xc(t'[x/u]) = xc(t'[x/u])$.

Lemma 5.3. Let $t$ be a labelled term. If $xc(t) \in SN_{\lambda_{\text{ex}}}$, then $t \in SN_{\lambda_{\text{ex}}}$.

Proof. We apply the Abstract Theorem A.2 in the Appendix A by taking $A_1 = \lambda_{\text{ex}}$, $A_2 = \lambda_{\text{ex}}$, $A = \lambda_{\text{ex}}$ and $u \rightarrow_{U} u$ if $xc(u) = U$. Lemma 5.2 guarantees properties P1 and P2 and Lemma 1.8 guarantees property P3. We then get that $xc(t) \in SN_{\lambda_{\text{ex}}}$ implies $t \in SN_{\lambda_{\text{ex}}}$, which is exactly $SN_{\lambda_{\text{ex}}}$ by Lemma 1.3. We thus conclude.

Corollary 5.4. Let $t, u, \overline{u}_n$ be terms. If $u \in SN_{\lambda_{\text{ex}}} \& t[x/u] \overline{u}_n \in SN_{\lambda_{\text{ex}}}$, then $t[x/u] \overline{u}_n \in SN_{\lambda_{\text{ex}}}$.

Proof. Take $S = f\overline{v}(u)$. The hypothesis $u \in SN_{\lambda_{\text{ex}}}$ allows us to construct the $S$-labelled term $t[x/u] \overline{u}_n$. Moreover, $xc(t) = t$ so that $xc(t[x/u] \overline{u}_n) = t[x/u] \overline{u}_n$ and we thus conclude by Lemma 5.3.

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

Definition 5.5 (Unlabelling). Unlabelling of labelled terms is defined by induction.
Lemma 5.6. Let $t \in \mathcal{L}_\lambda$ s.t. $\mathbb{U}(t) \rightarrow_{\lambda_{\mathbf{ex}}} t_1'$. Then $\exists t_1 \in \mathcal{L}_\lambda$ s.t. $t \rightarrow_{\lambda_{\mathbf{ex}}} t_1$ and $\mathbb{U}(t_1) = t_1'$.

Proof. By induction on $\rightarrow_{\lambda_{\mathbf{ex}}}$ and case analysis. The interesting cases are the following.

1. $t = u[x/v][y/w]$ where $y \in \mathcal{FV}(v)$, and

   $\mathbb{U}(u[x/v][y/w]) = \mathbb{U}(u)[x/\mathbb{U}(v)][y/w] \rightarrow_{\text{comp}} \mathbb{U}(u)[y/w][x/\mathbb{U}(v)][y/w] = t_1'$

   We then let $t_1 = u[y/w][x/v][y/w]$ so that $\mathbb{U}(t_1) = t_1'$ and $t \rightarrow_{\text{comp}} t_1$.

2. $t = u[y/w][x/v]$. By $\alpha$-conversion we can always choose $x \not\in \mathbb{S}$, which is a fixed set of variables, so that we necessarily have $x \not\in \mathcal{FV}(w)$ since $\mathcal{FV}(w) \subseteq \mathbb{S}$ by construction. Now, consider

   $\mathbb{U}(u[y/w][x/v]) = \mathbb{U}(u)[y/w][x/\mathbb{U}(v)] = t_1'$

   We then let $t_1 = u[y/w][x/v]$ so that $\mathbb{U}(t_1) = t_1'$ and $t =_\mathbb{S} t_1$.

3. $t = u[x_1/v_1][x_2/v_2]$. Again, by $\alpha$-conversion we can assume $x_i \not\in \mathbb{S}$ so that $x_i \not\in \mathcal{FV}(v_j)$ since $\mathcal{FV}(v_i) \subseteq \mathbb{S}$ by construction. Now, consider

   $\mathbb{U}(u[x_1/v_1][x_2/v_2]) = \mathbb{U}(u)[x_1/v_1][x_2/v_2][x_1/v_1] = \mathbb{U}(u)[x_2/v_2][x_1/v_1] = \mathbb{U}(u)[x_2/v_2][x_1/v_1] = t_1'$

   We then let $t_1 = u[x_2/v_2][x_1/v_1]$ so that $\mathbb{U}(t_1) = t_1'$ and $t =_\mathbb{S} t_1$.

All the other cases are straightforward. $\square$

Lemma 5.7. Let $t \in \mathcal{L}_\lambda$. If $t \in SN_{\lambda_{\mathbf{ex}}}$, then $\mathbb{U}(t) \in SN_{\lambda_{\mathbf{ex}}}$.

Proof. We prove $\mathbb{U}(t) \in SN_{\lambda_{\mathbf{ex}}}$ by induction on $\eta_{\lambda_{\mathbf{ex}}}(t)$. This is done by considering all the $\lambda_{\mathbf{ex}}$-reducts of $\mathbb{U}(t)$ and using Lemma [5.6]. $\square$

Taking $\mathbb{S} = \mathcal{FV}(u)$ and transforming the term $s[x/u][v_n]$ into the $\mathbb{S}$-labelled term $s[x/u][v_n]$ we have the following special case.

Corollary 5.8. If $t[x/u][v_n] \in SN_{\lambda_{\mathbf{ex}}}$, then $t[x/u][v_n] \in SN_{\lambda_{\mathbf{ex}}}$.

