Under consideration for publication in Math. Struct. in Comp. Science

Reductions, intersection types, and explicit substitutions †

Dan Dougherty[1] and Pierre Lescanne[2]

[1] *Department of Mathematics and Computer Science, Wesleyan University Middletown, CT 06459 USA.*

[2] *Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon 46, Allée d'Italie, 69364 Lyon 07, FRANCE.*

Received 21 January 2002

This paper is part of a general programme of treating explicit substitutions as the primary λ-calculi from the point of view of foundations as well as applications. We work in a composition-free calculus of explicit substitutions and an augmented calculus obtained by adding explicit garbage-collection, and explore the relationship between intersection-types and reduction.
We show that the terms which normalize by leftmost reduction and the terms which normalize by head reduction can each be characterized as the terms typable in a certain system. The relationship between typability and strong normalization is subtly different from the classical case: we show that typable terms are strongly normalizing but give a counterexample to the converse. 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 the classical normalization theorems. The proofs require some new techniques in the presence of reductions involving explicit substitutions. Indeed, in our proofs we do not rely on results from classical λ-calculus, which in our view is subordinate to the calculus of explicit substitution.

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 abstraction to application in terms of *substitution*. Classical λ-calculus treats substitution as an atomic operation, but in the presence of variable-binding substitution 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 substitutions should be considered when reasoning about complexity of implementations.

† A preliminary abstract of this paper was presented at the Fifth International Conference on Typed Lambda Calculi and Applications (TLCA '2001)

Abadi, Cardelli, Curien, and Lévy (Abadi, Cardelli, Curien & Lévy 1991) 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 Curry and Feys insist in their book (Curry & Feys 1958) 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 the strategy of "eagerly" applying the substitution induced 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 (Melliès 1995) 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 (Lescanne 1994, Bloo & Rose 1995, Kamareddine & Ríos 1997, Bloo & Geuvers 1999, Benaissa, Briaud, Lescanne & Rouyer-Degli 1996) 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 (Melliès 1997) 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, as we will see.

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 Prop. 2.4.12 in (Mitchell 1996). 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 ((Benaissa, Lescanne & Rose 1996)).

The proofs we present here readily yield the results that a term is strong-, leftmost-, or head-normalizing if and only if 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 3.2 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 (Barendregt 1992)). But the explicit reductions involving substitutions lead to combinatorial complications not arising in the traditional treatments and the proofs require some new techniques (see especially Section 3).

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.

Related work

The system of intersection types is due to Coppo and Dezani (Coppo & Dezani-Ciancaglini 1978) and Sallé (Sallé 1978). The fact that the strongly normalizing terms are precisely the typable terms seems to have been first proved in (Pottinger 1980). Other notable works in this area include (Leivant 1986, van Bakel 1992, Ghilezan 1996), and the books (Krivine 1993) and (Amadio & Curien 1998).

The first proof — actually a sketch of a proof — of preservation of strong normalization,

or PSN, appears in (Lescanne & Rouyer-Degli 1994) for $\lambda\upsilon$ ((Benaissa, Briaud, Lescanne & Rouyer-Degli 1996) has the complete proof) and at almost the same time Bloo and Rose (Bloo & Rose 1995) proposed a proof of PSN for $\lambda\mathtt{x}$ (see also (Bloo 1997, Bloo & Geuvers 1999, Rose 1996)). Kamareddine and Rios (Kamareddine & Ríos 1995, Kamareddine & Ríos 1997) gave proofs of PSN for another calculus of explicit substitutions called λs. The proof of Di-Cosmo and Kesner (Di Cosmo & Kesner 1997) relies on properties of Girard's proof nets for linear logic. Ritter (Ritter 1999) sketches a generic proof of PSN for a general family of systems of explicit substitutions. Bonelli (Bonelli 2001) proposes a proof of strong normalization for $\lambda\mathtt{x}_{gc}$ inspired by the proof of (Benaissa, Briaud, Lescanne & Rouyer-Degli 1996) and adapted to a calculus with names. His proof features a delicate combinatorial argument exploring the connection between the notion of minimal counter-example and the notion of perpetual reduction (van Raamsdonk, Severi, Sørensen & Xi 1999) and he characterizes SN-terms inductively. From this, he deduces a proof of SN for the terms typable in a system of polymorphic types. Herbelin (Herbelin 2001) explores reduction and strategies and like us, he uses reducibility. As mentioned above, Melliès (Melliès 1995) showed that $\lambda\sigma$ does not have PSN; Guillaume (Guillaume 2000) did the same for λs_e.

A subtle point of difference between the direct proofs in this paper and preservation of strong normalization results is that our results apply to all terms of the calculus, not just the pure substitution-free terms.

Our notation is consistent with that of (Barendregt 1992), to which we refer the reader for background on the classical λ-calculus.

