# Milner's Lambda-Calculus with Partial Substitutions

Delia Kesner[1] and Shane Ó Conchúir[2]

[1] PPS, CNRS, and Université Paris 7, France
[2] Trinity College Dublin, Ireland

**Abstract.** We study Milner's lambda-calculus with partial substitutions. Particularly, we show confluence on terms and metaterms, preservation of $\beta$-strong normalisation and characterisation of strongly normalisable terms via an intersection typing discipline. The results on terms transfer to Milner's bigraphical model of the calculus. We relate Milner's calculus to calculi with definitions, to explicit substitutions, and to MELL Proof-Nets.

## 1 Introduction

The $\lambda_{sub}$-calculus was introduced by Milner as a means to modelling the $\lambda$-calculus in bigraphs [25]. However, the $\lambda_{sub}$-calculus is interesting apart from the model; it enjoys confluence on terms, step-by-step simulation of $\beta$-reduction [28], and preservation of $\beta$-strong normalisation (PSN) *i.e.* every $\lambda$-calculus term which is $\beta$-strongly normalising is also $\lambda_{sub}$-strongly normalising [27].

In this paper we study many remaining open questions about $\lambda_{sub}$. The first of them concerns *confluence on metaterms* which are terms containing *metavariables* usually used to denote *incomplete* programs and/or proofs in higher-order frameworks [15]. To obtain a confluent reduction relation on metaterms we need to extend the existing notion of reduction on terms. We develop a proof of confluence for this extended new relation by using Tait and Martin-Löf's technique. This proof includes a formal argument to show that the calculus of substitution itself is terminating.

Our main contribution lies in studying the connections between $\lambda_{sub}$ and other formalisms. We start by considering calculi with definitions, namely, the partial $\lambda$-calculus [13, 26], which we call $\lambda_{\beta_p}$, and the $\lambda$-calculus with definitions [30], which we call $\lambda_{def}$. We distinguish arbitrary terms of the calculi with definitions, which we call $\lambda\mathtt{s}$-terms, from terms without definition, which are ordinary $\lambda$-terms. We show that the sets of strongly-normalising $\lambda$-terms in $\lambda_{sub}$ and $\lambda_{\beta_p}$ are the same. Similarly, we show that the sets of strongly-normalising $\lambda\mathtt{s}$-terms in $\lambda_{sub}$ and $\lambda_{def}$ are equal. Thus, we demonstrate that partial substitutions and definitions are similar notions.

We also relate $\lambda_{sub}$-strongly normalising terms to typed terms. We start by introducing an intersection type discipline for $\lambda\mathtt{s}$-terms and then give a simple (and constructive) argument to prove $\lambda_{\beta_p}$-strong normalisation for typed $\lambda$-terms. This argument turns out to be sufficient to conclude $\lambda_{sub}$-strong normalisation for *intersection* typed $\lambda\mathtt{s}$-terms. By proving the converse *i.e.* $\lambda_{sub}$-strongly normalising $\lambda\mathtt{s}$-terms can be typed in the intersection type discipline, we also provide a characterisation of $\lambda_{sub}$-strongly normalising terms.

The relation between typable and $\lambda_{sub}$-strongly normalising $\lambda$s-terms also gives an alternative proof of PSN for $\lambda_{sub}$, which is self-contained, and which simplifies previous work [27] considerably: the existing proof is quite involved, using a translation of $\lambda_{sub}$ into a rather complex calculus, a proof of PSN for the complex calculus, and a proof of simulation.

Another contribution of the paper is the study of the relation between partial substitutions and explicit substitutions. More precisely, we define a translation from $\lambda_{sub}$ to a calculus with explicit substitutions called $\lambda$es [18]. This translation preserves reduction and has at least two important consequences. On one hand, we obtain a simple proof of $\lambda_{sub}$-strong normalisation for *simply typed* $\lambda$s-terms. A second consequence is that the existing simulation of the simply typed $\lambda$es-calculus into MELL Proof-Nets [18] also gives a natural interpretation for the simply typed $\lambda_{sub}$-calculus by composition. As a corollary, $\lambda_{sub}$-strong normalisation for simply typed $\lambda$s-terms can also be inferred from strong normalisation of MELL Proof-Nets.

Finally, we transfer our confluence and strong normalisation proofs on $\lambda$s-terms without metavariables in $\lambda_{sub}$ to Milner's model using an existing result.

The paper is organised as follows. Section 2 introduces the $\lambda_{sub}$-calculus and confluence on metaterms is proven using Tait and Martin-Löf's technique. In Section 3 we relate $\lambda_{sub}$ to the calculi with definitions, $\lambda_{\beta_p}$ and $\lambda_{def}$. Section 4 presents a neat characterisation of $\lambda_{sub}$-strongly normalising terms using intersection type systems as well as the PSN property for untyped $\lambda$s-terms of $\lambda_{sub}$. In Section 5, we present the translation from $\lambda_{sub}$ to $\lambda$es and prove that reduction in the former is simulated by non-empty reduction sequences in the latter. We conclude $\lambda_{sub}$-strong normalisation for simply typed $\lambda$s-terms from strong $\lambda$es-normalisation for simply typed $\lambda$s-terms. Last but not least, we discuss a relation between $\lambda_{sub}$ and MELL Proof-Nets and transfer results to the bigraphical setting in Section 6.

Due to lack of space the full proofs are contained in the related technical report [19].

## 2 The $\lambda_{sub}$-calculus

The $\lambda_{sub}$-calculus was introduced by Milner to present a model of the $\lambda$-calculus in local bigraphs. The calculus was inspired by $\lambda_\sigma$ [1] although it is a *named* calculus and has turned out to have stronger properties as we show in this paper. Terms of the $\lambda_{sub}$-calculus, called $\lambda$s-terms, are given by:

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

The piece of syntax $[x/t]$, which is not a term itself, is called an *explicit substitution*.

*Free* and *bound* variables are defined as usual by assuming the terms $\lambda x.t$ and $t[x/u]$ bind $x$ in $t$. We consider $\alpha$-conversion which is the congruence generated by renaming of bound variables. Thus for example $(\lambda y.x)[x/y] =_\alpha (\lambda z.x')[x'/y]$. We work with $\alpha$-equivalence classes so that two bound variables of the same term are assumed to be distinct, and no free and bound variable of the same term have the same name. Thus, $\alpha$-conversion avoids capture of variables. *Implicit Substitution* on $\lambda$s-terms can be defined modulo $\alpha$-conversion in such a way that capture of variables is avoided:

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

The set of $\lambda_{sub}$-contexts can be defined by the following grammar:

$$C ::= \Box \mid C\ t \mid t\ C \mid \lambda x.C \mid C[x/t] \mid t[x/C]$$

We use the notation $C[\![u]\!]_\phi$ to mean that the hole $\Box$ in the context $C$ has been replaced by the term $u$ *without capture of the variables in the set* $\phi$. Thus for example, if $C = \lambda z.\Box$, then $C[\![x]\!]_\phi$ with $x \in \phi$ means in particular that $z \neq x$.

Reduction rules of the $\lambda_{sub}$-calculus are given in the following table.

$$
\boxed{
\begin{array}{llll}
(\lambda x.t)\ u & \to_{\mathtt{B}} & t[x/u] & \\
t[x/u] & \to_{\mathtt{Gc}} & t & \text{if } x \notin \mathtt{fv}(t) \\
C[\![x]\!]_\phi[x/u] & \to_{\mathtt{R}} & C[\![u]\!]_\phi[x/u] & \text{if } \{x\} \cup \mathtt{fv}(u) \subseteq \phi
\end{array}
}
$$

As Milner describes, an explicit substitution $[x/u]$ acts *'at a distance'* on each free occurrence of $x$ in turn, rather than migrating a copy of itself towards each such occurrence *e.g.* the reduction step $(\lambda x.x\ (y\ y))[y/t] \to_{\mathtt{R}} (\lambda x.x\ (t\ y))[y/t]$ demonstrates a partial substitution.