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

Lemma 5.9 (IE Property). Let $t, u, v_n$ be terms. If $u \in SN_{\lambda_{\mathbf{ex}}}$ & $t[x/u][v_n] \in SN_{\lambda_{\mathbf{ex}}}$, then $t[x/u][v_n] \in SN_{\lambda_{\mathbf{ex}}}$.

Proof. By Corollaries 5.4 and 5.8. $\square$
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 \( \lambda \)ex-calculus, and obtain a characterisation of the set of \( \lambda \)ex-strongly normalising terms by means of an intersection type system.

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

\[
A ::= \sigma \ (\text{atomic}) \mid A \rightarrow A \mid A \cap A
\]

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 \( \text{System } \cap \), is defined by means of the set of typing rules in Figure 3.

\[
\begin{array}{ll}
\Gamma \vdash x : A \vdash x : A & \text{(ax)} \\
\Gamma \vdash t : A \rightarrow B, \Gamma \vdash u : A & \Gamma \vdash tu : B \quad \text{(app)} \\
\Gamma \vdash \lambda x.t : A \rightarrow B & \text{(abs)} \\
\Gamma \vdash u : B, \Gamma \vdash x : B \vdash t : A & \Gamma \vdash t[x/u] : A \quad \text{(subs)} \\
\Gamma \vdash t : A, \Gamma \vdash t : B & \Gamma \vdash t : A \cap B \quad \text{(\cap I)} \\
\Gamma \vdash t : A_1 \cap A_2 & \Gamma \vdash t : A_i \quad \text{(\cap E)}
\end{array}
\]

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 \{ax, abs, app, \cap I, \cap 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 \( t \in SN_{\beta} \).

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 \( \Gamma \vdash t : B \) and \( B \ll A \), then \( \Gamma \vdash t : A \).

Proof. Let \( \Gamma \vdash t : B \). 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 \( \Gamma \vdash t : C \) and then \( \Gamma \vdash t : A \).
Case \( B \ll B_1, B \ll B_2, A = B_1 \cap B_2 \): Use (twice) the i.h. to get \( \Gamma \vdash t : B_1 \) and \( \Gamma \vdash t : B_2 \), then apply \( \cap I \).

We use the notation \( \overline{n} \) 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) \( \Gamma \vdash \cap_n x : A \) iff there is \( x : B \in \Gamma \) and \( B \ll A \).
(2) \( \Gamma \vdash t[x/u] : A \) iff there exist \( A_i, B_i \) (\( i \in \overline{n} \)) s.t. \( \cap_n A_i \ll A \) and \( \forall i \in \overline{n}, \Gamma \vdash u : B_i \) and \( \Gamma, x : B_i \vdash \cap_t : A_i \).
(3) \( \Gamma \vdash \cap t u : A \) iff there exist \( A_i, B_i \) (\( i \in \overline{n} \)) s.t. \( \cap_n A_i \ll A \) and \( \forall i \in \overline{n}, \Gamma \vdash t : B_i \rightarrow A_i \) and \( \Gamma \vdash u : B_i \).
(4) \( \Gamma \vdash \cap_n \lambda x.t : A \) iff there exist \( A_i, B_i \) (\( i \in \overline{n} \)) s.t. \( \cap_n (A_i \rightarrow B_i) \ll A \) and \( \forall i \in \overline{n}, \Gamma, x : A_i \vdash t : B_i \).
(5) \( \Gamma \vdash \cap \lambda x.t : B \rightarrow C \) 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

$$\Gamma \vdash x : C_1 \quad \Gamma \vdash x : C_2 \quad (\cap I)$$

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

$$\Gamma \vdash x : A \cap A' \quad (\cap E)$$

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 $\Gamma \vdash t[x/u] : A$.

• Suppose the root of the derivation is

$$\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A \quad (\text{subs})$$

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

$$\Gamma \vdash t[x/u] : C_1 \quad \Gamma \vdash t[x/u] : C_2 \quad (\cap I)$$

By the i.h. there are $A_i, B_i \ (i \in \mathbb{N})$ s.t. $\cap_n A_i \ll C_1$ and $\Gamma \vdash u : B_i$ and $\Gamma, x : B_i \vdash t : A_i$ for all $i \in \mathbb{N}$. Also there are $A'_i, B'_i \ (i \in \mathbb{N'})$ s.t. $\cap_{n'} A'_i \ll C_2$ and $\Gamma \vdash u : B'_i$ and $\Gamma, x : B'_i \vdash t : A'_i$ for all $i \in \mathbb{N'}$. Since $\cap_n A_i \cap_{n'} A'_i \ll C_1 \cap C_2$, this concludes this case.

