Reductions, intersection types, and explicit substitutions

Dan Dougherty[†] and Pierre Lescanne,[‡]

[†]Department of Mathematics and Computer Science, Wesleyan University
Middletown, CT 06459 USA
E-mail: ddougherty@wesleyan.edu

[‡]Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon 07, FRANCE
E-mail: Pierre.Lescanne@ens-lyon.fr

1 Introduction

The λ-calculus plays a key role in the foundations of logic and of programming language design, and in the implementation of logics and languages as well. The foundation of λ-calculus itself is β-conversion, which relates the primitive notions of abstraction and application in terms of *substitution*. Classical λ-calculus treats substitution as an atomic operation, but in the presence of variable-binding substitution it is a complex operation to define and to implement. So a more careful analysis is required if one is to reason about the correctness of compilers, theorem provers, or proof-checkers. Furthermore the actual cost of performing substitution should be considered when reasoning about complexity of implementations.

Abadi, Cardelli, Curien, and Lévy [1] defined a calculus of *explicit substitutions* to serve as a more faithful model of implementations of the λ-calculus. Since then a variety of explicit substitutions calculi have been defined. The original motivation for the Abadi-Cardelli-Curien-Lévy calculus was pragmatic, but there is another point of view one may take on such a calculus, namely that making substitution explicit permits a more refined analysis of substitution than does classical λ-calculus. As historical context we note that in their book [12] Curry and Feys insist on the importance of substitution in logic in general and especially in the framework of λ-calculus. They write [page 6] that the synthetic theory of combinators "gives the ultimate analysis of substitutions in terms of a system of extreme simplicity. The theory of lambda-conversion is intermediate in character between synthetic theories and ordinary logics. Although its analysis is in some ways less profound—many of the complexities in regard to variables are still unanalyzed there—yet it is none the less significant; and it has the advantage of departing less radically from our intuition." From this point of view one can see an explicit substitution calculus as an improvement on both the system of combinators and the classical λ-calculus, since it is a system whose mechanics are first-order and as simple as those of combinatory logic yet which retains the same intensional character as traditional λ-calculus. In particular we may view explicit substitution calculi as primary and see the classical λ-calculus as a subsystem of these systems, defined by a particular strategy of "eagerly" evaluating the substitution constructed by contracting a β-redex. In this way the study of explicit substitutions represents a deeper examination of the relationship between abstraction and application. This setting invites the programme of refining the results of the classical λ-calculus by finding proofs of their explicit-substitutions analogues *in the explicit substitutions system itself.* One can reasonably expect in this way to gain insight into the original λ-calculus. As a case study, in this paper we present a systematic study of the relation between normalization and types.

In many calculi of explicit substitutions, including the original Abadi, Cardelli, Curien, Lévy system, substitutions are first-class citizens and there is an algebraic/computational structure on the substitutions themselves, reflecting the fact that composition is a natural operation on substitutions. Melliès [17] made the somewhat surprising discovery that the presence of substitution-composition leads to the failure of strong normalization even for simply-typed terms. This suggests that it is useful to analyze the effect of making substitution explicit independently of studying composition of substitutions. Composition-free calculi of explicit substitutions have been studied in [16, 7, 4] among others.

Here we work in the composition-free calculus $\lambda\mathtt{x}$ (which uses names rather than de Bruijn indices) and the calculus $\lambda\mathtt{x}_{gc}$ obtained by adding explicit garbage-collection to $\lambda\mathtt{x}$.

Summary of results

Our main results concern the set of terms typable in various intersection-types disciplines. We show that in each of $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$ the terms which normalize by leftmost reduction and the terms which normalize by head reduction can each be characterized as the set of terms typable in a certain system. Our notions of leftmost- and head-reduction are non-deterministic, and our normalization theorems apply to any computations obeying these strategies. In this way we refine and strengthen these classical normalization theorems. See [18] where a similar issue is discussed. Surprisingly, the situation for the strongly normalizing terms diverges from the classical λ-calculus. For the natural generalization of the classical type system we prove that typable terms are strongly normalizing. But the converse fails: see Section 7.

In addition to their theoretical and methodological interest our results have consequences for the study of the implementation of functional programming languages. Recall that the theoretical foundation for the correctness of the standard evaluation strategy for functional languages is the classical theorem that leftmost reduction is normalizing (see for example [19] Prop. 2.4.12). When explicit substitutions are offered as a basis for an implementation one should define and analyze a corresponding notion of "leftmost" reduction. To our knowledge this analysis has not been previously done. The natural notion of leftmost reduction we define here is related to, but a refinement of, the classical notion; the non-determinism in leftmost reduction here corresponds to a choice between certain standard implementation strategies ([5]). The proofs we present here readily yield the results that a term is (strong-, leftmost-, or head-) normalizing iff it is so in the calculus extended by garbage-collection. Our results support the claim that garbage-collection is a very natural addition to the system, even from a purely theoretical point of view, since the resulting calculus has more convenient closure properties than the simple calculus (Lemma 2 is an example).