The *rewriting system* generated by the reduction rules `R` and `Gc` is denoted by `sm`. We write `Bsm` for `B` $\cup$ `sm`. The *reduction relation* generated by the *reduction rules* `sm` (resp. `Bsm`) *modulo the equivalence relation* $\alpha$ is denoted by $\to_{sub}$ (resp. $\to_{\lambda_{sub}}$).

$$
\begin{array}{ll}
t \to_{sub} t' & \text{iff there are } s, s' \text{ s.t. } t =_\alpha s \to_{\mathtt{sm}} s' =_\alpha t' \\
t \to_{\lambda_{sub}} t' & \text{iff there are } s, s' \text{ s.t. } t =_\alpha s \to_{\mathtt{Bsm}} s' =_\alpha t'
\end{array}
$$

Thus, the reduction relation acts on $\alpha$-equivalence classes. We use the notation $\to^*_{\lambda_{sub}}$ (resp. $\to^+_{\lambda_{sub}}$) to denote the reflexive (resp. reflexive and transitive) closure of $\to_{\lambda_{sub}}$. As a consequence if $t \to^*_{\lambda_{sub}} t'$ in 0 steps, then $t =_\alpha t'$ and not $t = t'$.

Reduction enjoys the following properties. In particular, the second one, called *full composition*, guarantees that explicit substitution implements the implicit one. While this property seems reasonable/natural, it is worth noticing that many calculi with explicit substitutions do not enjoy it. Fortunately, full composition holds for $\lambda_{sub}$ [28].

**Lemma 1 (Preservation of Free Variables).** *If* $t \to_{\lambda_{sub}} t'$*, then* $\mathtt{fv}(t') \subseteq \mathtt{fv}(t)$.

**Lemma 2 (Full Composition (i)).** *Let* $t, u$ *be* $\lambda\mathtt{s}$*-terms. Then* $t[x/u] \to^+_{\lambda_{sub}} t\{x/u\}$.

### 2.1 Confluence

We briefly discuss confluence on metaterms, usually used to denote *incomplete* programs/proofs in higher-order frameworks [15]. The set of $\lambda\mathtt{s}$*-metaterms* is obtained by adding *annotated metavariables* of the form $\mathbb{X}_\Delta$ (where $\Delta$ is a set of variables) to the grammar generating $\lambda\mathtt{s}$-terms. The notion of free variables is extended to metaterms by

$\mathtt{fv}(\mathbb{X}_{\Delta}) = \Delta$. As a consequence, $\alpha$-conversion can also be defined on metaterms and thus for example $\lambda x.\mathbb{X}_{x,y} =_{\alpha} \lambda z.\mathbb{X}_{z,y}$.

We extend the standard notion of *implicit substitution* to metaterms as follows:

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

It is worth noticing that Milner's original presentation did not consider metaterms as the bigraphical system did not model them however all properties we prove here involving metaterms hold also for terms.

We add to the reduction system for $\lambda\mathtt{s}$-terms one equation and one reduction rule.

$$\boxed{\begin{array}{lll} t[x/u][y/v] & =_{\mathtt{C}} \quad t[y/v][x/u] & \text{if } y \notin \mathtt{fv}(u) \;\&\; x \notin \mathtt{fv}(v) \\ \\ C[\![\mathbb{X}_{\Delta}]\!]_{\phi}[x/u] & \to_{\mathtt{R}_{\mathbb{X}}} C[\![\mathbb{X}_{\Delta}[x/u]]\!]_{\phi}[x/u] & \text{if } x \in \Delta \;\&\; x \cup \mathtt{fv}(u) \subseteq \phi \\ & & \quad \;\&\; C \neq \Box[y_1/v_1]\ldots[y_n/v_n]\ (n \geq 0) \end{array}}$$

Remark in particular that $\mathtt{R}_{\mathbb{X}}$ cannot be applied if the context is empty. Remark also that the equation $\mathtt{C}$ can always be postponed w.r.t. $\lambda_{sub}$-reduction if only terms (and not metaterms) are considered.

Throughout this section, we include $\mathtt{R}_{\mathbb{X}}$ in the reduction relation $\mathtt{sm}$ as well as $\mathtt{C}$ in the equivalence relation, called now $\mathtt{E}_{\mathtt{s}}$. Full composition still holds for $\lambda\mathtt{s}$-metaterms:

**Lemma 3 (Full Composition (ii)).** *Let $t, u$ be $\lambda\mathtt{s}$-metaterms. Then $t[x/u] \to^{+}_{\lambda_{sub}} t\{x/u\}$.*

While confluence on terms always holds for calculi with explicit substitutions, confluence on metaterms is often based on some form of interaction of substitutions such as that in $\lambda\sigma$ [1] or $\lambda_{ws}$ [12]. To illustrate this requirement, let us consider the typical diverging example adapted to $\lambda_{sub}$-reduction:

$$t\{y/v\}[x/u\{y/v\}] \;{}^{*}_{\lambda_{sub}}\!\leftarrow ((\lambda x.t)\ u)[y/v] \to_{\mathtt{B}} t[x/u][y/v]$$

This diagram can be closed using full composition with the sequence $t[x/u][y/v] \to^{+}_{\lambda_{sub}} t[x/u]\{y/v\} = t\{y/v\}[x/u\{y/v\}]$.

However, while de Bruijn notation for $\lambda$-terms allows a canonical representation of bound variables given by a certain order on their natural numbers, calculi with named variables suffer from the following (also typical) diverging example:

$$\mathbb{X}_{x,y}[y/v][x/z] \;{}^{*}_{\lambda_{sub}}\!\leftarrow ((\lambda x.\mathbb{X}_{x,y})\ z)[y/v] \to_{\mathtt{B}} \mathbb{X}_{x,y}[x/z][y/v]$$

The metaterms $\mathbb{X}_{x,y}[y/v][x/z]$ and $\mathbb{X}_{x,y}[x/z][y/v]$ are equal modulo permutation of *independent substitutions*, thus justifying the introduction of the equation $\mathtt{C}$ in the definition of the calculus for metaterms.

One possible technical tool to show confluence for $\lambda\mathtt{s}$-metaterms is the use of another confluent calculus well-related to $\lambda_{sub}$. We prefer to give a self-contained argument, and so adapt a proof based on Tait and Martin-Löf's technique: define a simultaneous reduction relation denoted $\Rrightarrow_{\lambda_{sub}}$; prove that $\lambda_{sub}$ can be projected to $\Rrightarrow_{\lambda_{sub}}$ on $sub$-normal forms; show that $\Rrightarrow^{*}_{\lambda_{sub}}$ has the diamond property; and finally conclude. Since this technique is considered nowadays as standard folklore, we just state here the main reduction properties used in our particular case and refer the reader to [19].

**Lemma 4.** *The $sub$-normal forms of metaterms exist and are unique modulo $\mathtt{E_s}$.*

*Proof.* Details can be found in [19]. The system $sub$ can be shown to be terminating by associating to each $\lambda\mathtt{s}$-metaterm a measure which does not change by $\mathtt{E_s}$ but strictly decreases by $\rightarrow_{\mathtt{sm}}$. Thus, $sub$-normal forms exist. The $sub$-critical pairs are joinable, so $sub$ is locally confluent and locally coherent. Therefore [17], $sub$ is confluent and hence $sub$-normal forms are unique modulo $\mathtt{E_s}$-equivalence.