• Suppose the root of the derivation is

$$\Gamma \vdash t[x/u] : A \cap B \quad (\cap E)$$

By the i.h. there are $A_i, B_i \ (i \in \mathbb{N})$ 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 $i \in \mathbb{N}$. 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 $\Gamma \vdash \lambda x.t : B \to C$, then point 4 gives $\Gamma, x : B_i \vdash t : C_i$ 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 $j \in \mathbb{N}$, thus $\Gamma, x : B \vdash t : C$. \hfill \Box

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

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

- $V(x) := x$
- $V(tu) := V(t)V(u)$
- $V(\lambda x.t) := \lambda x.V(t)$
- $V(\lambda x.t[x/u]) := \lambda x.V(t[\lambda x.V(t)]V(u))$

This function is compositional with respect to substitution:

**Lemma 6.7.** Let $t, u$ be terms. Then $V(t)[x/V(u)] = V(t[x/u])$.

**Proof.** By induction on $t$. \hfill \Box
The function $V(\_)$ does not modify typability.

**Lemma 6.8.** Let $t$ be a term. Then $\Gamma \vdash \_ \vdash A$ iff $\Gamma \vdash t : A$.

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

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

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

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

**Lemma 6.10.** Let $t$ be a term s.t. $V(t) \to^1_\beta t'_1$. Then, $\exists t_1$ s.t. $t \to^+_\lambda ex t_1$ and $t'_1 = V(t_1)$.

**Proof.** By induction on the reduction step $V(t) \to^1_\beta t'_1$.

- If $V((\lambda x. u) v) = (\lambda x. V(u))V(v) \to^1_\beta V(u)(x/V(v))$, then let $t_1 = u(x/v)$. We have $(\lambda x. u) v \to^1_\beta u(x/v) \to^+_\lambda ex t_1$. We conclude by Lemma 6.7.

- If $V(u[x/v]) = (\lambda x. V(u))V(v) \to^1_\beta V(u)(x/V(v))$, then again we conclude by letting $t_1 = u(x/v)$.

- If $V(u[x/v]) = (\lambda x. V(u))V(v) \to^1_\beta (\lambda x. u'_1) V(v)$, where $V(u) \to^1_\beta u'_1$ then the i.h. gives $u_1 \to^+_\lambda ex u_1$. Let $t_1 = u_1[x/v]$. We have $u[x/v] \to^+_\lambda ex u_1[x/v]$ and $(\lambda x. u'_1) V(v) = V(u_1[x/v])$.

- If $V(u[x/v]) = (\lambda x. V(u))V(v) \to^1_\beta (\lambda x. V(u))v'_1$, where $V(v) \to^1_\beta v'_1$, then proceed as in the previous one.

- All the other cases are straightforward.

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

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

Now, $V(t) \in SN_\beta$ implies that $V(t)$ 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 $t \in SN_{\lambda ex}$.

We conclude this section by focusing on the particular case of the *simply typed $\lambda ex$-calculus*: types are only built over atomic symbols and functional types so that the type system only contains the typing rules $\{ax, abs, app, subs\}$ 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 $\lambda ex$-calculus is $\lambda ex$-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 $t^A, u^B \in SN_{\lambda ex}$, then $t\{x^B/u^B\} \in SN_{\lambda ex}$.

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

• \( t = x \). Then \( x[x/u] = u \in \text{SN}_{\text{lex}} \) by the hypothesis.

• \( t = y^n \) with \( x \neq y \) and \( n \geq 0 \). The i.h. gives \( v_i[x/u] \in \text{SN}_{\text{lex}} \) since \( \eta_{\text{lex}}(v_i) \) decreases and \( v_i \) is strictly smaller than \( t \). Then we conclude by Definition 3.4 and Proposition 3.5.

• \( t = xu^n \). The i.h. gives \( V = v[x/u] \) and \( V_i = v_i[x/u] \) in \( \text{SN}_{\text{lex}} \). We show \( t[x/u] = uV^n \in \text{SN}_{\text{lex}} \) by induction on \( \eta_{\text{lex}}(u) + \eta_{\text{lex}}(V) + \sum_{i=1}^{\ldots n} \eta_{\text{lex}}(V_i) \). For that, it is sufficient to show that all its reducts are in \( \text{SN}_{\text{lex}} \). If the reduction takes place in a subterm of \( u, V, V_n \), then we conclude by the i.h. Otherwise, suppose \( u = \lambda y. U \) and \( (\lambda y. U)V^n \to U[y/V]V_n \). Then type\( V \) = type\( u \) = type\( x \) so that \( U(y/V) \in \text{SN}_{\text{lex}} \) by the i.h. Let us write \( U(y/V) \equiv_n (zV_n)\{z/U[y/V]\} \). We have type\( U(y/V) \) = type\( U \) < type\( u \) so that again by the i.h. we get \( U(y/V) \equiv_n \in \text{SN}_{\text{lex}} \). We conclude \( U[y/V]V_n \in \text{SN}_{\text{lex}} \) by Definition 3.4 and Proposition 3.5.

• \( t = \lambda y. v \). Then \( v[x/u] \in \text{SN}_{\text{lex}} \) by the i.h. and thus \( t[x/u] = \lambda x. v[x/u] \in \text{SN}_{\text{lex}} \) follows from Definition 3.4 and Proposition 3.5.