2. Terms and reduction strategies

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

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

Definition 2.1 (The basic taxonomy). The set $\Lambda\mathtt{x}$ of terms with explicit substitutions is the set of terms M defined as follows:

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

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 $FV(M)$.

In writing terms we assume that closure-formation has higher precedence than application and abstraction. So $UV\langle x = A\rangle$ is an application term and $\lambda y.B\langle x = A\rangle$ is an abstraction.

We assume Barendregt's convention (Barendregt 1984), 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 to which is applied a sequence of closures then a sequence of applications of terms. 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 2.2 (The head form taxonomy). Every term is of precisely one of the following forms:

$\lambda x.B$	$(\lambda y.B)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$
$xT_1 \cdots T_n$ with $n \geq 0$	$(UV)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$
$(\lambda x.B)AT_1 \cdots T_n$ with $n \geq 0$	$x\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$
	$y\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$ with $x \not\equiv y$

Proof. Clearly those terms are well-formed according to Definition 2.1. We see that each term M has this form by induction on the size of M. The cases when M is an abstraction or a variable are clearly covered by the first two forms above. If M is an application, M is of the form $N\overrightarrow{T}$ where N is either a variable or an abstraction or a closure. The first two cases are covered by the second and third forms; when N is a closure it is, by induction, of one of the forms in the right-hand column with $\boldsymbol{T}$ empty; so we obtain M by taking $\boldsymbol{T}$ to be $\overrightarrow{T}$. If M is a closure, we may write M as $P\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$ where P is either a variable, an abstraction, or an application, just as prescribed in the right-hand column. □

Following (Barendregt 1984) 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 (Benaissa, Lescanne & Rose 1996). 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 (Bloo & Rose 1995, Rose 1996, Bloo 1997). The rules Varl 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 2.3. The notions of reduction $\lambda\mathsf{x}$ and $\lambda\mathsf{x}_{gc}$ are subsets of the rules in Table 1: the notion of reduction $\lambda\mathsf{x}$ is obtained by deleting the rule gc, and the notion of reduction $\lambda\mathsf{x}_{gc}$ is obtained by deleting the rule VarK.

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

Of course in the presence of rule gc we do not need rule VarK. 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,

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

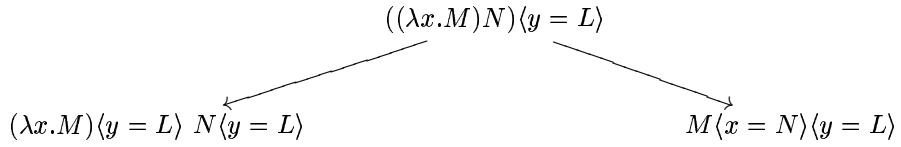
Table 1. *The reduction rules.*

atomic, substitution operations. On the other hand with an appropriate data structure for maintaining (the free variables in) terms gc 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 (a prime example is Lemma 3.2 below).

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 "reduction" refers to reduction in system $\lambda\mathtt{x}_{gc}$. At the end of the paper (Section 8) we will be able to show that the results for the system not including garbage collection follow readily from the results for $\lambda\mathtt{x}_{gc}$.

As usual, a *normal form* is a term to which no reduction may be applied. These are the terms with no B-redexes and no occurrence of a substitution. 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 $A_i \in \Lambda\mathrm{x}$.

The head and leftmost redexes from the classical λ-calculus appear in $\lambda\mathtt{x}_{gc}$ as head or leftmost B-redexes. But the general notions of head or leftmost redex in $\lambda\mathtt{x}_{gc}$ must take into account the rules for applying substitutions. In fact the correct definitions of head and leftmost reduction are more subtle than in the classical calculus. Essentially this is because $\lambda\mathtt{x}_{gc}$ has a critical pair, namely:



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

Definition 2.4 (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 2.5 (Head reduction). *Head reduction* is the closure of $\lambda\mathtt{x}_{gc}$ under the structural rules of Table 2.

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

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

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

Table 2. *Head reduction*

Definition 2.6 (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 $\mathcal{LN}$.

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

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

Table 3. *Leftmost reduction*

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 T be $((\lambda x.M)N)\langle y = L\rangle$. Then T can rewrite by leftmost reduction either to $P \equiv M\langle x = N\rangle\langle y = L\rangle$, or (in two steps) to $Q \equiv ((\lambda x.M\langle y = L\rangle)\ N\langle y = L\rangle)$. Then since $\lambda x.M\langle y = L\rangle$ is an abstraction Q leftmost-rewrites via rule B leading to $Q' \equiv M\langle y = L\rangle\langle x = \ N\langle y = L\rangle\rangle$.

3. Two fundamental Lemmas

We have pointed out that the combinatorics of reduction in $\lambda \mathtt{x}_{gc}$ is more complex than that of classical λ-calculus. In this section we isolate two key lemmas to handle these complications. In that sense this section is the technical heart of the paper. The two lemmas aim at proving the following two facts (where $\mathcal{N}$ stands for $\mathcal{SN}$, $\mathcal{LN}$, or $\mathcal{HN}$).