Remark that metaterms in $sub$-normal form have all explicit substitutions directly above metavariables. Thus in particular terms without metavariables in $sub$-normal form have no explicit substitutions at all.

**Definition 5.** *The relation $\Rrightarrow$ on metaterms in $sub$-normal form is given by:*

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

The relation $\Rrightarrow_{sub}$ is defined by $t \Rrightarrow_{sub} t'$ iff $\exists s, s'$ s.t. $t =_{\mathtt{E_s}} s \Rrightarrow s' =_{\mathtt{E_s}} t'$.

**Lemma 6.**

1. *The reflexive and transitive closures of $\Rrightarrow_{\lambda_{sub}}$ and $\rightarrow_{\lambda_{sub}}$ are the same relation.*
2. *If $s \rightarrow_{\lambda_{sub}} s'$ then $sub(s) \Rrightarrow_{\lambda_{sub}} sub(s')$.*
3. *The reduction relation $\Rrightarrow_{\lambda_{sub}}$ enjoys the diamond property.*

**Corollary 7.** *The $\lambda_{sub}$-reduction relation is confluent on terms and metaterms.*

*Proof.* Suppose $t \rightarrow^*_{\lambda_{sub}} t_i$ for $i = 1, 2$. Lemma 6:2 gives $sub(t) \Rrightarrow^*_{\lambda_{sub}} sub(t_i)$. Since the diamond property implies confluence [5], then Lemma 6:3 implies confluence of $\Rrightarrow_{\lambda_{sub}}$; therefore, there is a metaterm $s$ s.t. $sub(t_i) \Rrightarrow^*_{\lambda_{sub}} s$. We can then close the diagram by $t_i \rightarrow^*_{sub} sub(t_i) \rightarrow^*_{\lambda_{sub}} s$ using Lemma 6:1.

## 3 Relating Partial Substitutions to Definitions

Partial substitution can be related to calculi with definitions. A definition can be understood as an abbreviation given by a name for a larger term which can be used several times in a program or a proof. A definition mechanism is essential for practical use; current implementations of proof assistants provide such a facility.

We consider two calculi, the first one, which we call $\lambda_{\beta_p}$, appears in [13] and uses a notion of partial substitution on $\lambda$-terms, while the second one, which we call $\lambda_{def}$, uses partial substitutions on $\lambda\mathtt{s}$-terms to model definitions and combines standard $\beta$-reduction with the rules of the substitution calculus $sub$. The general result of this section is that normalisation in $\lambda_{\beta_p}$ and $\lambda_{sub}$ are equivalent on $\lambda$-terms and normalisation in $\lambda_{def}$ and $\lambda_{sub}$ are equivalent on $\lambda\mathtt{s}$-terms. More precisely, for every $\lambda$-term $t$, $t \in \mathcal{SN}_{\lambda_{\beta_p}}$ if and only if $t \in \mathcal{SN}_{\lambda_{sub}}$ and for every $\lambda\mathtt{s}$-term $t$, $t \in \mathcal{SN}_{\lambda_{def}}$ if and only if $t \in \mathcal{SN}_{\lambda_{sub}}$. Thus, the $\lambda_{sub}$-calculus can be understood as a concise and simple language implementing partial and ordinary substitution, both in implicit and explicit style at the same time.

### 3.1 The partial $\lambda$-calculus $\lambda_{\beta_p}$

Terms of the partial calculus $\lambda_{\beta_p}$ are $\lambda$-terms. The operational semantics of $\lambda_{\beta_p}$ is given by the following rules:

$$\boxed{\begin{array}{lll} (\lambda x.C[\![x]\!]_\phi)\ u \to_{\beta_p} (\lambda x.C[\![u]\!]_\phi)\ u & \text{if } \{x\} \cup \mathtt{fv}(u) \subseteq \phi \\ (\lambda x.t)\ u \to_{\mathtt{BGc}} t & \text{if } x \notin \mathtt{fv}(t) \end{array}}$$

Consider the following translation from $\lambda$-terms to $\lambda\mathtt{s}$-terms:

$$\begin{array}{l} \mathtt{U}(x) := x \\ \mathtt{U}(\lambda x.t) := \lambda x.\mathtt{U}(t) \end{array} \qquad \mathtt{U}(t\ u) := \begin{cases} \mathtt{U}(t)\ \mathtt{U}(u) & \text{if } t \text{ is not a } \lambda\text{-abstraction} \\ \mathtt{U}(v)[x/\mathtt{U}(u)] & \text{if } t = \lambda x.v \end{cases}$$

**Lemma 8.** *If $t \to_{\lambda_{\beta_p}} t'$, then $\mathtt{U}(t) \to^+_{\lambda_{sub}} \mathtt{U}(t')$. Proof.* By induction on $\to_{\lambda_{\beta_p}}$.

**Corollary 9.** *Let $t$ be a $\lambda$-term. If $t \in \mathcal{SN}_{\lambda_{sub}}$, then $t \in \mathcal{SN}_{\lambda_{\beta_p}}$.*

*Proof.* Let $t \in \mathcal{SN}_{\lambda_{sub}}$ and suppose $t \notin \mathcal{SN}_{\lambda_{\beta_p}}$. Then, from an infinite $\lambda_{\beta_p}$-reduction sequence starting at $t$ we can construct, by Lemma 8, an infinite $\lambda_{sub}$-reduction sequence starting at $\mathtt{U}(t)$. Since $t \to^*_{\lambda_{sub}} \mathtt{U}(t)$, then $t \notin \mathcal{SN}_{\lambda_{sub}}$, which leads to a contradiction. We thus conclude $t \in \mathcal{SN}_{\lambda_{\beta_p}}$.

The converse reasoning also works. Define a translation from $\lambda\mathtt{s}$-terms to $\lambda$-terms:

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

Remark that $\mathtt{V}(t)\{x/\mathtt{V}(u)\} = \mathtt{V}(t\{x/u\})$. Lemma 10 follows by induction on $\to_{\lambda_{sub}}$.

**Lemma 10.** *Let $t \to_{\lambda_{sub}} t'$. If $t \to_{\mathtt{B}} t'$, then $\mathtt{V}(t) = \mathtt{V}(t')$. Also, if $t \to_{sub} t'$, then $\mathtt{V}(t) \to^+_{\lambda_{\beta_p}} \mathtt{V}(t')$.*

**Corollary 11.** *Let $t$ be a $\lambda$-term. If $t \in \mathcal{SN}_{\lambda_{\beta_p}}$, then $t \in \mathcal{SN}_{\lambda_{sub}}$.*

*Proof.* Follows from Lemma 10 similarly to Corollary 9, using the fact that infinite $\to_{\lambda_{sub}}$ sequences must contain an infinite number of $\to_{sub}$ steps.