• \( t = (\lambda y.s)v^n \). The i.h. gives \( S = s[x/u] \), \( V = v[x/u] \) and \( V_i = v_i[x/u] \) in \( \text{SN}_{\text{lex}} \). To show \( t[x/u] = (\lambda y.S)V^n \in \text{SN}_{\text{lex}} \) we reason by induction on \( \eta_{\text{lex}}(S) + \eta_{\text{lex}}(V) + \sum_{i=1}^{\ldots n} \eta_{\text{lex}}(V_i) \). For that, it is sufficient to show that all its reducts are in \( \text{SN}_{\text{lex}} \). If the reduction takes place in a subterm of \( (\lambda y.S)V^n \), \( V, V_n \), we conclude by the i.h. Otherwise suppose \( (\lambda y.S)V^n \to S[y/V]V_n \). Take \( T = s[y/v]V_n \). Since \( \eta_{\text{lex}}(T) < \eta_{\text{lex}}(t) \), then the i.h. gives \( T[x/u] \in \text{SN}_{\text{lex}} \). But \( S[y/V]V_n = T[x/u] \) so that \( S[y/V]V_n \in \text{SN}_{\text{lex}} \).

• \( t = (\lambda y. S)v^n \). The i.h. gives \( S = s[x/u] \) and \( V = v[x/u] \) and \( V_i = v_i[x/u] \) are in \( \text{SN}_{\text{lex}} \). They are also typed. We claim \( t[x/u] = S[y/V]V_n \in \text{SN}_{\text{lex}} \). The perpetual strategy gives

\[
t[x/u] = S[y/V]V_n \sim S[y/V]V_n
\]

This last term can be written as \( T[x/u] \) where \( T = s[y/v]V_n \). Since \( \eta_{\text{lex}}(T) < \eta_{\text{lex}}(t) \), then the i.h. gives \( T[x/u] \in \text{SN}_{\text{lex}} \) and thus Theorem 3.3 gives \( S[y/V]V_n \in \text{SN}_{\text{lex}} \).

**Corollary 6.15** (Simply Typed Terms are SN - Second Proof). *Simply typed lex-calculus is lex-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 \( u, v \in \text{SN}_{\text{lex}} \) by the i.h. We write \( t = (\lambda z. v)\{z/u\} \), where \( zv \in \text{SN}_{\text{lex}} \) by Definition 3.4. The term \( zv \) is also appropriately typed. Lemma 6.14 then gives \( t \in \text{SN}_{\text{lex}} \). If \( t = u[x/v] \), then \( u, v \) are typed and by the i.h. \( u, v \in \text{SN}_{\text{lex}} \) so that Lemma 6.14 gives \( u[x/v] \in \text{SN}_{\text{lex}} \). Definition 3.4 and Proposition 3.5 allow us to conclude \( u[x/v] \in \text{SN}_{\text{lex}} \).

**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 \( \text{lex} \).

- The \( \lambda x \)-calculus [Lin86, Lin92, Ros92] is just a sub-calculus of \( \text{lex} \), with no equation and no composition rule. Thus, the fact that \( t \rightarrow 1 x t' \) implies \( t \rightarrow 1 x t' \) is straightforward. Since simply typed terms in both calculi are the same, we thus deduce that typed terms are \( \text{Ax} \)-strongly normalising.

- The \( \lambda \text{es} \)-calculus [Kes07] can be seen as a refinement of \( \text{lex} \), 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 \( t \rightarrow \lambda x t' \) implies \( t \rightarrow 1 x t' \) is
straightforward. Simply typed terms in both calculi are the same, we thus deduce that typed terms are \(\lambda\text{es}\)-strongly normalising.

- Milner’s calculus with explicit partial substitution [Mil06], called \(\lambda_{\text{sub}}\), is able to encode \(\lambda\)-calculus in terms of a bigraphical reactive system. The operational semantics of \(\lambda_{\text{sub}}\) 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 \(T\) from terms to terms such that \(t \rightarrow \lambda_{\text{sub}} t'\) implies \(T(t) \rightarrow^+ \lambda\text{es} T(t')\). Since simply typed terms in both calculi are the same, we conclude that typed terms are \(\lambda_{\text{sub}}\)-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 \(\lambda_{\text{sub}}\) since arguments are consumed on only one occurrence at a time. Similarly to [KC] one can define a translation \(T\) from \(\lambda\)-terms to terms such that one-step reduction in \(\lambda_{\beta_p}\) is projected into at least one-step reduction in \(\lambda_{\text{sub}}\). 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 \(\lambda_{\text{ws}}\), which allows controlled composition of ES without losing PSN. The calculus \(\lambda_{\text{ws}}\) 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 \(\lambda_{\text{ws}}\) implies at least one-step reduction in \(\lambda\text{ex}\). Thus, SN for typed terms in \(\lambda_{\text{ws}}\) is a consequence of SN for typed \(\lambda\text{ex}\).

- A calculus with a safe notion of composition in director string notation is defined in [SFIM03]. The named version of this calculus can be understood as the \(\lambda\text{x}\)-calculus together with a composition rule of the form:

\[
t[x/u][y/v] \rightarrow t[x/u[y/v]] \text{ if } y \in \text{fv}(u) \cup y \notin \text{fv}(t)
\]

This composition rule can be easily simulated by the rules Comp and Gc of the \(\lambda\text{ex}\)-calculus so that the whole calculus can be simulated by \(\lambda\text{ex}\). As a consequence, simply typed terms turn out to be strongly normalising.

- The \(\lambda\text{esw}\)-calculus [Kes07] was used as a technical tool to show that \(\lambda\text{es}\) enjoys PSN. The syntax extends terms with weakening constructors so that it is straightforward to define a translation \(T\) from \(\lambda\text{esw}\)-terms to terms which forgets these weakening operators. The reduction relation \(\lambda\text{esw}\) can be split into an equational system \(E\) and two rewriting relations \(L_1\) and \(L_2\) s.t.