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

and

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

where in the case that $\mathcal{N}$ is $\mathcal{SN}$we require $A \in \mathcal{SN}$ as well.

Remark (on the classical Substitution Lemma). The Substitution Lemma of the classical λ-calculus (Barendregt 1984) 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 immediately above can have quite different reduction behaviors. 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 $L \equiv \lambda u.uu$.

We will show next 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. *A lemma on composition*

Let us consider the following rule

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

which we call *composition*. It abstracts the composition one finds in systems like $\lambda\sigma$ (Abadi et al. 1991, Curien, Hardin & Lévy 1996) (namely the rule called *Map*) or in the extension $\lambda\mathtt{x}\|c$ of $\lambda\mathtt{x}$ (Bloo 1997, Bloo & Geuvers 1999, Kamareddine & Ríos 2000, Kamareddine & Nederpelt 1993) (namely the rule $\xrightarrow[\|c]{}$ of (Bloo 1997), see also (Rose 1996) page 75).

We want to prove that the converse of the composition rule preserves (strong, head, or leftmost) normalization. A natural first attempt would be to try to show that instances of this rule can be permuted with the rules of $\lambda\mathtt{x}_{gc}$. 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 3.1. The relation $\Rightarrow$ is given by the inductive definition indicated in Table 4.

Lemma 3.2. 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 $M' \overset{+}{\longrightarrow} M^*$

[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* $\Rightarrow$

with at least one B-step.

$$\begin{array}{ccc} M & \Longrightarrow & M' \\ \downarrow & & \vdots \\ M'' & \cdots\Rightarrow & M^* \end{array}$$

Proof. Let us start with some easy facts about this relation. Suppose $M \Rightarrow M'$. Then
— $FV(M') \subseteq FV(M)$.
— If M is a variable then $M' \equiv M$.
— If M is M_1M_2 then M' is of the form $M_1'M_2'$ with (each) $M_i \Rightarrow M_i'$.
— If M is $\lambda x.B$ then M' is of the form $\lambda x.B'$ with $B \Rightarrow B'$.
Each of these is readily proved by induction over the definition of $\Rightarrow$.

Now the proof of the Lemma is by induction over the definition of $M \Rightarrow M'$. We look at each case of the definition in turn; in each instance there are several sub-cases depending on the nature of the reduction $M \longrightarrow M''$ (we suppose first that it is a plain reduction and we add comments on what happens if it is a head or leftmost reduction).

$M \Rightarrow M'$ is Cref:
Trivial.

$M \Rightarrow M'$ is Cabs:
We have

$$\begin{array}{ccc} \lambda x.B & \Longrightarrow & \lambda x.B' \\ \downarrow & & \\ \lambda x.B'' & & \end{array}$$

(note that M'' must be of this form) with

$$\begin{array}{ccc} B & \Longrightarrow & B' \\ \downarrow & & \\ B'' & & \end{array}$$

The induction hypothesis yields a B^* closing this diagram and we have

$$\begin{array}{ccc} \lambda x.B & \Longrightarrow & \lambda x.B' \\ \downarrow & & \vdots \\ \lambda x.B'' & \dashrightarrow & \lambda x.B^* \end{array}$$

as desired. Clearly if $\lambda x.B \longrightarrow \lambda x.B''$ is a head [leftmost] reduction then $B \longrightarrow B''$ is a head [leftmost] reduction, so that by induction $B' \longrightarrow B^*$ is a head [leftmost] reduction and $\lambda x.B' \longrightarrow \lambda x.B^*$ is a head [leftmost] reduction.

$M \Rightarrow M'$ is Capp:
There are two sub-cases. The first is

$$\begin{array}{ccc} UV & \Longrightarrow & U'V' \\ \downarrow & & \\ U''V'' & & \end{array}$$

With two applications of the induction hypothesis we obtain U^* and V^* and the term we need is U^*V^*. It is easy to see that this construction preserves the property of being a head [leftmost] reduction.

The other case is

$$\begin{array}{ccc} (\lambda x.B)A & \Longrightarrow & (\lambda x.B')A' \\ \downarrow & & \\ B\langle x = A\rangle & & \end{array}$$

Note that M' must be of that form.

We may take M^* to be $B'\langle x = A'\rangle$. Recall that the theorem requires that we do a B-step from M' to M^*. Clearly both reductions (from M to M'' and from M' to M^*) are head reductions.

$M \Rightarrow M'$ is Cclo:
Here we have

$$\begin{array}{ccc} B\langle z = A\rangle & \Longrightarrow & B'\langle z = A'\rangle \\ \downarrow & & \\ M' & & \end{array}$$

and there are cases depending on the reduction ρ from $B\langle z = A\rangle$ to M'. If ρ is internal to B (a possible head [leftmost] reduction) or to A (not a head [leftmost] reduction) then we build M^* by an easy application of the induction hypothesis and preserve head [leftmost] reductions. Otherwise ρ is at the root; it is either Varl, gc, App, or Abs.