### 3.2 The $\lambda$-calculus with definitions $\lambda_{def}$

The syntax of the $\lambda$-calculus with definitions $\lambda_{def}$ [30], is isomorphic to that of $\lambda_{sub}$, where the use of a definition $x := u$ in a term $v$, denoted $\mathtt{let}\ x := u\ \mathtt{in}\ v$, can be thought as the term $v[x/u]$ in $\lambda_{sub}$. The original presentation [30] of the operational semantics of $\lambda_{def}$ is given by a reduction system which is not a (higher-order) term rewriting system. This is due to the fact that given a definition $x := u$, the term $x$ can be reduced to the term $u$, so that reduction creates new free variables since $\mathtt{fv}(u)$ does not necessarily belong to $\{x\}$. Here, we present $\lambda_{def}$ by a set of reduction rules which preserve free variables of terms. Moreover, we consider a more general reduction system where any $\beta$-redex can be either $\beta$-reduced or transformed to a definition, while the calculus appearing in [30] does not allow dynamic creation of definitions.

$$\boxed{\begin{array}{llll} (\lambda x.t)\ u \to_\beta t\{x/u\} & t[x/u] & \to_{\mathtt{Gc}} t & \text{if } x \notin \mathtt{fv}(t) \\ (\lambda x.t)\ u \to_{\mathtt{B}} t[x/u] & C[\![x]\!]_\phi[x/u] & \to_{\mathtt{R}} C[\![u]\!]_\phi[x/u] & \text{if } \{x\} \cup \mathtt{fv}(u) \subseteq \phi \end{array}}$$

**Lemma 12.** *If $t \to_{\lambda_{def}} t'$, then $t \to^{+}_{\lambda_{sub}} t'$. Also, if $t \to_{\lambda_{sub}} t'$, then $t \to^{+}_{\lambda_{def}} t'$.*

*Proof.* The first point can be shown by induction on $\to_{\lambda_{def}}$ using the fact that any $\beta$ step can be simulated by `B` followed by several `R` steps and one `Gc` step. The second point is straightforward.

We can then conclude that normalisation for $\lambda_{def}$ and $\lambda_{sub}$ is equivalent.

**Corollary 13.** *Let $t$ be a $\lambda$s-term. Then $t \in \mathcal{SN}_{\lambda_{sub}}$ if and only if $t \in \mathcal{SN}_{\lambda_{def}}$.*

## 4 Normalisation Properties

Intersection type disciplines [9, 10] are more flexible than simple type systems in the sense that not only are typed terms strongly normalising, but the converse also holds, thus giving a characterisation of the set of strongly normalising terms. Intersection types for calculi with explicit substitutions not enjoying full composition have been studied [22, 20]. Here, we apply this technique to $\lambda_{sub}$, and obtain a characterisation of the set of strongly-normalising terms.

*Types* are built over a countable set of atomic symbols (base types) and the type constructors $\to$ (functional types) and $\cap$ (intersection types). An *environment* is a finite set of pairs of the form $x : A$. Two environments $\Gamma$ and $\Delta$ are said to be *compatible* iff for all $x : A \in \Gamma$ and $y : B \in \Delta$, $x = y$ implies $A = B$. We denote the *union of compatible contexts* by $\Gamma \uplus \Delta$. Thus for example $(x : A, y : B) \uplus (x : A, z : C) = (x : A, y : B, z : C)$.

**Definition 14.** *The relation $\ll$ on types is defined by the following axioms and rules*

1. $A \ll A$
2. $A \cap B \ll A$
3. $A \cap B \ll B$
4. $A \ll B \;\&\; B \ll C$ *implies* $A \ll C$
5. $A \ll B \;\&\; A \ll C$ *implies* $A \ll B \cap C$

We use $\underline{n}$ for $\{1 \dots n\}$ and $\cap_n A_i$ for $A_1 \cap \dots \cap A_n$. The following property can be shown by induction on the definition of $\ll$.

**Lemma 15.** *Let $\cap_n A_i \ll \cap_m B_j$, where none of the $A_i$ and $B_j$ is an intersection type. Then for each $B_j$ there is $A_i$ s.t. $B_j = A_i$.*

*Typing judgements* have the form $\Gamma \vdash t : A$ where $t$ is a term, $A$ is a type and $\Gamma$ is an environment. *Derivations* of typing judgements in a certain type discipline system are obtained by application of the typing rules of the system. We consider several systems.

The *additive simply type* system for $\lambda$-terms (resp. for $\lambda$s-terms), written $\mathtt{add}_{\lambda}$ (resp. $\mathtt{add}_{\lambda\mathtt{s}}$), is given by the following rules $\mathtt{ax}^{+}$, $\mathtt{app}^{+}$, and $\mathtt{abs}^{+}$ (resp. $\mathtt{ax}^{+}$, $\mathtt{app}^{+}$, $\mathtt{abs}^{+}$, and $\mathtt{subs}^{+}$).

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

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

The *additive intersection type* system for $\lambda$-terms (resp. for $\lambda$s-terms), written $\mathtt{add}^i_\lambda$ (resp. $\mathtt{add}^i_{\lambda\mathtt{s}}$), is obtained by adding the following rules to $\mathtt{add}_\lambda$ (resp. $\mathtt{add}_{\lambda\mathtt{s}}$).

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

The *multiplicative simple type* system for $\lambda$-terms (resp. for $\lambda$s-terms), written $\mathtt{mul}_\lambda$ (resp. $\mathtt{mul}_{\lambda\mathtt{s}}$), is given by the following rules $\mathtt{ax}^*$, $\mathtt{app}^*$, and $\mathtt{abs}^*$ (resp. $\mathtt{ax}^*$, $\mathtt{app}^*$, $\mathtt{abs}^*$, and $\mathtt{subs}^*$). The *multiplicative intersection type* system for $\lambda$-terms (resp. for $\lambda$s-terms), written $\mathtt{mul}^i_\lambda$ (resp. $\mathtt{mul}^i_{\lambda\mathtt{s}}$), is obtained by adding the rules $\cap$ I and $\cap$ E.

$$\boxed{\begin{array}{ll} \dfrac{}{x : A \vdash x : A}(\mathtt{ax}^*) & \dfrac{\Gamma \vdash t : A \to B \qquad \Delta \vdash u : A}{\Gamma \uplus \Delta \vdash (t\ u) : B}(\mathtt{app}^*) \\[2ex] \dfrac{\Gamma \vdash t : B}{\Gamma \setminus \{x : A\} \vdash \lambda x.t : A \to B}(\mathtt{abs}^*) & \dfrac{\Gamma \vdash u : B \qquad \Delta \vdash t : A}{\Gamma \uplus (\Delta \setminus \{x : B\}) \vdash t[x/u] : A}(\mathtt{subs}^*) \end{array}}$$

A term $t$ is said to be *typable* in system $\mathcal{T}$, written $\Gamma \vdash_{\mathcal{T}} t : A$ iff there is $\Gamma$ and $A$ s.t. the judgement $\Gamma \vdash t : A$ is derivable from the set of typing rules of system $\mathcal{T}$. Remark that for any $\lambda$-term $t$ we have $\Gamma \vdash_{\mathtt{add}^i_\lambda} t : A$ iff $\Gamma \vdash_{\mathtt{add}^i_{\lambda\mathtt{s}}} t : A$ and $\Gamma \vdash_{\mathtt{mul}^i_\lambda} t : A$ iff $\Gamma \vdash_{\mathtt{mul}^i_{\lambda\mathtt{s}}} t : A$.

We need generation lemmas for each system.