The intersection type systems we study are natural generalizations of the corresponding classical systems, and in fact the global structure of the proofs follow a standard paradigm (as in [3]). But the explicit reductions involving substitutions lead to combinatorial complications not arising in the traditional treatments and the proofs require some new techniques. The first result on strong normalization of calculi of explicit substitution was the so-called *preservation of strong normalization:* a pure (substitution-free) term is strongly normalizing under reduction in the presence of explicit substitutions if and only if it is strongly normalizing under β-reduction. We stress that, in keeping with our aim of treating the explicit substitutions calculus as logically prior to the traditional λ-calculus, we develop the machinery needed for direct proofs which do not depend on results from the theory of β-reduction.

This paper contains few proofs, but a full version with all the proofs is available at `http://www.ens-lyon.fr/~plescann/publications.html`

2 Terms and reduction strategies

In this section, we describe the terms of the calculus of explicit substitutions with explicit names $\lambda\mathtt{x}$ and specify strategies of reduction toward normal forms, namely $\lambda\mathtt{x}$-reduction, head reduction, and leftmost reduction.

Actually the same set of terms can be described in many different ways which we call *taxonomies.*

Definition 1 (The basic taxonomy). *The set of terms with explicit substitutions $\Lambda\mathtt{x}$ is the set of terms M defined as follows:*

$$M, N ::= x \mid \lambda x.M \mid M\,N \mid M\langle x = N\rangle$$

The set of free variables of a term is defined just as for classical λ-calculus, with an additional clause ensuring that the free variables of $M\langle x = N\rangle$ are the same as the free variables of $(\lambda x.M)N$. In particular, x is bound in $M\langle x = N\rangle$. The set of free variables of a term M is written $FV(M)$, sometimes for simplicity we write $x \in M$ instead of $x \in FV(M)$.

We assume Barendregt's [2] convention, namely that *a variable does not occur free and bound in the same term.* For instance, we assume that x does not occur free in N in the term $M\langle x = N\rangle$. The rules we define further assume this convention and the reader should keep this fact in mind when reading them and certain forthcoming lemmas.

To describe the second taxonomy nicely it will be very convenient to have a notation to describe a term M on which is applied a sequence of closures $\langle z_1 = S_1\rangle$, ..., $\langle z_m = S_m\rangle$ then a sequence of applications of terms T_1,..., T_n. Such a term $M\langle z_1 = S_1\rangle ... \langle z_m = S_m\rangle T_1 ... T_n$ will be abbreviated as $M\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$.

Lemma 1 (The head form taxonomy). *Every term is of precisely one of the following forms:*

$$
\begin{array}{ll}
\lambda x.B & (\lambda y.B)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T} \\
(\lambda x.B)AT_1 \cdots T_n \quad \textit{with} \quad n \geq 0 & (UV)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T} \\
xT_1 \cdots T_n \quad \textit{with} \quad n \geq 0 & x\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T} \\
 & y\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T} \quad \textit{with} \quad x \not\equiv y
\end{array}
$$

Proof. Straightforward. ///

Following Barendregt [2] we distinguish between a set of rules defining a *notion of reduction* and a *reduction relation* induced by closing a notion of reduction under certain contexts. Sometimes the latter are called strategies and play a main role in evaluation of functional programming languages [5]. Some reductions are deterministic, which means that the structural rules determine a unique redex to be reduced. Others are non deterministic.

The following notion of reduction is due to R. Bloo and K. Rose [8, 21, 6]. The rules VarI and VarK, called respectively xv and xvgc by Rose, have been renamed here to recall the distinction between the classical λ_I and λ_K calculi.

Definition 2. *The notions of reduction $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$ are induced by the rules in Table 1: the notion of reduction $\lambda\mathtt{x}$ is obtained by deleting the rule gc, and the notion of reduction $\lambda\mathtt{x}_{gc}$ is obtained by deleting the rule VarK.*

The rule gc is called "garbage collection", as it removes useless substitutions.

(B)	$(\lambda x B)\, A$	$\to$	$B\langle x = A\rangle$	
(App)	$(MN)\langle x = A\rangle$	$\to$	$M\langle x = A\rangle\, N\langle x = A\rangle$	
(Abs)	$(\lambda y M)\langle x = N\rangle$	$\to$	$\lambda y M\langle x = N\rangle$	
(VarI)	$x\langle x = N\rangle$	$\to$	N	
(VarK)	$y\langle x = N\rangle$	$\to$	y	
(gc)	$M\langle x = A\rangle$	$\to$	M	if $x \notin FV(M)$

Table 1. The reduction rules.

Of course in the presence of rule gc we do not need rule VarK. On the other hand it is not the case that gc can be directly simulated by the other rules: consider the garbage-collection $x\langle x = y\rangle\langle v = w\rangle \longrightarrow x\langle x = y\rangle$. Rule gc has a different character from the other rules in the sense that it represents a more complex transformation than those of the other, atomic, substitution operations. On the other hand with an appropriate data structure for maintaining (the free variables in) terms it can be efficiently implemented and provides a tool to prevent memory leaks. And as Bloo and Rose have demonstrated it is quite convenient when reasoning about the formal properties of the calculus.