If ρ is Varl then M' is A, B' is z, and we may take M^* to be A'.

If ρ is gc then M' is B, and since $B \Rightarrow B'$ we know that z is not free in B'. So we may apply gc and take M^* to be B'.

If ρ is App then we have

$$\begin{array}{ccc} (UV)\langle z=A\rangle & \Longrightarrow & (U'V')\langle z=A'\rangle \\ \downarrow & & \\ U\langle z=A\rangle V\langle z=A\rangle & & \end{array}$$

Then

$$\begin{array}{ccc} (UV)\langle z=A\rangle & \Longrightarrow & (U'V')\langle z=A'\rangle \\ \downarrow & & \vdots \\ U\langle z=A\rangle V\langle z=A\rangle & \cdots\Rightarrow & U'\langle z=A'\rangle V'\langle z=A'\rangle \end{array}$$

If ρ is Abs the argument is similar. In each case, if the given reduction is a head [leftmost] reduction, then the reduction constructed is a head [leftmost] reduction.

$M \Rightarrow M'$ is Cpabs:
Here

$$\begin{array}{ccc} (\lambda x.B)\langle y=Q\rangle & \Longrightarrow & \lambda x.B^+ \quad \text{with} \quad B\langle y=Q\rangle \Rightarrow B^+ \\ \downarrow & & \\ M'' & & \end{array}$$

There are three cases: M'' can be obtained by reduction inside B, inside Q, or by an application of Abs at the root.

If we have $B \longrightarrow B''$ then we may obtain B^* by induction, completing the following diagram:

$$\begin{array}{ccc} B\langle y=Q\rangle & \Longrightarrow & B^+ \\ \downarrow & & \\ B''\langle y=Q\rangle & & \end{array}$$

and so

$$\begin{array}{ccc} (\lambda x.B)\langle y=Q\rangle & \Longrightarrow & \lambda x.B^+ \\ \downarrow & & \vdots \\ (\lambda x.B'')\langle y=Q\rangle & \cdots\Rightarrow & \lambda x.B^* \end{array} .$$

Similarly if we have $Q \longrightarrow Q''$ then we obtain B^* by induction, completing the following diagram:

$$\begin{array}{ccc} B\langle y=Q\rangle & \Longrightarrow & B^+ \\ \downarrow & & \\ B\langle y=Q''\rangle & & \end{array}$$

and so

$$\begin{array}{ccc} (\lambda x.B)\langle y = Q\rangle & \Longrightarrow & \lambda x.B^+ \\ \downarrow & & \vdots \\ (\lambda x.B)\langle y = Q''\rangle & \cdots\Rightarrow & \lambda x.B^* \end{array}$$

Finally, if M'' is $\lambda x.B\langle y = Q\rangle$ then

$$\begin{array}{ccc} (\lambda x.B)\langle y = Q\rangle & \Longrightarrow & \lambda x.B^+ \\ \downarrow & & \vdots \\ \lambda x.B\langle y = Q\rangle & \cdots\Rightarrow & \lambda x.B^+ \end{array}$$

The empty reduction $\lambda x.B^+ \; -\!-\!\twoheadrightarrow \; \lambda x.B^+$ is trivially a sequence of head reductions.

$M \Rightarrow M'$ is Cpapp:
Here

$$\begin{array}{ccc} (UV)\langle y = Q\rangle & \Longrightarrow & U^+V^+ \quad \text{with} \quad U\langle y = Q\rangle \Rightarrow U^+ \quad \text{and} \quad V\langle y = Q\rangle \Rightarrow V^+ \\ \downarrow & & \\ M'' & & \end{array}$$

The reduction ρ can be inside V or Q (not a head reduction), or inside U, or it can be a B-step at the root of (UV), or it can be a root App.

If ρ takes place inside U then $M'' \equiv (U''V)\langle y = Q\rangle$, and by induction hypothesis there is a U^* with $U''\langle y = Q\rangle \Rightarrow U^*$. It is easy to check that $M'' \equiv (U''V)\langle y = Q\rangle \Rightarrow U^*V^+$. Furthermore U^+V^+ reduces to U^*V^+ and if ρ is a head [leftmost] reduction then so is this one.

If ρ takes place inside V the argument is similar, and there is nothing to check as regards head [leftmost] reductions.

If ρ is a root App-reduction then, as is easily verified, we may take U^*V^* to be U^+V^+.