**Lemma 16 (Multiplicative Generation Lemma).**

1. $\Gamma \vdash x : A$ *iff* $\Gamma = x : B$ *and* $B \ll A$.
2. $\Gamma \vdash t\ u : A$ *iff* $\Gamma = \Gamma_1 \uplus \Gamma_2$, *where* $\Gamma_1 = \mathtt{fv}(t)$ *and* $\Gamma_2 = \mathtt{fv}(u)$ *and there exist* $A_i, B_i, i \in \underline{n}$ *s.t.* $\cap_n A_i \ll A$ *and* $\forall i \in \underline{n}$, $\Gamma_1 \vdash t : B_i \to A_i$ *and* $\Gamma_2 \vdash u : B_i$.
3. $\Gamma \vdash t[x/u] : A$ *iff* $\Gamma = \Gamma_1 \uplus \Gamma_2$, *where* $\Gamma_1 = \mathtt{fv}(t) \setminus \{x\}$ *and* $\Gamma_2 = \mathtt{fv}(u)$ *and there exist* $A_i, B_i, i \in \underline{n}$ *s.t.* $\cap_n A_i \ll A$ *and* $\forall i \in \underline{n}$, $\Gamma_2 \vdash u : B_i$ *and either* $x \notin \mathtt{fv}(t)$ & $\Gamma_1 \vdash t : A_i$ *or* $x \in \mathtt{fv}(t)$ & $\Gamma_1, x : B_i \vdash t : A_i$.
4. $\Gamma \vdash \lambda x.t : A$ *iff* $\Gamma = \mathtt{fv}(\lambda x.t)$ *and there exist* $A_i, B_i, i \in \underline{n}$ *s.t.* $\cap_n (A_i \to B_i) \ll A$ *and l* $\forall i \in \underline{n}$, *either* $x \notin \mathtt{fv}(t)$ & $\Gamma \vdash t : B_i$ *or* $x \in \mathtt{fv}(t)$ & $\Gamma, x : A_i \vdash t : B_i$.
5. $\Gamma \vdash \lambda x.t : B \to C$ *iff* $\Gamma = \mathtt{fv}(\lambda x.t)$ *and* $\Gamma, x : B \vdash t : C$ *or* $\Gamma \vdash t : C$.

*Proof.* The right to left implications follow from the typing rules in the multiplicative systems and two simple lemmas full detailed in [19].

The left to right implication of points 1-4 is by induction on the typing derivation of the left part. The left to right implication of point 5 follows from point 4 and Lemma 15.

**Lemma 17 (Additive Generation Lemma).**

1. $\Gamma \vdash x : A$ *iff there is* $x : B \in \Gamma$ *and* $B \ll A$.
2. $\Gamma \vdash t\ u : A$ *iff there exist* $A_i, B_i, i \in \underline{n}$ *s.t.* $\cap_n A_i \ll A$ *and* $\Gamma \vdash t : B_i \to A_i$ *and* $\Gamma \vdash u : B_i$.
3. $\Gamma \vdash t[x/u] : A$ *iff there exist* $A_i, B_i, i \in \underline{n}$ *s.t.* $\cap_n A_i \ll A$ *and* $\forall i \in \underline{n}\ \Gamma \vdash u : B_i$ *and* $\Gamma, x : B_i \vdash t : A_i$.

4. $\Gamma \vdash \lambda x.t : A$ *iff there exist* $A_i, B_i, i \in \underline{n}$ *s.t.* $\cap_n(A_i \to B_i) \ll A$ *and* $\forall i \in \underline{n}$ $\Gamma, x : A_i \vdash t : B_i$.
5. $\Gamma \vdash \lambda x.t : B \to C$ *iff* $\Gamma, x : B \vdash t : C$.

*Proof.* Similar to the proof of Lemma 16.

In order to prove our main result we need a correspondence between the systems.

**Lemma 18.** *Let $t$ be a $\lambda\mathtt{s}$-term. Then $\Gamma \vdash_{\mathtt{add}^i_{\lambda\mathtt{s}}} t : A$ iff $\Gamma \cap \mathtt{fv}(t) \vdash_{\mathtt{mul}^i_{\lambda\mathtt{s}}} t : A$. Moreover, if $t$ is a $\lambda$-term, then $\Gamma \vdash_{\mathtt{add}^i_{\lambda}} t : A$ iff $\Gamma \cap \mathtt{fv}(t) \vdash_{\mathtt{mul}^i_{\lambda}} t : A$.*

*Proof.* The right to left implication is by induction on $t$ using both generation lemmas and a Weakening Lemma (see [19]). The left to right implication is by induction on $t$ using the generation lemmas.

Thus, from now on, *typed $\lambda$-term* means typable in $\mathtt{add}^i_{\lambda}$ and $\mathtt{mul}^i_{\lambda}$, and *typed $\lambda\mathtt{s}$-term* means typable in $\mathtt{add}^i_{\lambda\mathtt{s}}$ and $\mathtt{mul}^i_{\lambda\mathtt{s}}$.

### 4.1 Typed $\lambda\mathtt{s}$-terms are $\lambda_{sub}$-strongly normalising

The goal of this section is to show that typed $\lambda\mathtt{s}$-terms are $\lambda_{sub}$-strongly normalising. This result will be a consequence of strong normalisation of $\lambda$-terms in the partial calculus $\lambda_{\beta_p}$; a result which can be shown using a simple arithmetical proof [31, 11]. The proof is constructive as it only uses induction and intuitionistic reasoning.

**Lemma 19.** *Let $t, u$ be typed $\lambda$-terms. If $t, u \in \mathcal{SN}_{\lambda_{\beta_p}}$, then $t\{x/u\} \in \mathcal{SN}_{\lambda_{\beta_p}}$.*

*Proof.* By induction on $\langle \mathtt{type}(u), \eta_{\lambda_{\beta_p}}(t), \mathtt{size}(t)\rangle$. We treat the interesting cases.

- $t = xv\overline{v_n}$. The i.h. gives $V = v\{x/u\}$ and $V_i = v_i\{x/u\}$ in $\mathcal{SN}_{\lambda_{\beta_p}}$. To show $t\{x/u\} = uV\overline{V_n} \in \mathcal{SN}_{\lambda_{\beta_p}}$ it is sufficient to show that all its reducts are in $\mathcal{SN}_{\lambda_{\beta_p}}$. We reason by induction on $\eta_{\lambda_{\beta_p}}(u) + \eta_{\lambda_{\beta_p}}(V) + \Sigma_{i\in 1\ldots n}\, \eta_{\lambda_{\beta_p}}(V_i)$.
  - If the reduction takes place in $u$, $V$ or $V_i$, then the property holds by the i.h.
  - Suppose $u = \lambda y.U$ and $(\lambda y.U)\, V\, \overline{V_n} \to_{\mathtt{BGc}} U\, \overline{V_n}$. We write $U\, \overline{V_n}$ as $(z\, \overline{V_n})\{z/U\}$, where $z$ is a fresh variable. Since every $V_i \in \mathcal{SN}_{\lambda_{\beta_p}}$, then $z\, \overline{V_n} \in \mathcal{SN}_{\lambda_{\beta_p}}$. Also, $u \in \mathcal{SN}_{\lambda_{\beta_p}}$ implies $U \in \mathcal{SN}_{\lambda_{\beta_p}}$. Thus, $\mathtt{type}(U) < \mathtt{type}(u)$ implies $(z\, \overline{V_n})\{z/U\} \in \mathcal{SN}_{\lambda_{\beta_p}}$ by the i.h.
  - Suppose $u = \lambda y.C[\![y]\!]$ and $(\lambda y.C[\![y]\!])\, V\, \overline{V_n} \to_{\beta_p} (\lambda y.C[\![V]\!])\, V\, \overline{V_n}$. We write $\lambda y.C[\![V]\!]$ as $(\lambda y.C[\![z]\!])\{z/V\}$, where $z$ is a fresh variable. Since $u \in \mathcal{SN}_{\lambda_{\beta_p}}$, then $C[\![y]\!] \in \mathcal{SN}_{\lambda_{\beta_p}}$. The change of free occurrences of variables preserve normalisation so that $C[\![z]\!] \in \mathcal{SN}_{\lambda_{\beta_p}}$ and thus $\lambda y.C[\![z]\!] \in \mathcal{SN}_{\lambda_{\beta_p}}$. We also have $\mathtt{type}(V) = \mathtt{type}(v) < \mathtt{type}(u)$ so that we get $(\lambda y.C[\![z]\!])\{z/V\} \in \mathcal{SN}_{\lambda_{\beta_p}}$ by the i.h.
- $t = (\lambda y.s)v\overline{v_n}$. The i.h. gives $S = s\{x/u\}$ and $V = v\{x/u\}$ and $V_i = v_i\{x/u\}$ are in $\mathcal{SN}_{\lambda_{\beta_p}}$. These terms are also typed. To show $t\{x/u\} = (\lambda y.S)V\overline{V_n} \in \mathcal{SN}_{\lambda_{\beta_p}}$ it is sufficient to show that all its reducts are in $\mathcal{SN}_{\lambda_{\beta_p}}$. We reason by induction on $\eta_{\lambda_{\beta_p}}(S) + \eta_{\lambda_{\beta_p}}(V) + \Sigma_{i\in 1\ldots n}\, \eta_{\lambda_{\beta_p}}(V_i)$.