1. If \(t =_E t'\) or \(t \rightarrow_{L_1} t'\) then \(T(t) =_E T(t')\)
2. If \(t \rightarrow_{L_2} t'\) then \(T(t) \rightarrow^+_{\lambda\text{ex}} T(t')\)

The reduction relation generated by the rules \(L_1\) modulo the equations \(E\) can be easily shown to be terminating. Also, simply typed \(\lambda\text{esw}\)-terms trivially translate via \(T\) to simply typed terms. Thus, the Abstract Theorem given in the Appendix [A] allows us to conclude that typed \(\lambda\text{esw}\)-terms are \(\lambda\text{esw}\)-strongly normalising.

8. Confluence

In this section we study confluence of the \(\lambda\text{ex}\)-calculus. More precisely, we show confluence of the relation \(\rightarrow_{\lambda\text{ex}}\) 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 \( X, Y, \ldots \). To each raw metavariable \( X \), we associate a set of variables \( \Delta \), thus yielding a decorated metavariable denoted by \( X_\Delta \). Thus for example \( X_{x,y,z} \) and \( Y_{x,z} \) 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.

\[
\mathcal{M} ::= x \mid X_\Delta \mid \mathcal{M} \mid \lambda x.\mathcal{M} \mid \mathcal{M}[x/\mathcal{M}]
\]

Notice that terms are in particular metaterms.

We extend the notion of free variables to metaterms by \( \text{fv}(X_\Delta) := \Delta \). 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 \( \lambda x. Y_x X_{x,y} =_\alpha \lambda z. Y_x X_{z,y} \).

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

\[
X_\Delta \{ x/v \} := X_\Delta \quad \text{if} \ x \notin \Delta
\]
\[
X_\Delta \{ x/v \} := X_\Delta[x/v] \quad \text{if} \ x \in \Delta
\]

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

**Proof.** By induction on \( t \). \( \square \)

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 \( x \notin \text{fv}(v) \). Then \( t \{ x/u \} \{ y/v \} =_e t \{ y/v \} \{ x/u \{ y/v \} \} \).

**Proof.** By induction on metaterms using Lemma 8.1. Notice that \( =_e \) is needed for the case where \( t \) is a metavariable. \( \square \)

Reduction on metaterms must be understood in the same way reduction on terms: the \( \lambda ex \)-relation is generated by the \( \rightarrow_{Bx} \)-reduction relation on \( e \)-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 \( R \in \{ x, ex, \lambda x, \lambda ex \} \), if \( t \rightarrow_R t' \), then \( u \{ x/t \} \rightarrow^*_R u \{ x/t' \} \) and \( t \{ x/u \} \rightarrow_R t' \{ x/u \} \). Thus in particular \( t \{ x/u \} \in \text{SN}_R \) implies \( t \in \text{SN}_R \).

**Proof.** By induction on \( t \rightarrow t' \). \( \square \)

**Lemma 8.4** (Full Composition for Metaterms). Let \( t, u \) be metaterms. Then \( t \{ x/u \} \rightarrow^*_ex t \{ x/u \} \).

**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 \( t = X_\Delta \) with \( x \in \Delta \) since \( X_\Delta \{ x/u \} = X_\Delta \{ x/u \} \). \( \square \)
It is well-known that confluence on metaterms fails for calculi without composition for ES as for example the following critical pair in the $\lambda x$-calculus shows

\[
s = t[x/u][y/v] \rightsquigarrow ((\lambda x.t) u)[y/v] \rightarrow^*_\Delta t[y/v][x/u[y/v]] = s'.
\]

Indeed, while this diagram can be closed in $\lambda x$ 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 $\lambda x$-reduction rule is able to mimic composition on raw/decorated metavariables. Fortunately, this diagram can be closed in the $\lambda \text{ex}$-calculus as follows. If $y \in \mathsf{fv}(u)$, then $s \rightarrow^*_\text{comp} s'$, otherwise $s' \rightarrow^*_\text{ex} (L.8.7.4) t[y/v][x/u] =_c s'$.

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 $\rightarrow_R$ iff $t \rightarrow_R u$ implies $u \rightarrow_R t^o$ and $t^o \rightarrow_R u^o$. A reduction relation $\rightarrow_R$ has the Z-property if there is a map which satisfies the Z-property for $\rightarrow_R$.

It turns out [VO] that $\rightarrow_R$ is confluent if it has the Z-property (see Theorem A.1 in the Appendix A), so to show confluence of $\lambda \text{ex}$ it is then sufficient to define a map on metaterms satisfying the Z-property. Such a map can be defined in terms of the superdevelopment function for the $\lambda$-calculus [AC78, MR93].

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

\[
\begin{align*}
\mathbb{K}_\Delta^o & := \mathbb{K}_\Delta, & (tu)^o & := t^o u^o & \text{if } t^o \text{ is not an abstraction} \\
x^o & := x, & (tu)^o & := v(x/u)^o & \text{if } t^o = \lambda x.v \\
(\lambda x.t)^o & := \lambda x.t^o & t[x/u]^o & := t^o(x/u^o)
\end{align*}
\]

Notice that $\mathsf{fv}(t^o) \subseteq \mathsf{fv}(t)$.