The B-step case itself has two sub-cases, since we have $U \equiv \lambda x.B$ and so $(\lambda x.B)\langle y = Q\rangle \Rightarrow U^+$ can be either a Cclo step or a Cpabs step. The first sub-case is:

$$\begin{array}{ccc} ((\lambda x.B)V)\langle y = Q\rangle & \Longrightarrow & ((\lambda x.B')\langle y = Q'\rangle)V^+ \quad \text{with} \quad B \Rightarrow B' \quad \text{and} \quad Q \Rightarrow Q' \\ \downarrow & & \\ B\langle x = V\rangle\langle y = Q\rangle & & \end{array}$$

Complete the square by:

$$\begin{array}{ccc} ((\lambda x.B)V)\langle y = Q\rangle & \Longrightarrow & ((\lambda x.B')\langle y = Q'\rangle)V^+ \\ \downarrow & & \vdots \\ B\langle x = V\rangle\langle y = Q\rangle & \cdots\Rightarrow & B'\langle y = Q'\rangle\langle x = V^+\rangle \end{array}$$

The bottom $\Rightarrow$ reduction is a Cpclo-step; to verify this it suffices to verify that

$$B\langle y = Q\rangle \Rightarrow B'\langle y = Q'\rangle \quad \text{and} \quad V\langle y = Q\rangle \Rightarrow V^+.$$

The reduction $((\lambda x.B')\langle y = Q'\rangle)V^+ \; -\!-\!\twoheadrightarrow \; B'\langle y = Q'\rangle\langle x = V^+\rangle$ is an Abs followed by a B, both head reductions.

The second sub-case is:

$$\begin{array}{ccc} ((\lambda x.B)V)\langle y = Q\rangle & \Longrightarrow & (\lambda x.B^+)V^+ \quad \text{with} \quad B\langle y = Q\rangle \Rightarrow B^+ \\ \downarrow & & \\ B\langle x = V\rangle\langle y = Q\rangle & & \end{array}$$

Complete the square by:

$$\begin{array}{ccc} ((\lambda x.B)V)\langle y = Q\rangle & \Longrightarrow & (\lambda x.B^+)V^+ \\ \downarrow & & \vdots \\ B\langle x = V\rangle\langle y = Q\rangle & \Rightarrow\!\!\!\Rightarrow & B^+\langle x = V^+\rangle \end{array}$$

The bottom $\Rightarrow$ reduction is a Cpclo-step. The reduction $-\!-\!\twoheadrightarrow$ is a head reduction.

$M \Rightarrow M'$ is Cpclo:
Here we have

$$\begin{array}{ccc} B\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & B^+\langle x = P^+\rangle \quad \text{with} \quad \begin{array}{l} B\langle y = Q\rangle \Rightarrow B^+ \quad \text{and} \\ P\langle y = Q\rangle \Rightarrow P^+. \end{array} \\ \downarrow & & \\ M'' & & \end{array}$$

The reduction ρ from M to M'' could be internal to any of B, P, or Q (handled as in earlier cases), or it could be a Varl, gc, Abs, or App at the root of the term $B\langle x = P\rangle$ (head reductions).

When ρ is Varl: First note that, since $x\langle y = Q\rangle \Rightarrow B^+$, then $B^+ \equiv x\langle y = Q^+\rangle$. So we have the following square

$$\begin{array}{ccc} x\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & x\langle y = Q^+\rangle\langle x = P^+\rangle \\ \downarrow & & \vdots \\ P\langle y = Q\rangle & \Rightarrow\!\!\!\Rightarrow & P^+ \end{array}$$

via $B^+\langle x = P^+\rangle \longrightarrow\!\!\!\!\twoheadrightarrow x\langle x = P^+\rangle \longrightarrow P^+$ (head reductions).

When ρ is gc: Here x is not free in B; recall that x is assumed to be not free in P. It follows that x is not free in $B^+\langle x = P^+\rangle$. Then

$$\begin{array}{ccc} B\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & B^+\langle x = P^+\rangle \\ \downarrow & & \vdots \\ B\langle y = Q\rangle & \Rightarrow\!\!\!\Rightarrow & B^+ \end{array}$$

where $B^+\langle x = P^+\rangle \longrightarrow B^+$ by (head) garbage-collection.

When ρ is Abs:

$$\begin{array}{l}(\lambda z.T)\langle x = P\rangle\langle y = Q\rangle \Longrightarrow B^+\langle x = P^+\rangle \\ \quad\Big\downarrow \\ (\lambda z.T\langle x = P\rangle)\langle y = Q\rangle\end{array} \quad \text{with} \quad \begin{array}{l}(\lambda z.T)\langle y = Q\rangle \Rightarrow B^+ \quad \text{and} \\ P\langle y = Q\rangle \Rightarrow P^+.\end{array}$$

Now B^+ can be obtained from $(\lambda z.T)\langle y = Q\rangle$ either by Cclo or by Cpabs. In the first sub-sub-case we have, for certain $T \Rightarrow T'$ and $Q \Rightarrow Q'$:

$$\begin{array}{ccc}(\lambda z.T)\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & (\lambda z.T')\langle y = Q'\rangle\langle x = P^+\rangle \\ \Big\downarrow & & \Big\vdots \\ (\lambda z.T\langle x = P\rangle)\langle y = Q\rangle & \cdots\Rightarrow & \lambda z.T'\langle y = Q'\rangle\langle x = P^+\rangle\end{array}$$

We may verify $(\lambda z.T\langle x = P\rangle)\langle y = Q\rangle \Rightarrow \lambda z.T'\langle y = Q'\rangle\langle x = P^+\rangle$ as an instance of Cpabs by checking that $T\langle x = P\rangle\langle y = Q\rangle \Rightarrow T'\langle y = Q'\rangle\langle x = P^+\rangle$, which is itself an instance of Cpclo. $--\twoheadrightarrow$ are head reductions.