- If the reduction takes place in $S$, $V$ or $V_i$, then the property holds by the i.h.
- Suppose $(\lambda y.S)\ V\ \overline{V_n} \to_{\mathtt{BGc}} S\ \overline{V_n}$. We write $S\ \overline{V_n}$ as $(s\ \overline{v_n})\{x/u\}$. Since $(\lambda y.s)\ v\ \overline{v_i} \to_{\lambda_{\beta_p}} s\ \overline{v_n}$, then $\eta_{\lambda_{\beta_p}}(s\ \overline{v_n}) < \eta_{\lambda_{\beta_p}}((\lambda y.s)\ v\ \overline{v_n})$ and thus we conclude $S\ \overline{V_n} \in \mathcal{SN}_{\lambda_{\beta_p}}$ by the i.h.
- Suppose $u = \lambda y.C[\![y]\!]$ and $(\lambda y.C[\![y]\!])\ V\ \overline{V_n} \to_{\beta_p} (\lambda y.C[\![V]\!])\ V\ \overline{V_n}$. We write $\lambda y.C[\![V]\!]$ as $(\lambda y.C[\![v]\!])\{x/u\}$. Since $(\lambda y.C[\![y]\!])\ v\ \overline{v_n} \to_{\beta_p} (\lambda y.C[\![v]\!])\ v\ \overline{v_n}$, then $\eta_{\lambda_{\beta_p}}((\lambda y.C[\![v]\!])\ v\ \overline{v_n}) < \eta_{\lambda_{\beta_p}}((\lambda y.C[\![v]\!])\ v\ \overline{v_n})$ and thus we conclude $(\lambda y.C[\![V]\!])\ V\ \overline{V_n} \in \mathcal{SN}_{\lambda_{\beta_p}}$ by the i.h.

**Theorem 20 (SN for $\lambda_{\beta_p}$).** *If $t$ is a typed $\lambda$-term, then $t \in \mathcal{SN}_{\lambda_{\beta_p}}$.*

*Proof.* By induction on the structure of $t$. The cases $t = x$ and $t = \lambda x.u$ are straightforward. If $t = uv$, then write $t = (z\ v)\{z/u\}$. By the i.h. $u, v \in \mathcal{SN}_{\lambda_{\beta_p}}$ and thus Lemma 19 gives $t \in \mathcal{SN}_{\lambda_{\beta_p}}$.

**Corollary 21 (SN for $\lambda_{sub}$ (i)).** *If $t$ is a typed $\lambda\mathtt{s}$-term, then $t \in \mathcal{SN}_{\lambda_{sub}}$.*

*Proof.* Take $t$ typed in $\mathtt{add}^i_{\lambda\mathtt{s}}$. Then, $\mathtt{V}(t)$ is a $\lambda$-term. One shows by induction on $t$ that $\mathtt{V}(t)$ is typable in $\mathtt{add}^i_{\lambda}$ and that $\mathtt{V}(t) \to^+_{\mathtt{B}} t$. By Corollary 20 $\mathtt{V}(t) \in \mathcal{SN}_{\lambda_{\beta_p}}$ and by Corollary 11 $\mathtt{V}(t) \in \mathcal{SN}_{\lambda_{sub}}$ . Thus $t$ is also in $\mathcal{SN}_{\lambda_{sub}}$.

We now show that $\lambda_{sub}$-reduction preserves $\beta$-strong normalisation. This property, known as PSN, received a lot of attention (see for example [1, 6, 7]), starting from an unexpected result given by Melliès [23] who has shown that there are $\beta$-strongly normalisable terms in $\lambda$-calculus that are not strongly normalisable in calculi such as $\lambda\sigma$ [1]. Since then, many formalisms have been shown to enjoy PSN. In particular, $\lambda_{sub}$ enjoys PSN [27]. We reprove this property in a more simple way.

**Corollary 22 (PSN for $\lambda_{sub}$).** *If $t \in \mathcal{SN}_{\beta}$, then $t \in \mathcal{SN}_{\lambda_{sub}}$.*

*Proof.* If $t \in \mathcal{SN}_{\beta}$, then $t$ is typable in $\mathtt{add}^i_{\lambda}$ by [29], so that $t$ is also typable in $\mathtt{add}^i_{\lambda\mathtt{s}}$ and thus we conclude $t \in \mathcal{SN}_{\lambda_{sub}}$ by Corollary 21.

### 4.2 $\lambda_{sub}$-strongly normalising terms are typed $\lambda\mathtt{s}$-terms

We now complete the picture by showing that the intersection type discipline for $\lambda\mathtt{s}$-terms gives a characterisation of $\lambda_{sub}$-strongly normalising terms. To do this, we use the translation $\mathtt{V}()$ introduced in Section 3.1 to relate $\lambda\mathtt{s}$-terms to $\lambda$-terms.

**Lemma 23.** *Let $t$ be a $\lambda\mathtt{s}$-term. Then $\Gamma \vdash_{\mathtt{add}^i_{\lambda}} \mathtt{V}(t) : A$ iff $\Gamma \vdash_{\mathtt{add}^i_{\lambda\mathtt{s}}} t : A$.*

*Proof.* By induction on $t$ using the Generation Lemma 17.

**Lemma 24.** *If $\mathtt{V}(t) \to_{\beta} t'$, then $\exists\ u$ s.t. $t \to^+_{\lambda_{sub}} u$ and $t' = \mathtt{V}(u)$.*

*Proof.* By induction on the reduction step $\mathtt{V}(t) \to_{\beta} t'$ and Lemma 2.

**Theorem 25.** *If $t \in \mathcal{SN}_{\lambda_{sub}}$, then $t$ is a typed $\lambda\mathtt{s}$-term.*

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

*Proof.* Let $t \in \mathcal{SN}_{\lambda_{sub}}$. Suppose $\mathtt{V}(t) \notin \mathcal{SN}_{\beta}$. Then, there is an infinite $\beta$-reduction sequence starting at $\mathtt{V}(t)$, which can be projected, by Lemma 24, to an infinite $\lambda_{sub}$-reduction sequence starting at $t$. Thus $t \notin \mathcal{SN}_{\lambda_{sub}}$, which leads to a contradiction.

Therefore $\mathtt{V}(t) \in \mathcal{SN}_{\beta}$, so that $\mathtt{V}(t)$ is typable in $\mathtt{add}^i_{\lambda}$ by [29]. By Lemma 23 $t$ is typable in $\mathtt{add}^i_{\lambda\mathtt{s}}$. Lemma 18 also gives $t$ typable in $\mathtt{mul}^i_{\lambda\mathtt{s}}$.