Notation. In the main technical development of this paper we will work exclusively with the full system $\lambda\mathtt{x}_{gc}$. So unless explicitly stated otherwise, phrases such as "reduction" refer to reduction in system $\lambda\mathtt{x}_{gc}$. At the end of the paper (Section 8) we will see that the results for the system not including garbage collection follow readily from the results for $\lambda\mathtt{x}_{gc}$.

Remark. As is well-known, $\lambda\mathtt{x}_{gc}$ has a critical pair, namely:

$$\begin{tikzcd}[column sep=small]
 & ((\lambda x.M)N)\langle y = L\rangle \arrow[dl] \arrow[dr] & \\
(\lambda x.M\langle y = L\rangle)\ N\langle y = L\rangle & & M\langle x = N\rangle\langle y = L\rangle
\end{tikzcd}$$

Most of the difficulty in working with the system is due to this critical pair; this will be amply demonstrated in the sequel.

Definition 3 (Unrestricted reduction). *Unrestricted reduction (or $\lambda\mathtt{x}_{gc}$-reduction) allows a reduction rule to be applied in any context.*

A term M is strongly normalizing *if there is no infinite $\lambda\mathtt{x}_{gc}$-reduction starting from M. The set of strongly normalizing terms is denoted $\mathcal{SN}$.*

Definition 4 (Head reduction). Head reduction *is the closure of $\lambda\mathtt{x}_{gc}$ under the structural rules of Table 2.*

$$\frac{U \xrightarrow{h} U' \qquad U \text{ is not an abstraction}}{UV \xrightarrow{h} U'V} \qquad\qquad \frac{B \xrightarrow{h} B'}{\lambda x.B \xrightarrow{h} \lambda x.B'}$$

$$\frac{M \xrightarrow{h} M' \qquad M \text{ is not an abstraction}}{M\langle x = A\rangle \xrightarrow{h} M'\langle x = A\rangle}$$

Table 2. Head reduction

A term M is head normalizing *if there is no infinite head-reduction starting from M. The set of head normalizing terms is denoted $\mathcal{HN}$.*

A head normal form *is a term of the form $\lambda x_1..x_k.xA_1...A_n$ where x is a free variable or one of the x_i and $A_i \in \Lambda\boldsymbol{x}$.*

$$\frac{U \xrightarrow{l} U' \quad U \text{ is not an abstraction}}{UV \xrightarrow{l} U'V} \qquad \frac{B \xrightarrow{l} B'}{\lambda x.B \xrightarrow{l} \lambda x.B'}$$

$$\frac{M \xrightarrow{l} M' \quad M \text{ is not an abstraction}}{M\langle x = A\rangle \xrightarrow{l} M'\langle x = A\rangle} \qquad \frac{A_i \xrightarrow{l} A_i' \quad A_i \text{ is the leftmost non-normal form}}{xA_1...A_i...A_n \xrightarrow{l} xA_1...A_i'...A_n}$$

Table 3. Leftmost reduction

Definition 5 (Leftmost reduction). Leftmost reduction *is the closure of* $\lambda\mathtt{x}_{gc}$ *under the structural rules in Table 3.*

A term M is leftmost normalizing *if there is no infinite leftmost reduction starting from M. The set of leftmost-normalizing terms is denoted $\mathcal{LN}$.*

Remark. Observe that in contrast to the classical notions both head reduction and leftmost reduction are nondeterministic strategies. Indeed both reductions out of the critical pair noted earlier count as head reductions. For example, let M be $((\lambda x.B)A)\langle y = C\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$. Then M can rewrite by leftmost reduction either to $P \equiv B\langle x = A\rangle\langle y = C\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$, or (in two steps) to $Q \equiv ((\lambda x.B\langle y = C\rangle)\ A\langle y = C\rangle)\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$. Then since $\lambda x.B\langle y = C\rangle$ is an abstraction Q leftmost-rewrites via rule B leading to $Q' \equiv B\langle y = C\rangle\langle x = A\langle y = C\rangle\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$.

3 Two fundamental Lemmas

To prove the main theorems of this paper, we need two very general lemmas which we present in this section. These lemmas aim at proving the following two facts (where $\mathcal{N}$ stands for $\mathcal{SN}$, $\mathcal{LN}$, or $\mathcal{HN}$).

$$M\langle x = N\rangle\langle y = L\rangle \in \mathcal{N} \text{ if } M\langle y = L\rangle\langle x = N\langle y = L\rangle\rangle \in \mathcal{N},$$

and

$$M\langle x = A\rangle \in \mathcal{N} \text{ if } M \in \mathcal{N} \text{ and } x \notin FV(M),$$

where, when $\mathcal{N}$ is $\mathcal{SN}$, we require $A \in \mathcal{SN}$ as well.

A remark on the classical Substitution Lemma

The Substitution Lemma of the classical λ-calculus [2] states a fundamental property of (implicit) substitutions, namely that, when x is not free in L

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

The two terms are syntactically identical above. When generalized to an explicit substitutions calculus the analogous statement is weakened to provable equality:

$$M\langle x = N\rangle\langle y = L\rangle = M\langle y = L\rangle\langle x = N\langle y = L\rangle\rangle$$

It is not hard to see that the two terms above can have quite different reduction behavior. In particular it is possible for the left-hand side to be $\mathcal{SN}$ while the right-hand side admits an infinite reduction. For instance, take $M \equiv z$, $N \equiv yy$ and $L \equiv \lambda u.uu$.