In the second sub-sub-case we have a T^+ with $T\langle y = Q\rangle \Rightarrow T^+$ and

$$\begin{array}{ccc}(\lambda z.T)\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & (\lambda z.T^+)\langle x = P^+\rangle \\ \Big\downarrow & & \Big\vdots \\ (\lambda z.T\langle x = P\rangle)\langle y = Q\rangle & \cdots\Rightarrow & \lambda z.T^+\langle x = P^+\rangle\end{array}$$

The upper Cpclo-step is based on the fact that $(\lambda z.T)\langle y = Q\rangle \Rightarrow \lambda z.T^+$ which is itself an instance of Cpabs. The lower $\Rightarrow$ is an instance of Cpabs, justified by the fact that $T\langle x = P\rangle\langle y = Q\rangle \Rightarrow T^+\langle x = P^+\rangle$, the latter by Cpclo. $(\lambda x.T^+)\langle x = P^+\rangle \; --\twoheadrightarrow \; \lambda z.T^+\langle x = P^+\rangle$ is an Abs head reduction.

When ρ is App:

$$\begin{array}{l}(UV)\langle x = P\rangle\langle y = Q\rangle \Longrightarrow B^+\langle x = P^+\rangle \\ \quad\Big\downarrow \\ ((U\langle x = P\rangle)(V\langle x = P\rangle))\langle y = Q\rangle\end{array} \quad \text{with} \quad \begin{array}{l}(UV)\langle y = Q\rangle \Rightarrow B^+ \quad \text{and} \\ P\langle y = Q\rangle \Rightarrow P^+.\end{array}$$

As in previous cases we consider the two ways in which $(UV)\langle y = Q\rangle \Rightarrow B^+$:the possibilities are Cclo and Cpapp.

In the first sub-sub-case we have U', V', and Q' with $(UV)\langle y = Q\rangle \Rightarrow (U'V')\langle y = Q'\rangle$ and then

$$\begin{array}{ccc}(UV)\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & (U'V')\langle y = Q'\rangle\langle x = P^+\rangle \\ \Big\downarrow & & \Big\vdots \\ (U\langle x = P\rangle V\langle x = P\rangle)\langle y = Q\rangle & \cdots\Rightarrow & (U'\langle y = Q'\rangle\langle x = P^+\rangle)(V'\langle y = Q'\rangle\langle x = P^+\rangle)\end{array}$$

The bottom $\Rightarrow$ is an instance of Cpapp which holds since $U\langle x = P\rangle\langle y = Q\rangle \Rightarrow U'\langle y = Q'\rangle\langle x = P^+\rangle$, which is an instance of Cpclo; and similarly for V.

When $(UV)\langle y = Q\rangle \Rightarrow B^+$ by Cpapp we have

$$\begin{array}{ccc} (UV)\langle x = P\rangle\langle y = Q\rangle & \Longrightarrow & (U^+V^+)\langle x = P^+\rangle \\ \downarrow & & \vdots \\ (U\langle x = P\rangle V\langle x = P\rangle)\langle y = Q\rangle & \Rightarrow & (U^+\langle x = P^+\rangle)(V^+\langle x = P^+\rangle) \end{array}$$

The bottom $\Rightarrow$ is an instance of Cpapp since $U\langle x = P\rangle\langle y = Q\rangle \Rightarrow U^+\langle x = P^+\rangle$ and $V\langle x = P\rangle\langle y = Q\rangle \Rightarrow V^+\langle x = P^+\rangle$. In both cases the diagram is closed by head reductions (one App or two App's). □

Corollary 3.3. 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 λx_{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 3.2. □

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. *A lemma on garbage-collection*

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

Definition 3.4. An 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 3.5. 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. The structure of the proof for each of the three statements is the same, so let us introduce notation to avoid repetition of the argument. We let $\mathcal{N}$ stand for any of $\mathcal{SN}$, $\mathcal{LN}$, or $\mathcal{HN}$, where $\longrightarrow$ (the unrestricted reduction), $\xrightarrow{l}$ and $\xrightarrow{h}$ are the corresponding reductions.