**Corollary 26.** *For $\lambda$s-terms, $t$ is typable in $\mathtt{add}^i_{\lambda\mathtt{s}}$ iff $t$ is typable in $\mathtt{mul}^i_{\lambda\mathtt{s}}$ iff $t \in \mathcal{SN}_{\lambda_{sub}}$ iff $t \in \mathcal{SN}_{\lambda_{def}}$. Furthermore, for $\lambda$-terms, $t$ is typable in $\mathtt{add}^i_{\lambda}$ iff $t$ is typable in $\mathtt{mul}^i_{\lambda}$ iff $t \in \mathcal{SN}_{\lambda_{sub}}$ iff $t \in \mathcal{SN}_{\lambda_{\beta_p}}$ iff $t \in \mathcal{SN}_{\beta}$.*

## 5 Relating Partial to Explicit Substitutions

We now relate $\lambda_{sub}$ to a calculus based on explicit substitutions called $\lambda\mathtt{es}$, summarised below. We then give a translation from $\lambda_{sub}$ to $\lambda\mathtt{es}$ and we show that each $\lambda_{sub}$-reduction step can be simulated by a non-empty reduction sequence in $\lambda\mathtt{es}$. This translation will provide a second proof of $\lambda_{sub}$-strong normalisation for typed $\lambda\mathtt{s}$-terms.

Terms of the $\lambda\mathtt{es}$-calculus are $\lambda\mathtt{s}$-terms. Besides $\alpha$-conversion, we consider the equations and reduction rules in the figure below. Remark that working modulo $\alpha$-conversion allows us to assume implicitly some conditions to avoid capture of variables such as for example $x \neq y$ and $y \notin \mathtt{fv}(v)$ in the reduction rule Lamb.

We consider the equivalence relation $\mathtt{E_s}$ generated by $\alpha$ and C. The *rewriting system* containing all the reduction rules except B is denoted s. We write Bs for $\mathtt{B} \cup \mathtt{s}$. We note ALC, the *reduction relation* generated by the rules $\{\mathtt{App}_1, \mathtt{App}_2, \mathtt{App}_3, \mathtt{Lamb}, \mathtt{Comp}_1, \mathtt{Comp}_2\}$ *modulo the equivalence relation* $\mathtt{E_s}$. The *reduction relation* generated by s (resp. Bs) *modulo* $\mathtt{E_s}$ is denoted by $\rightarrow_{\mathtt{es}}$ (resp. $\rightarrow_{\lambda\mathtt{es}}$), where **e** means equational and **s** means substitution.

As expected, reduction preserves free variables. Remark that ALC may only propagate garbage substitutions through abstractions and not through applications or inside explicit substitutions. We now use ALC as a function on $\mathtt{E_s}$-equivalence classes.

**Lemma 27.** *The* ALC*-normal forms of terms exist and are unique modulo* $\mathtt{E_s}$.

*Proof.* The system es is terminating and so ALC is terminating and ALC-normal forms exist. As all the ALC-critical pairs are joinable (see [19] for full details), ALC is locally confluent and locally coherent. Therefore [17], ALC is confluent and hence ALC-normal forms are unique modulo ALC-equivalence.

We define the following translation $T$ from $\lambda$s-terms to ALC-normal forms.

$$\begin{array}{lll}
T(x) & = x & \\
T(\lambda x.t) & = \lambda x.T(t) & \\
T(t\ u) & = (T(t)\ T(u))[\widehat{y}/T(u)] & \text{where } \widehat{y} \text{ is fresh} \\
T(t[y/u]) & = \mathtt{ALC}(T(t)[y/T(u)]) & \text{if } y \notin \mathtt{fv}(t) \\
T(t[y/u]) & = \mathtt{ALC}(T(t)[y/T(u)][\widehat{y}/T(u)]) & \text{if } y \in \mathtt{fv}(t) \text{ where } \widehat{y} \text{ is fresh}
\end{array}$$

Remark that the translation preserves free variables.

The translation of closures $t[y/u], y \in \mathtt{fv}(t)$ and applications $(t\ u)$ introduces extra substitutions $[\widehat{y}/T(u)]$. We do this for the following technical reason. The use of ALC-normal forms allows us to simulate $\to_{\mathtt{R}}$ reduction with $\to_{\mathtt{Var}}$ reduction. However, $\to_{\mathtt{Var}}$ reduction cannot simulate the reduction $x[x/z] \to_{\mathtt{R}} z[x/z]$ unless the translation were to discard garbage (which is unsound for reasoning about normalisation). Thus, the translation keeps garbage copies of substitutions to allow these cases to be simulated.

**Proposition 28 ($\lambda$es simulates $\lambda_{sub}$).** *If $t \to_{\lambda_{sub}} t'$ then $T(t) \to^{+}_{\lambda\mathtt{es}} T(t')$.*

*Proof.* By induction on the definition of $t \to_{\lambda_{sub}} t'$ using some technical lemmas [19].

**Corollary 29 (SN for $\lambda_{sub}$ (ii)).** *If $t$ is typable in $\mathtt{mul}_{\lambda\mathtt{s}}$, then $t \in \mathcal{SN}_{\lambda_{sub}}$.*

*Proof.* Let $\Gamma \vdash_{\mathtt{mul}_{\lambda\mathtt{s}}} t : A$. We show that $T(t)$ is also typable in $\mathtt{mul}_{\lambda\mathtt{s}}$ by induction on $t$. Then $T(t) \in \mathcal{SN}_{\lambda\mathtt{es}}$ by [18]. Now, suppose $t \notin \mathcal{SN}_{\lambda_{sub}}$. Then given an infinite $\lambda_{sub}$-reduction sequence starting at $t$ we can construct, by Proposition 28, an infinite $\lambda$es-reduction sequence starting at $T(t)$. This leads to a contradiction. Thus $t \in \mathcal{SN}_{\lambda_{sub}}$.

## 6 Relating Partial Substitutions to Graphical Formalisms

### 6.1 MELL Proof-nets

Calculi with explicit substitutions enjoy a nice relation with the multiplicative exponential fragment of linear logic (MELL). This is done by interpreting terms into *proof-nets*, a graphical formalism which represent MELL proofs in natural deduction style. In order to obtain this interpretation, one first defines a (simply) typed version of the term calculus. The translation from $\lambda$s-terms to proof-nets gives a simulation of the reduction rules for explicit substitutions via cut elimination in proof-nets. As an immediate consequence of this simulation, one proves that a simply typed version of the term calculus is strongly normalizing. Also, an important property of the simulation is that each step in the calculus with ES is simulated by a *constant* number of steps in proof-nets: this shows that the two systems are very close, unlike what happens when simulating the $\lambda$-calculus. This gives also a powerful tool to reason about the complexity of $\beta$-reduction.

We apply this idea to the $\lambda_{sub}$-calculus by using previous work based on an interpretation of $\lambda\mathtt{s}$-terms into MELL proof-nets [18] and our translation in Section 5. We thus compose both translations:

Let $t$ be typable in $\mathtt{mul}_{\lambda\mathtt{s}}$. Then the translation of $t$ into a MELL proof-net is given by $W(t) = Z(T(t))$, where $T$ is the translation from $\lambda\mathtt{s}$-terms to $\mathtt{ALC}$-normal forms and $Z$ is the translation from $\lambda\mathtt{s}$-terms to MELL proof-nets given in [18].

Call $R/E$ the strongly normalising reduction relation on MELL proof-nets. Then:

**Proposition 30.** *If $t$ is typable in $\mathtt{mul}_{\lambda\mathtt{s}}$ and $t \to_{\lambda_{sub}} t'$, then $W(t) \to^{+}_{R/E} \mathcal{C}[W(t')]$, where $\mathcal{C}[W(t')]$ denotes a proof-net containing $W(t')$ as a sub proof-net.*

*Proof.* Proposition 28 gives $T(t) \to^{+}_{\lambda\mathtt{es}} T(t')$. Moreover, by a simple inspection of the proof of this proposition we know that there at least one $\mathtt{B}$, $\mathtt{Var}$, or $\mathtt{Gc}$ step in the reduction sequence $T(t) \to^{+}_{\lambda\mathtt{es}} T(t')$. This together with Theorem 8.2 in [18] gives us $W(t) = Z(T(t)) \to^{+}_{R/E} \mathcal{C}[Z(T(t'))] = \mathcal{C}[W(t')]$.