**Lemma 8.7.** Let $t, u$ be metaterms. Then $t^o u^o \rightarrow^*_\text{ex} (tu)^o$.

**Proof.** If $t^o$ is not an abstraction, then $t^o u^o = (tu)^o$. If $t^o = \lambda y.s$, then $t^o u^o = (\lambda y.s) u^o \rightarrow_B s[y/u]^o \rightarrow^*_\text{ex} (L.8.3) s\{y/u^o\} = (tu)^o$. \hfill \hfill \square

**Lemma 8.8.** Let $t, u$ be metaterms. Then $t^o \{x/u^o\} \rightarrow^*_\text{ex} t\{x/u\}^o$.

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

- If $v^o$ is not an abstraction, then

\[
(vw)^o\{x/u^o\} = v^o\{x/u^o\}w^o\{x/u^o\} \rightarrow^*_\text{ex} (L.8.7) v\{x/u\}^o w\{x/u\}^o \rightarrow^*_\text{ex} (vw)\{x/u\}^o
\]

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

\[
\begin{align*}
(vw)^o\{x/u^o\} &= \lambda z^o\{x/u^o\} \rightarrow^*_\text{ex} (L.8.2) r\{x/u\}^o z^o\{x/u^o\} \\
r\{x/u\}^o z^o\{x/u^o\} &\rightarrow^*_\text{ex} (L.8.3) s\{z/w^o\{x/u^o\}\} \\
&\rightarrow^*_\text{ex} (L.8.3) s\{z/w\{x/u\}^o\} = (v\{x/u\} w\{x/u\})^o = (vw)\{x/u\}^o
\end{align*}
\]
The case \( t = v[y/w] \) also uses the i.h. and Lemma 8.2. All the other cases are straightforward.

**Lemma 8.9.** Let \( t \) be a metaterm. Then \( t \xrightarrow{\ast_{\text{ex}}} t^0 \).

**Proof.** By induction on \( t \). The interesting cases are the following ones.

- If \( t = uv \): Then \( uv \xrightarrow{\ast_{\text{ex}} \text{ (i.h.)}} u^0v^0 \xrightarrow{\ast_{\text{ex}}} (uv)^0 = t^0 \).
- If \( t = u[x/v] \): Then \( u[x/v] \xrightarrow{\ast_{\text{ex}} \text{ (i.h.)}} u^0[x/v]^0 \xrightarrow{\ast_{\text{ex}}} (L.8.4) u^0 \{x/v^0\} \xrightarrow{\ast_{\text{ex}}} (L.8.8) u \{x/v\}^0 \).

All the other cases are straightforward.

**Lemma 8.10** (Towards the Z-Property). Let \( t, u \) be metaterms. If \( t \xrightarrow{Bx} u \), then \( u \xrightarrow{\ast_{\text{ex}}} t^0 \xrightarrow{\ast_{\text{ex}}} u^0 \).

**Proof.** By induction on \( t \xrightarrow{Bx} u \).

- If \( t = \lambda x.r \xrightarrow{Bx} \lambda x.s = u \), where \( r \xrightarrow{Bx} s \), then the property holds by the i.h.
- If \( t = r[x/v] \xrightarrow{Bx} s[x/v] = u \), where \( r \xrightarrow{Bx} s \), then
  \[
  u = s[x/v] \xrightarrow{\ast_{\text{ex}} \text{ (i.h.)}} r^0[x/v] \xrightarrow{\ast_{\text{ex}}} (L.8.4) r^0[x/v^0] \xrightarrow{\ast_{\text{ex}}} (L.8.4) r^0 \{x/v^0\} = t^0 \xrightarrow{\ast_{\text{ex}}} (L.8.8) s^0 \{x/v^0\} = u^0.
  \]
- If \( t = v[x/r] \xrightarrow{Bx} v[x/s] = u \), where \( r \xrightarrow{Bx} s \), then proceed as in the previous case.
- If \( t = rv \xrightarrow{Bx} sv = u \), where \( r \xrightarrow{Bx} s \), then \( sv \xrightarrow{\ast_{\text{ex}} \text{ (i.h.)}} r^0v^0 \xrightarrow{\ast_{\text{ex}}} (L.8.9) r^0v^0 \xrightarrow{\ast_{\text{ex}}} (L.8.7) (rv)^0 \).
  