The Composition Lemma below states that if the right-hand side is $\mathcal{SN}$ then so is the left-hand side. So there is a fundamental asymmetry in this situation.

3.1 The Composition Lemma

Let us consider the following rule

$$M\langle x = N\rangle\langle y = L\rangle \;\rightarrow\; M\langle y = L\rangle\langle x = N\langle y = L\rangle\rangle$$

which we call *composition*. It abstracts the composition one finds in systems like $\lambda\sigma$ [1, 11] (namely the rule called *Map*) or in the extension $\lambda\mathtt{x}\|c$ of $\lambda\mathtt{x}$ [6, 7, 14, 15] (namely the rule $\xrightarrow[\|c]{}$ of [6], see also [21] page 75).

We would like to see that the converse of the composition rule preserves (strong, head, or leftmost) normalization. Unfortunately this rule does not commute in a nice way with reduction, essentially due to the duplication of substitutions in the App rule.

The following relation is a "bottom-up parallel extension" of the composition rule, which propagates and duplicates the applications of this rule inside terms. In particular, rule Cpabs pushes a substitution through an abstraction, Cpapp pushes through an application, and Cpclo pushes through a closure. The other rules make it a congruence.

Definition 6. *The relation $\Rightarrow$ is given by the inductive definition indicated in Table 4.*

[Cref]	$M \Rightarrow M$		
[Cabs]	$\dfrac{B \Rightarrow B'}{\lambda x.B \Rightarrow \lambda x.B'}$	[Cpabs]	$\dfrac{B\langle y = Q\rangle \Rightarrow B^+}{(\lambda x.B)\langle y = Q\rangle \Rightarrow \lambda x.B^+}$
[Capp]	$\dfrac{U \Rightarrow U' \qquad V \Rightarrow V'}{UV \Rightarrow U'V'}$	[Cpapp]	$\dfrac{U\langle y = Q\rangle \Rightarrow U^+ \qquad V\langle y = Q\rangle \Rightarrow V^+}{(UV)\langle y = Q\rangle \Rightarrow (U^+V^+)}$
[Cclo]	$\dfrac{B \Rightarrow B' \qquad A \Rightarrow A'}{B\langle z = A\rangle \Rightarrow B'\langle z = A'\rangle}$	[Cpclo]	$\dfrac{M\langle y = Q\rangle \Rightarrow M^+ \qquad P\langle y = Q\rangle \Rightarrow P^+}{M\langle x = P\rangle\langle y = Q\rangle \Rightarrow M^+\langle x = P^+\rangle}$

Table 4. The rules for $\Rightarrow$

Lemma 2. *Let* $\longrightarrow$ *stand for either unrestricted reduction, leftmost reduction, or head reduction. If* $M \longrightarrow M''$ *and* $M \Rightarrow M'$ *then there is an* M^* *with* $M'' \Rightarrow M^*$ *and* $M' \longrightarrow M^*$*. Furthermore, if* $M \longrightarrow M''$ *is a B-step then in fact* $M' \xrightarrow{+} M^*$ *with at least one B-step.*

$$\begin{tikzcd} M \arrow[r, Rightarrow] \arrow[d] & M' \arrow[d, dashed, twoheadrightarrow] \\ M'' \arrow[r, Rightarrow, dotted] & M^* \end{tikzcd}$$

Proof. By induction over the definition of $M \Rightarrow M'$. ///

Corollary 1. *Suppose* $M \Rightarrow M'$*.*

- *If* $M' \in \mathcal{HN}$ *then* $M \in \mathcal{HN}$*.*
- *If* $M' \in \mathcal{LN}$ *then* $M \in \mathcal{LN}$*.*
- *If* $M' \in \mathcal{SN}$ *then* $M \in \mathcal{SN}$*.*

Proof. As is well-known, the set of rules of $\lambda\mathtt{x}_{gc}$ other than the B-rule comprise a strongly normalizing rewrite system. So any infinite reduction out of M must involve infinitely many B-steps. With this observation each of the claims follows easily from Lemma 2. ///

In particular, suppose that if T' is obtained from T by the composition rule as defined at the beginning of this section. We conclude that if T' is strongly normalizing then T is strongly normalizing; similarly for leftmost and head normalization. These results will be crucial in the coming sections.

3.2 The Closure lemma

Now we want to prove that (head, leftmost, or strong) normalization is not affected by garbage-collection of normalizing terms. For technical reasons we state the result rather generally.

Definition 7. *A* n*-multi-context is a term with* n *holes in which we can insert* n *terms. If* n *is understood, we say a multi-context.*

If $C[\![\ldots,\ldots,\ldots]\!]$ is a multi-context and $M_1, \ldots, M_n$ are terms, then the insertions of those terms in $C[\![\ldots,\ldots,\ldots]\!]$ is denoted $C[\![M_1,\ldots,M_n]\!]$.