We consider triples $(D, \boldsymbol{M}, \boldsymbol{A})$ where D is a term, $\boldsymbol{M}$ and $\boldsymbol{A}$ are multisets of terms. Let $\sqsupset^m$ be the multiset extension (Dershowitz & Manna 1979) of $\sqsupset$, the converse of the proper subterm order, and let $\longrightarrow^m$ be the multiset extension of the reduction relation.The proof is by induction over the following relation: $(D, \boldsymbol{M}, \boldsymbol{A}) \gg (D', \boldsymbol{M}', \boldsymbol{A}')$ if and only if

1 $D \longrightarrow D'$, or
2 $D = D'$ and $\boldsymbol{M} \sqsupset^m \boldsymbol{M}'$, or
3 (when unrestricted reduction is considered) $D = D'$, $\boldsymbol{M} = \boldsymbol{M}'$, and $\boldsymbol{A} \longrightarrow^m \boldsymbol{A}'$.

In what follows, D will be $C[\![M_1, ..., M_n]\!]$ and $\longrightarrow$ (respectively $\xrightarrow{l}$ and $\xrightarrow{h}$) will be well-founded out of D by hypothesis; $\boldsymbol{M}$ will be $\{M_1, ..., M_n\}$; $\boldsymbol{A}$ will be $\{A_1, ..., A_n\}$ and its $\lambda \mathtt{x}_{gc}$-reducts. The relation $\longrightarrow^m$ will be well-founded since multiset extension preserves well-foundedness. Therefore $\gg$ is well-founded and a noetherian induction on $\gg$ is possible. A remark on cases 4 and 5 below: in these cases the term D does not change, only its representation as $C[\![\ldots]\!]$ does. This means we insert the A_i's at "lower" positions, allowing us to perform a noetherian induction.

Assume the induction hypothesis and that $C[\![M_1, ..., M_n]\!] \in \mathcal{N}$ (and that $A_i \in \mathcal{SN}$ for $1 \leq i \leq n$, when we consider unrestricted reduction). Let us prove that $C[\![M_1\langle x = A_1\rangle, ..., M_n\langle x = A_n\rangle]\!]$ reduces only to terms that are in $\mathcal{N}$.

1 $C[\![M_1\langle x = A_1\rangle, ..., M_n\langle x = A_n\rangle]\!] \longrightarrow C'[\![M_{i_1}\langle x = A_{i_1}\rangle, ..., M_{i_p}\langle x = A_{i_p}\rangle]\!]$ (where the $i_j \in [1..n]$), then

$$C[\![M_1, ..., M_n]\!] \longrightarrow C'[\![M_{i_1}, ..., M_{i_p}]\!],$$

and by induction $C'[\![M_{i_1}\langle x = A_{i_1}\rangle, ..., M_{i_p}\langle x = A_{i_p}\rangle]\!] \in \mathcal{N}$.

2 $M_i \longrightarrow M_i'$, works also by induction.

3 $A_j \longrightarrow A_j'$, works also by induction. Note that this case occurs only when we consider unrestricted reduction, so that the A_i are assumed to be in $\mathcal{SN}$.

4 $M_i = M_i^1 M_i^2$ and $M_i\langle x = A_i\rangle \longrightarrow M_i^1\langle x = A_i\rangle M_i^2\langle x = A_i\rangle$.

$$\{M_1, ..., M_i, ..., M_n\} \sqsupset^m \{M_1, ..., M_i^1, M_i^2, ..., M_n\},$$

hence $C[\![M_1\langle x = A_1\rangle, ..., M_i^1\langle x = A_i\rangle M_i^2\langle x = A_i\rangle, ..., M_n\langle x = A_n\rangle]\!] \in \mathcal{N}$ by induction.

5 $M_i = \lambda y.M_i'$ and $M_i\langle x = A_i\rangle \longrightarrow \lambda y.(M_i'\langle x = A_i\rangle)$.

$$\{M_1, ..., M_i, ..., M_n\} \sqsupset^m \{M_1, ..., M_i', ..., M_n\},$$

hence $C[\![M_1\langle x = A_1\rangle, ..., \lambda y.(M_i'\langle x = A_i\rangle), ..., M_n\langle x = A_n\rangle]\!] \in \mathcal{N}$ by induction.

6 $M_i\langle x = A_i\rangle \longrightarrow M_i$, which is always applicable since $x \notin FV(M_i)$.

$$\{M_1, ..., M_i, ..., M_n\} \sqsupset^m \{M_1, ..., \ , ..., M_n\},$$

hence $C[\![M_1\langle x = A_1\rangle, ..., M_i, ..., M_n\langle x = A_n\rangle]\!] \in \mathcal{N}$ by induction.

□

Corollary 3.6. 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

In most proofs of normalization results one finds it convenient to isolate some "perpetuality" results, such as "if a term with a head-redex has an infinite reduction, then the result of contracting that redex also has an infinite reduction," or in other words, "the set of strongly normalizing terms is closed under head-expansion." The notion of saturated set — present in many SN-proofs, see for example (Girard 1971, Girard 1972, Tait 1975, Mitchell 1986) — captures such closure conditions. The following defines a notion of saturation appropriate to $\lambda \mathtt{x}_{gc}$ reduction.