  For the second part of the statement there are two cases:
  - If \( r^0 \) is not an abstraction, then \( (rv)^0 = r^0v^0 \xrightarrow{\ast_{\text{ex}}} (L.8.9) s^0v^0 \xrightarrow{\ast_{\text{ex}}} (L.8.7) (sv)^0 \).
  - If \( r^0 = \lambda z.w \), then the i.h. \( r^0 \xrightarrow{\ast_{\text{ex}}} s^0 \) implies \( s^0 = \lambda z.q \), where \( w \xrightarrow{\ast_{\text{ex}}} q \). We conclude with \( (rv)^0 = w[z/v^0] \xrightarrow{\ast_{\text{ex}}} (L.8.9) q[z/v^0] = (sv)^0 \).
- If \( t = vr \xrightarrow{Bx} vs = u \), where \( r \xrightarrow{Bx} s \), then \( vs \xrightarrow{\ast_{\text{ex}} \text{ (i.h.)}} vr^0 \xrightarrow{\ast_{\text{ex}}} (L.8.9) vr^0 \xrightarrow{\ast_{\text{ex}}} (L.8.7) (vr)^0 \).
  For the second part of the statement there are two cases:
  - If \( v^0 \) is not an abstraction, then \( (vr)^0 = v^0r^0 \xrightarrow{\ast_{\text{ex}} \text{ (i.h.)}} v^0s^0 \xrightarrow{\ast_{\text{ex}}} (L.8.9) (vs)^0 \).
  - If \( v^0 = \lambda y.w \), then \( (vr)^0 = w[y/r^0] \xrightarrow{\ast_{\text{ex}}} (L.8.9) w[y/s^0] = (vs)^0 \).
- If \( t = x[x/v] \xrightarrow{\ast_{\text{ex}}} v = u \), then \( x[x/v]^0 = x \{x/v^0\} = v^0 \). We conclude since \( v \xrightarrow{\ast_{\text{ex}}} v^0 \) holds by Lemma 8.9.
- If \( t = r[x/v] \xrightarrow{\ast_{\text{ex}}} r = u \), then \( r[x/v]^0 = r^0 \{x/v^0\} = (L.8.1) r^0 \). We conclude since \( r \xrightarrow{\ast_{\text{ex}}} r^0 \) holds by Lemma 8.9.
- If \( t = (rs)[x/v] \xrightarrow{\text{App}} r[x/v]s[x/v] = u \), then
  \[
  u \xrightarrow{\ast_{\text{ex}}} (L.8.9) r^0[x/v^0]s^0 \{x/v^0\} \xrightarrow{\ast_{\text{ex}}} (L.8.3) r^0 \{x/v^0\} s^0 \{x/v^0\} = \lambda x.r \{x/v^0\} \xrightarrow{\ast_{\text{ex}}} (L.8.8) (rs)^0 \{x/v^0\} = (rs)[x/v]^0 = t^0
  \]

For the second part there are two cases.

- If \( r^0 \) is not an abstraction, then
  \[
  t^0 = r^0 \{x/v^0\} s^0 \{x/v^0\} = r[x/v]^0 s[x/v]^0 \xrightarrow{\ast_{\text{ex}}} (L.8.7) (r[x/v]s[x/v])^0 = u^0
  \]
– If \( r^o = \lambda y.q \), then \( r\{x/v\}_o = \lambda y.q\{x/v^o\} \), so that
\[
\begin{align*}
t^o &= (rs)[x/v]_o \\
&= (rs)^o\{x/v^o\} \\
&= q\{y/s^o\}\{x/v^o\} = \_e (L.8.2) q\{x/v^o\}\{y/s\{x/v\}_o\} = \\
&= (r[x/v]s[x/v])_o = u^o
\end{align*}
\]

• If \( t = (\lambda y.r)[x/v] \rightarrow_{\text{Lamb}} \lambda y.r[x/v] = u \), then \( (\lambda y.r)[x/v^o] = \lambda y.r^o\{x/v^o\} \). We have \( u = \lambda y.r[x/v] \rightarrow^{*_{\text{lex}}} (L.8.9) \lambda y.r^o[x/v^o] \rightarrow^{*_{\text{ex}}} (L.8.1) \lambda y.r^o[x/v^o] = t^o = u^o \)

• If \( t = r[x/v][y/w] \rightarrow_{\text{Comp}} r[y/w][x/v][y/w] = u \), then
\[
\begin{align*}
u &= r[y/w][x/v][y/w] \\
r^o[y/w^o][x/v^o][y/w^o] \rightarrow^{*_{\text{ex}}} (L.8.9) \\
r^o\{y/w^o\}\{x/v^o\}(y/w^o) = \_e (L.8.2) r^o\{x/v^o\}(y/w^o) = t^o
\end{align*}
\]

Since \( u^o = r^o\{y/w^o\}\{x/v^o\}(y/w^o) \), then we have \( t^o \rightarrow^{*_{\text{lex}}} u^o \) as well. \( \Box \)

**Lemma 8.11.** Let \( t, u \) be metaterms s.t. \( t =_e u \). Then,

• If \( r =_e s \), then \( t\{x/r\} =_e u\{x/s\} \).

• \( t^o =_e u^o \).

**Proof.** Suppose \( t =_e u \) holds in \( n \) steps. Both properties can be simultaneously proved by induction on the lexicographic pair \( (n,t) \). \( \Box \)

**Corollary 8.12** (Z-Property). Let \( t, u \) be metaterms. If \( t \rightarrow^{*_{\text{lex}}} u \), then \( u \rightarrow^{*_{\text{lex}}} t^o \rightarrow^{*_{\text{lex}}} u^o \).