Lemma 3. *Let* $C[\![\ldots]\!]$ *be a multi-context,* $A_1,\ldots,A_n$*, and* $M_1,\ldots,M_n$ *be terms, with* $x \notin FV(M_1), \ldots, x \notin FV(M_n)$*.*

- *if* $C[\![M_1,...,M_n]\!] \in \mathcal{HN}$ *then* $C[\![M_1\langle x = A_1\rangle,...,M_n\langle x = A_n\rangle]\!] \in \mathcal{HN}$*.*
- *if* $C[\![M_1,...,M_n]\!] \in \mathcal{LN}$ *then* $C[\![M_1\langle x = A_1\rangle,...,M_n\langle x = A_n\rangle]\!] \in \mathcal{LN}$*.*
- *if* $C[\![M_1,...,M_n]\!] \in \mathcal{SN}$ *and* $A_i \in \mathcal{SN}$ *for* $1 \le i \le n$ *then* $C[\![M_1\langle x = A_1\rangle,...,M_n\langle x = A_n\rangle]\!] \in \mathcal{SN}$*.*

Proof. By induction on triples $(D, \boldsymbol{M}, \boldsymbol{A})$ where D is a term, $\boldsymbol{M}$ and $\boldsymbol{A}$ are multisets of terms. ///

Corollary 2. *Let* $M \equiv N\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$ *with* $x \notin FV(N)$ *and let* $M' \equiv N\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}$*,*

- *If* $M' \in \mathcal{HN}$ *then* $M \in \mathcal{HN}$*.*
- *If* $M' \in \mathcal{LN}$ *then* $M \in \mathcal{LN}$*.*
- *If* $M' \in \mathcal{SN}$ *and* $A \in \mathcal{SN}$ *then* $M \in \mathcal{SN}$*.*

4 Saturated Sets

Definition 8. *A set* $\mathcal{S}$ *is* $\mathcal{X}$*-saturated (or saturated if there is no ambiguity about the set* $\mathcal{X}$*), if it is closed under the rules of inference in Table 5.*

$$\textsf{sat-B}\ \frac{B\langle x = A\rangle\, \boldsymbol{T}}{(\lambda x.B)A\, \boldsymbol{T}} \qquad \textsf{sat-I}\ \frac{A\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}{x\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}$$

$$\textsf{sat-Abs}\ \frac{(\lambda y.B\langle x = A\rangle)\,\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}{(\lambda y.B)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}} \qquad \textsf{sat-App}\ \frac{(U\langle x = A\rangle)(V\langle x = A\rangle)\,\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}{(UV)\langle x = A\rangle\,\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}$$

$$\textsf{sat-comp}\ \frac{M\langle y = Q\rangle\langle x = P\langle y = Q\rangle\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}{M\langle x = P\rangle\langle y = Q\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}} \qquad \textsf{sat-gc}\ \frac{N\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T} \qquad A \in \mathcal{X} \text{ and } x \notin FV(N)}{N\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle \boldsymbol{T}}$$

Table 5. $\mathcal{X}$-saturated sets

Note that the set $\mathcal{X}$ occurs only in the rule sat-gc. In practice the set $\mathcal{X}$ will depend on the reduction we consider.

Definition 9 (Function space). *If* $\mathcal{A}$ *and* $\mathcal{B}$ *are sets of terms then* $\mathcal{A} \twoheadrightarrow \mathcal{B}$ *is* $\{M \mid \forall A \in \mathcal{A}, (MA) \in \mathcal{B}\}$*.*

The following are very easy consequences of the definition.

Lemma 4. *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be sets of terms.*

1. *If* $\mathcal{B}$ *is saturated then so is* $\mathcal{A} \twoheadrightarrow \mathcal{B}$
2. *If* $\mathcal{A}$ *and* $\mathcal{B}$ *are saturated then so is* $\mathcal{A} \cap \mathcal{B}$*.*

The major part of the technical difficulty in lifting the classical normalization proofs to our explicit substitutions setting is embodied in the next Lemma. In fact all of the work in the previous section was for the purpose of establishing these results.

Lemma 5.

- $\mathcal{SN}$ *is* $\mathcal{SN}$*-saturated.*

– $\mathcal{HN}$ *is* $\Lambda\mathbf{x}$*-saturated.*
– $\mathcal{LN}$ *is* $\Lambda\mathbf{x}$*-saturated.*

The notion of saturation is key to the proof of the Soundness Theorem below for the type systems. It is amusing to note that closure under sat-gc for $\mathcal{SN}$, $\mathcal{HN}$, and $\mathcal{LN}$ is used precisely in showing that the start rule below for typing variables is sound: in the standard λ-calculus this is a triviality but in our calculus we ultimately rely on the difficult argument embodied in Lemma 3.

5 Types and Soundness

Definition 10 (The system of type assignment $\mathcal{D}_\omega$). *Given an infinite set of type-variables and a distinguished type-constant ω the set of types is formed by closing the type-variables and ω under the operations $\sigma \to \tau$ and $\sigma \cap \tau$.*

A statement is an expression of the form $M : \tau$; where M is a term, the subject of the statement, and τ is a type. A basis is a set of statements with distinct variables as subjects. A judgment is a triple Γ, M, τ where Γ is a basis, M is a term, and τ is a type; the notion of a judgment's being derivable, denoted $\Gamma \vdash M : \tau$ is given by the rules of inference in Table 6.