**Corollary 31 (SN for $\lambda_{sub}$ (iii)).** *If $t$ is typable in $\mathtt{mul}_{\lambda\mathtt{s}}$, then $t \in \mathcal{SN}_{\lambda_{sub}}$.*

*Proof.* As $R/E$ is strongly normalising, we conclude $t \in \mathcal{SN}_{\lambda_{sub}}$ using Proposition 30.

### 6.2 Local bigraphs

Milner, Leifer, and Jensen's bigraphical reactive systems [24, 21, 16] have been proposed as a framework for modelling the mobility of distributed agents able to manipulate their own linkages and nested locations. Milner has presented an encoding of $\lambda_{sub}$ as a bigraphical reactive system $'\Lambda\mathrm{BIG}$ as a means to study confluence in bigraphs [25]. This encoding may also be understood as a formalism with *partial* substitutions.

$\lambda_{sub}$ is close to $'\Lambda\mathrm{BIG}$ both statically and dynamically; $\alpha$-equivalent terms have the same encoding and one-step reduction in the former matches one-step reaction in the latter. Thus, any properties proven for $\lambda_{sub}$ hold for the image of the encoding in $'\Lambda\mathrm{BIG}$.

**Proposition 32 ([25]).** *Let $t$ be a $\lambda\mathtt{s}$-term. Then $t \to_{\lambda_{sub}} t'$ iff the encoding of $t$ in $'\Lambda\mathrm{BIG}$ can react in one step to the encoding of $t'$ in $'\Lambda\mathrm{BIG}$.*

Thus, the image $'\Lambda\mathrm{BIG}^{e}$ of the encoding is closed under reaction. We can reason about reaction in $'\Lambda\mathrm{BIG}^{e}$ by considering reduction of $\lambda_{sub}$ terms without metavariables:

**Corollary 33 (Confluence, PSN, SN).** *$'\Lambda\mathrm{BIG}^{e}$ is confluent and satisfies PSN. Encodings of intersection typed terms are strongly normalising.*

## 7 Conclusions

We answer some fundamental remaining questions concerning the adequacy of Milner's $\lambda$-calculus with partial substitutions. In particular, we prove that the $\lambda_{sub}$-calculus is confluent on terms and metaterms, that it enjoys PSN, and that it allows a characterisation of $\lambda_{sub}$-strongly normalising terms by using intersection type disciplines.

We relate $\lambda_{sub}$ to the calculi with definitions $\lambda_{\beta_p}$ and $\lambda_{def}$, thus obtaining a certain number of interesting results concerning normalisation. We also relate the $\lambda_{sub}$-calculus to classical calculi with explicit substitutions. Thus, the $\lambda_{sub}$-calculus can be understood as a concise and simple language implementing partial and ordinary substitution, both in implicit and explicit style at the same time.

Last but not least, we establish a clear connection between simply typed $\lambda_{sub}$-calculus and MELL proof-nets, thus injecting again a graph representation to $\lambda_{sub}$-terms which were inspired from bigraphical reactive systems.

In related work, Bundgaard and Hildebrandt [8] use partial substitution similar to $\lambda_{sub}$ in their extension of Homer, a higher-order process calculus. Partial substitution is also used in different frameworks such as for example Ariola and Felleisen's [2] call-by-need lambda calculus and Ariola and Klop's [3] cyclic $\lambda$-calculus.

Grohmann and Miculan have modelled the call-by-name and call-by-value $\lambda$-calculi with bigraphs [14] by adapting Milner's model. While they concentrate on encodings of $\lambda$-terms, the model is still based on $\lambda_{sub}$ and our results can be used to reason about normalisation and confluence in their models.

*Acknowledgements* We are grateful to V. van Oostrom who pointed out to us references to calculi with partial notions of substitutions such as $\lambda_{\beta_p}$ and $\lambda_{def}$.

## References

1. M. Abadi, L. Cardelli, P. L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.
2. Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.
3. Z. M. Ariola and J. W. Klop. Lambda calculus with explicit recursion. *Information and Computation*, 139(2):154–233, 1997.
4. F. Baader, editor. *Term Rewriting and Applications, 18th International Conference, RTA-07*, volume 4533 of *Lecture Notes in Computer Science*. Springer-Verlag, June 2007.
5. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
6. Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.
7. R. Bloo and K. Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. In *Computer Science in the Netherlands (CSN)*, pages 62–72, 1995.
8. M. Bundgaard and T. T. Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. *Electronic Notes in Theoretical Computer Science*, 154(2):7–29, 2006.
9. M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung*, (19):139–156, 1978.
10. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the $\lambda$-calculus. *Notre-Dame Journal of Formal Logic*, 4(21):685–693, 1980.
11. R. David. A short proof of the strong normalization of the simply typed lambda calculus. Available as `http://www.lama.univ-savoie.fr/~david/`.
12. R. David and B. Guillaume. A $\lambda$-calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11:169–206, 2001.

13. N. 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, pages 71–92, New York, 1987. Marcel Dekker.
14. D. Grohmann and M. Miculan. Directed bigraphs. *Electronic Notes in Theoretical Computer Science*, 173:121–137, 2007.
15. G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Thèse de doctorat d'état, Université Paris VII, 1976.
16. O. H. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, Computer Laboratory, University of Cambridge, February 2004.
17. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.
18. D. Kesner. The theory of calculi with explicit substitutions revisited. In J. Duparc and T. 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, Sept. 2007.
19. D. Kesner and S. Ó Conchúir. Milner's lambda-calculus with partial substitutions, 2008. available on `www.pps.jussieu.fr/~kesner/papers/`.
20. K. Kikuchi. Simple proofs of characterizing strong normalization for explicit substitution calculi. In Baader [4], pages 257–272.
21. J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2000.
22. S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.
23. P.-A. Melliès. Typed $\lambda$-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. 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, Apr. 1995.
24. R. Milner. Computational flux. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 220–221, New York, NY, USA, 2001. ACM Press.
25. R. Milner. Local bigraphs and confluence: two conjectures. In R. Amadio and I. Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency (EXPRESS '06)*. Electronic Notes in Theoretical Computer Science, 2006.
26. R. P. Nederpelt. The fine-structure of lambda calculus. Technical Report Computing Science Notes 92/07, Eindhoven University of Technology, Department of Mathematics and Computer Science, 1992.
27. S. Ó Conchúir. Proving PSN by simulating non-local substitution with local substitution. In D. Kesner, M.-O. Stehr, and F. van Raamsdonk, editors, *Proceedings of the Third International Workshop on Higher-Order Rewriting (HOR)*, pages 37–42, Aug. 2006.
28. S. Ó Conchúir. $\Lambda_{\mathrm{sub}}$ as an explicit substitution calculus. Technical Report TR-2006-95, IT-Universitetet, København, September 2006.
29. G. Pottinger. A type assignment for the strongly normalizable $\lambda$-terms. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 561–577. Academic Press, New York, 1980.
30. P. Severi and E. Poll. Pure type systems with definitions. In *Logical Foundations of Computer Science'94*, volume 813 of *Lecture Notes in Computer Science*, pages 316–328. Springer-Verlag, 1994.
31. D. T. van Daalen. *The language theory of automath*. PhD thesis, Technische Hogeschool Eindhoven, 1977.