**Proof.** Let \( t =_e r \rightarrow^{*_{\text{ex}}} s =_e u \). By Lemma 8.11 \( r \rightarrow^{*_{\text{lex}}} s \rightarrow^{*_{\text{lex}}} r^o \) and by Lemma 8.11 \( t^o =_e r^o \) and \( s^o =_e u^o \). We thus conclude \( t \rightarrow^{*_{\text{lex}}} u^o \rightarrow^{*_{\text{lex}}} t^o \). \( \Box \)

**Corollary 8.13** (Confluence). The reduction relation \( \rightarrow^{*_{\text{lex}}} \) is confluent on metaterms.

**Proof.** Corollary 8.12 guarantees the Z-property. We conclude by Theorem A.1 in the Appendix A. \( \Box \)

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 and the PSN theorem are really modular with respect to other proofs in the literature [LLD+04, Bom01b], 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 [KRR09, AG09]. On the other hand, full composition alone is not sufficient to achieve the SN proof, otherwise the λσ-calculus [ACCL91], which is known to not being strongly normalising [Mel95], could be treated. Indeed, our strategy is not perpetual for λσ: Melliès’ counter-example is based on an infinite λσ-reduction sequence starting from a simply typed term which is not reached by our perpetual strategy. In other words, is incomplete for λσ. The definition of a perpetual strategy for λσ remains open.

We believe that a de Bruijn or nominal version of λex could be useful in real implementations. In the first case, this could be achieved by using for example λσ⇑ technology (so that equation 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 [Mun01]. The main contribution of λex with respect to the formalisms previously mentioned would be the safe notion of full composition.

References


Appendix A. Abstract Reduction Results

Theorem A.1 (Z implies Confluence). If $\rightarrow_R$ has the Z-property, then $\rightarrow_R$ is confluent.

Proof. We give a proof following the picture appearing in [vO] which proceeds in many steps. Suppose that $\omega$ is some map satisfying the Z-property for $\rightarrow_R$.

1. Define $a^* := a$ if $a$ is in $R$-normal form, $a^* := a^\circ$ otherwise.
2. Prove that $\omega$ also satisfies the Z-property for $\rightarrow_R$.
   Proof. If $a \rightarrow_R b$, then $b \rightarrow_R a^* \rightarrow_R b^\circ$ by the hypothesis and $a^* = a^\circ$ by Point (1) so that $b \rightarrow_R a^*$. If $b$ is an $R$-normal form, then $b^* = b = a^\circ = a^*$ so that $a^* \rightarrow_R b^*$. If $b$ is not an $R$-normal form, then $b^* = b^\circ$ so that also $a^* = a^\circ \rightarrow_R b^\circ = b^*$.

3. Prove that $a \rightarrow_R a^*$.
   Proof. If $a$ is an $R$-normal form, then $a^* = a$ so we are done. Otherwise, there is $b$ such that $a \rightarrow_R b$, so that Point (2) gives $b \rightarrow_R a^*$ and thus $a \rightarrow_R a^*$.

4. Prove that $a \rightarrow_R b$ implies $a^* \rightarrow_R b^*$.
   Proof. By induction on the number $n$ of steps from $a$ to $b$. If $n = 0$, then $a = b$ and $a^* = b^*$. If $n > 0$, then $a \rightarrow_R c \rightarrow_R b$, where $c \rightarrow_R b$ holds in $n - 1$ steps. Point (2) and the i.h. give $a^* \rightarrow_R c^* \rightarrow_R b^*$.

5. Conclude confluence of $\rightarrow_R$.
   Proof. Let $t \rightarrow_R t_1$ and $t \rightarrow_R t_2$. We want to show that there is $t_3$ such that $t_1 \rightarrow_R t_3$ and $t_2 \rightarrow_R t_3$. 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 $t \rightarrow_R u \rightarrow_R t_2$, with $n - 1$ steps from $u$ to $t_2$. By Point (2) $u \rightarrow_R t^*$ and by Point (1) $t^* \rightarrow_R t_2^*$ so that $u \rightarrow_R t_2^*$. By Point (3) $t_1 \rightarrow_R t_1^*$. Now, $u \rightarrow_R t_1^*$ and $u \rightarrow_R t_2$ holds in $n - 1$ steps so we close the diagram by the i.h. 

Theorem A.2 (Modular Strong Normalisation). Let $A_1$ and $A_2$ be two reduction relations on $S$ and let $\mathcal{A}$ be a reduction relation on $S$. Let $R \subseteq S \times S$. Suppose

P1: For every $u, v, U$ ($u \mathcal{R} U \cup u A_1 v$ imply $\exists V \ s.t. \ V R V$ and $U A^+ V$).

P1: For every $u, v, U$ ($u \mathcal{R} U \cup u A_2 v$ imply $\exists V \ s.t. \ V R V$ and $U A^+ V$).

P1: The relation $A_1$ is well-founded.

Then, $t \mathcal{R} T \in SN_A$ imply $t \in SN_{A_1 \cup A_2}$.

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 $t \notin SN_{A_1 \cup A_2}$. Then, there is an infinite $A_1 \cup A_2$-reduction sequence starting at $t$, and since $A_1$ is a well-founded relation by P3, this reduction sequence has necessarily the form

$$t \rightarrow^{\omega}_A t_1 \rightarrow^+_A t_2 \rightarrow^{\omega}_A t_3 \rightarrow^+_A \ldots$$

and can be projected by P1 and P2 into an infinite $A$-reduction sequence as follows:

$$t \rightarrow^+_A t_1 \rightarrow^+_A t_2 \rightarrow^+_A t_3 \rightarrow^+_A \ldots$$

We thus get a contradiction with the fact the $T \in SN_A$. 