We say that a term M is typable if there exists a Γ and a τ such that $\Gamma \vdash M : \tau$.

We identify two systems: the system $\mathcal{D}_\omega$ itself and the subsystem $\mathcal{D}$ obtained by omitting type ω and the rule ω-I.

$$\text{start}\quad \frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$\to\text{I}\quad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \to\text{E}\quad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau}$$

$$\cap\text{-I}\quad \frac{\Gamma \vdash M : \sigma_1 \qquad \Gamma \vdash M : \sigma_2}{\Gamma \vdash M : \sigma_1 \cap \sigma_2} \qquad \cap\text{-E}\quad \frac{\Gamma \vdash M : \sigma_1 \cap \sigma_2}{\Gamma \vdash M : \sigma_i}\quad i \in \{1, 2\}$$

$$\omega\text{-I}\quad \Gamma \vdash M : \omega \qquad \text{cut}\quad \frac{x : \sigma, \Gamma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\langle x = N\rangle : \tau}$$

Table 6. Typing rules for $\mathcal{D}_\omega$

The form of the cut rule ensures that a closure $M\langle x = N\rangle$ has exactly the same typing behavior as the associated B-redex $(\lambda x.M)N$. That is, for every Γ and τ,

$$\Gamma \vdash M\langle x = N\rangle : \tau \qquad \text{iff} \qquad \Gamma \vdash (\lambda x.M)N : \tau.$$

Definition 11. *An interpretation $\mathcal{I}$ is a function from types to sets of terms obeying the following*

- $\mathcal{I}_\omega = \Lambda\mathbf{x}$ *(in system $\mathcal{D}_\omega$)*
- $\mathcal{I}_{\alpha\cap\beta} = \mathcal{I}_\alpha \cap \mathcal{I}_\beta$
- $\mathcal{I}_{\alpha\to\beta} = \mathcal{I}_\alpha \twoheadrightarrow \mathcal{I}_\beta$

Obviously an interpretation is completely determined by its value on the type variables. Suppose $\mathcal{I}$ is an interpretation and $\mathcal{X}$ is a set of terms such that $\mathcal{I}_t$ is $\mathcal{X}$-saturated for each type-variable t. Then $\mathcal{I}_\tau$ is $\mathcal{X}$-saturated for each type τ: for $\tau = \omega$ we observe that $\Lambda\mathrm{x}$ is itself $\mathcal{X}$-saturated, and for the other types we invoke Lemma 4.

Theorem 1 (Soundness). *Let $\mathcal{I}$ be an interpretation and let $\mathcal{X}$ be a class of terms such that $\mathcal{I}_t$ is $\mathcal{X}$-saturated for each type-variable t and such that $\mathcal{I}_\sigma \subseteq \mathcal{X}$ for each type σ.*

Suppose M is typable with type τ in either $\mathcal{D}$ or $\mathcal{D}_\omega$. Then $M \in \mathcal{I}_\tau$.

6 Normalization

Definition 12. *(See Cardone and Coppo [10]) A type is* proper *if it has no positive occurrence of ω,* antiproper *if it has no negative occurrence of ω, and* strictly proper *if it has no occurrence of ω.*

The trivial *types are determined by the following rules:*

- *ω is trivial.*
- *If σ is trivial and θ is any type, then $\theta \to \sigma$ is trivial.*
- *If σ and τ are trivial, then $\sigma \cap \tau$ is trivial.*

Head normalization. Consider the system $\mathcal{D}_\omega$; let "type" mean "type of $\mathcal{D}_\omega$" and let "typable" mean "typable in $\mathcal{D}_\omega$".

Definition 13. *Let $\mathcal{H}$ be the interpretation which maps each type variable to the set $\mathcal{HN}$ of head normalizing terms.*

Lemma 6. *$\mathcal{H}_\tau \subseteq \mathcal{HN}$ for each non-trivial type τ.*

Corollary 3. *If M is typable in $\mathcal{D}_\omega$ with a non-trivial type then M is head normalizing.*

Leftmost normalization. Consider the system $\mathcal{D}_\omega$; let "type" mean "type of $\mathcal{D}_\omega$" and let "typable" mean "typable in $\mathcal{D}_\omega$".

Definition 14. *Let $\mathcal{L}$ be the interpretation which maps each type variable to the set $\mathcal{LN}$ of leftmost-normalizing terms.*

Lemma 7. *$\mathcal{L}_\tau \subseteq \mathcal{LN}$ for each proper type τ.*

Corollary 4. *If M is typable in $\mathcal{D}_\omega$ with a proper type then M is leftmost normalizing.*

Strong normalization. Consider the system $\mathcal{D}$; let "type" mean "type of $\mathcal{D}$" and let "typable" mean "typable in $\mathcal{D}$.

Definition 15. *Let $\mathcal{S}$ be the interpretation which maps each type variable to the set $\mathcal{SN}$ of strongly normalizing terms.*

Lemma 8. *$\mathcal{S}_\tau \subseteq \mathcal{SN}$ for each type τ.*

Corollary 5. *If M is typable in $\mathcal{D}$ then M is strongly normalizing.*

7 Typings for normalizable terms

Proposition 1.

1. *If H is a head normal form then H is typable in system $\mathcal{D}_\omega$ with a non-trivial type.*
2. *If N is a normal form then N is typable in system $\mathcal{D}$.*

Theorem 2 (Subject Reduction). *In either of the systems $\mathcal{D}_\omega$ or $\mathcal{D}$: suppose $\Gamma \vdash M : \tau$ and $M \longrightarrow M_1$. Then $\Gamma \vdash M_1 : \tau$.*

Theorem 3 (Subject Expansion for $\mathcal{D}_\omega$). *In system $\mathcal{D}_\omega$: suppose $\Gamma \vdash M : \tau$ and $M_0 \longrightarrow M$. Then $\Gamma \vdash M_0 : \tau$.*

Corollary 6. *Suppose $\Gamma \vdash M : \tau$ in system $\mathcal{D}_\omega$ and $M \twoheadleftarrow\!\!\twoheadrightarrow M'$. Then $\Gamma \vdash M' : \tau$.*

Subject Expansion and system $\boldsymbol{\mathcal{D}}$.

The Subject Expansion theorem plays a key role in deriving converses involving system $\mathcal{D}_\omega$ to the normalization results concerning head- and leftmost reduction (Corollaries 3 and 4 above). We present these converses in the next section.

It is well-known from the classical λ-calculus that the Subject Expansion theorem fails for system $\mathcal{D}$. But with some care (involving the potential erasing of non-typable terms) one can analyze β-expansion in order to derive a converse to the classical version of Corollary 5 and so obtain a characterization of the strongly normalizing classical terms.

It seems to be much more difficult to perform such an analysis for expansion in the calculus $\lambda\mathbf{x}$. In particular it is not the case that $\mathcal{D}$-typability is preserved by expansion even when the reduction-rule in question erases a strongly-normalizing subterm.

Example. Let D be the term $\lambda u.uu$ and M be the term $(\lambda x.(\lambda y.z)(xx))D$. D is $\mathcal{D}$-typable by $(t \cap (t \to t)) \to t$ but DD is not $\mathcal{SN}$ so M is not $\mathcal{SN}$. Now consider

$$M \longrightarrow (\lambda x.z\langle y = xx\rangle)D \longrightarrow M' \equiv z\langle y = xx\rangle\langle x = D\rangle \longrightarrow M'' \equiv z\langle x = D\rangle$$

M'' is $\mathcal{SN}$ and is easily seen to be $\mathcal{D}$-typable. But M' is $\mathcal{SN}$ yet not $\mathcal{D}$-typable.

It is not hard to see that M' is $\mathcal{SN}$. To see that M' is not $\mathcal{D}$-typable, first note that by Corollary 5 M cannot be typed since it is not $\mathcal{SN}$. But $(\lambda y.z)(xx)$ and $z\langle y = xx\rangle$ have exactly the same typing behavior in our system.

The reduction from M' to M'' witnesses the failure of Subject Expansion; the notable thing here is the innocuous nature of the erased subterm xx. The reduction $(\lambda y.z)(xx) \longrightarrow z\langle y = xx\rangle$, which reduces an inner B-redex, changes the behavior of the term w.r.t. to normalization. This should somewhat be translated into the typing system, which is not the case in $\mathcal{D}$.

The natural reaction to such an example is conclude that the type system $\mathcal{D}$ should be modified. But the terms M and M' above are related simply by an application of the B-rule. So if we are to have a type system which characterizes strong normalization it seems that we must abandon the property that closures $B\langle x = A\rangle$ have the same typing behavior as the associated B-redexes $(\lambda x.B)A$ (perhaps only in the case when x is not free in B). This would be a fundamental change in what seems to us to be the most natural generalization of the classical type system. We leave the search for a type system characterizing the strongly normalizing terms as a subject for future investigation.

8 Summary

In this section we summarize the results of this paper and also address the role of the garbage-collection rule gc in the development.

As suggested in the introduction one may view the rule gc as being somewhat out of character with the rest of the explicit substitutions program, since it does not really correspond to an *atomic* operation on terms. So it is natural to ask whether the relationships we have established between typings and reduction properties carries for the "pure" calculus without rule gc.

Since the pure calculus is a subsystem of the full (gc) calculus one direction of the relationship is immediate, but it is mildly surprising that the full equivalence between various normalization properties and typing properties can be established for the pure calculus with essentially no extra work. This is shown in the three theorems of this section.

Recall that head- and leftmost reduction are each non-deterministic and that when we speak of head- or leftmost-reduction below we mean *any* sequence of reduction steps obeying the given discipline.

Theorem 4 (Head normalization). *Let M be a closed term. The following are equivalent.*

1. *M is typable with a non-trivial type in system $\mathcal{D}_\omega$.*
2. *$M \in \mathcal{HN}$.*
3. *M is head-normalizing in the calculus $\lambda\mathsf{x}$ (without garbage-collection).*
4. *M has a head normal form.*
5. *M is solvable, that is, there is an n and terms $X_1, \ldots X_n$ such that $MX_1 \cdots X_n = \lambda x.x$.*

Proof. We prove $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 1$ and $4 \Rightarrow 5 \Rightarrow 2$. ///

Theorem 5 (Leftmost normalization). *Let M be a closed term. The following are equivalent.*

1. *M is typable in system $\mathcal{D}_\omega$ with a type not involving ω.*
2. *M is typable with a proper type in system $\mathcal{D}_\omega$.*
3. *$M \in \mathcal{LN}$.*
4. *M is leftmost-normalizing in the calculus $\lambda\mathtt{x}$ (without garbage-collection).*
5. *M has a normal form.*

Proof. We prove $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 1$. ///

It is worth emphasizing the fact that the implications 5 to 3 and 5 to 4 state that in $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$ leftmost reduction is a normalizing strategy.

Theorem 6 (Strong normalization). *Let M be a closed term.*

1. *$M \in \mathcal{SN}$ if and only if M is strongly normalizing in the calculus $\lambda\mathtt{x}$ (without garbage-collection).*
2. *If M is typable in system $\mathcal{D}$ then $M \in \mathcal{SN}$.*
3. *If M is a pure term then $M \in \mathcal{SN}$ if and only if M is typable in system $\mathcal{D}$.*

As described in the previous section we do not have the implication "$M \in \mathcal{SN}$ implies M is typable in system $\mathcal{D}$." As is well-known this result holds for pure terms under β-reduction. It then follows from the preservation of strong normalization in $\lambda\mathtt{x}$ that the result holds for pure terms under $\lambda\mathtt{x}_{gc}$-reduction. The problem of finding a reasonable type system characterizing the strongly normalizing terms in $\Lambda\mathtt{x}$ is left as an open problem.

Acknowledgements The authors are grateful to Eduardo Bonelli, Nachum Dershowitz, Delia Kesner, Frédéric Lang, and Kristoffer Rose for many helpful discussions, and to several anonymous referees for improvements in style and substance.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
2. H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier, Amsterdam, 1984. Second edition.
3. H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabby, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 2, pages 117–309. Oxford University Press, 1992.
4. Z. 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, September 1996.
5. Z. Benaissa, P. Lescanne, and K. H. Rose. Modeling sharing and recursion for weak reduction strategies using explicit substitution. In H. Kuchen and D. Swierstra, editors, *PLILP '96—8th Int. Symp. on Programming Languages: Implementation, Logics and Programs*, number 1140 in LNCS, pages 393–407, Aachen, Germany, September 1996. Springer-Verlag.

6. R. Bloo. *Preservation of Termination for Explicit Substitution*. PhD thesis, Technische Universiteit Eindhoven, 1997. IPA Dissertation Series 1997-05.
7. R. Bloo and J. H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211:375 – 395, 1999.
8. R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95—Computing Science in the Netherlands*, pages 62–72, Utrecht, November 1995.
9. E. Bonelli. Perpetuality in a named lambda calculus with explicit substitutions and some applications. Technical Report RR 1221, LRI, University of Paris-Sud, 1999. to appear in MSCS, 2000.
10. F. Cardone and M. Coppo. Two extension of Curry's type inference system. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Series*, pages 19–75. Academic Press, New York, NY, 1990.
11. P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996.
12. H. B Curry and R. Feys. *Combinatory Logic I*. North-Holland, Amsterdam, 1958.
13. R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *LICS '97—Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 35–46. Warsaw U., IEEE, June 1997.
14. F. Kamareddine and A. Ríos. Relating the lambda-sigma and lambda-s styles of explicit substitutions. *Journal of Logic and Computation*, 10(3), 2000. Special issue on Type Theory and Term Rewriting.
15. F. Kamereddine and R.P. Nederpelt. On stepwise explicit substitutions. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.
16. P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$: a journey through calculi of explicit substitutions. In Hans-J. Boehm, editor, *POPL '94—21st Annual ACM Symposium on Principles of Programming Languages*, pages 60–69, Portland, Oregon, January 1994. ACM.
17. P.-A. Melliès. Typed λ-calculi with explicit substitution may not terminate. In M. Dezani, editor, *TLCA '95—Int. Conf. on Typed Lambda Calculus and Applications*, volume 902 of *LNCS*, pages 328–334, Edinburgh, Scotland, April 1995. Springer-Verlag.
18. P.-A. Melliès. Axiomatic rewriting theory III, a factorisation theorem in rewriting theory. In *Proceedings of the 7th Conference on Category Theory and Computer Science*, volume 1290 of *LNCS*, pages 49–68. Springer-Verlag, 1997.
19. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, MA, 1996.
20. E. Ritter. Characterising explicit substitutions which preserve termination. In *TLCA'99*, volume 1581 of *LNCS*. Springer-Verlag, 1999.
21. K.H. Rose. *Operational Reduction Models for Functional Programming Languages*. PhD thesis, DIKU, København, February 1996. DIKU report 96/1.