Definition 4.1. 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.

sat-B	$\dfrac{B\langle x = A\rangle\, \boldsymbol{T}}{(\lambda x.B)A\, \boldsymbol{T}}$	sat-I	$\dfrac{A\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}{x\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}$
sat-Abs	$\dfrac{(\lambda y.B\langle x = A\rangle)\,\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}{(\lambda y.B)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}$	sat-App	$\dfrac{(U\langle x = A\rangle)(V\langle x = A\rangle)\,\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}{(UV)\langle x = A\rangle\,\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}$
sat-comp	$\dfrac{M\langle y = Q\rangle\langle x = P\langle y = Q\rangle\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}{M\langle x = P\rangle\langle y = Q\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}$	sat-gc	$\dfrac{N\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T} \qquad A \in \mathcal{X}, \quad x \notin FV(N)}{N\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}$

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.

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 4.2.
— $\mathcal{HN}$ is $\Lambda\mathtt{x}$-saturated.
— $\mathcal{LN}$ is $\Lambda\mathtt{x}$-saturated.
— $\mathcal{SN}$ is $\mathcal{SN}$-saturated.

Proof. We wish to show that each of $\mathcal{HN}$, $\mathcal{LN}$, and $\mathcal{SN}$ is closed under the rules in Definition 4.1.

It suffices to show that if M fails to be in $\mathcal{HN}$, $\mathcal{LN}$, or $\mathcal{SN}$ respectively then any term which is the hypothesis of a rule yielding M also fails to be in this set.

For all of the inference rules except sat-comp, sat-gc, and sat-App this is easy to see. Corollary 3.3 directly handles the case of sat-comp. Corollary 3.6 implies closure under sat-gc. The case of sat-App is subtle precisely because of the critical pair we identified earlier: the term UV might itself be a B-redex, and it is conceivable that pushing the substitution through this B-redex might spoil the existence of an infinite reduction out of M. But in fact this is precisely what Lemma 3.2 precludes. □

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 in Lemma 3.5.

5. Types

Definition 5.1. Given an infinite set of type-variables and a 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; such a judgment is *derivable*, denoted $\Gamma \vdash M : \tau$, if this form can be inferred from the rules 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.

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

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

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

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

Table 6. *Typing rules*

Remark. 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 5.2. Let Γ_1 and Γ_2 be bases. The basis $\Gamma_1 \sqcap \Gamma_2$ contains $x\colon \sigma$ if either:
— $(x\colon \sigma)$ is in Γ_1 and $x \notin \mathcal{D}om(\Gamma_2)$, or
— $(x\colon \sigma)$ is in Γ_2 and $x \notin \mathcal{D}om(\Gamma_1)$, or
— $\sigma = \sigma_1 \cap \sigma_2$ with $(x\colon \sigma_1) \in \Gamma_1$ and $(x\colon \sigma_2) \in \Gamma_2$.

The following lemma collects some technical properties of the type system.

Lemma 5.3. In each of $\mathcal{D}$ and $\mathcal{D}_\omega$:
1 If $\Gamma \vdash M\colon \tau$ then for all Γ', $\Gamma \sqcap \Gamma' \vdash M\colon \tau$.
2 If $\Gamma \vdash M\colon \tau$ and Γ' agrees with Γ on the free variables of M then $\Gamma' \vdash M\colon \tau$.
3 If $\Gamma \vdash M\langle x = S\rangle\colon \tau$ then there is a σ such that $\Gamma, (x\colon \sigma) \vdash M\colon \tau$ and $\Gamma \vdash S\colon \sigma$
4 If $\Gamma \vdash \lambda x.B\colon \sigma \to \tau$ then $\Gamma, (x\colon \sigma) \vdash B\colon \tau$.

Proof. Parts 1 and 2 are routine inductions.

For part 3 we induct on the length of the derivation of $\Gamma \vdash M\langle x = S\rangle\colon \tau$. If the last inference is an instance of the cut rule the result is immediate. When the last inference is an instance of $\cap$-E we have, for some τ', $\Gamma \vdash M\langle x = S\rangle\colon (\tau \cap \tau')$; by induction there is a σ with $\Gamma \vdash S\colon \sigma$ and $\Gamma, (x\colon \sigma) \vdash M\colon (\tau \cap \tau')$, so clearly $\Gamma, (x\colon \sigma) \vdash M\colon \tau$ and we are done. Finally suppose the last inference is an instance of $\cap$-I. Then τ is $\tau_1 \cap \tau_2$ and for $1 \le i \le 2$ we have $\Gamma \vdash M\langle x = S\rangle\colon \tau_i$. By induction there are σ_i such that $\Gamma, (x\colon \sigma_i) \vdash M\colon \tau_i$ and $\Gamma \vdash S\colon \sigma_i$. So $\Gamma \vdash S\colon \sigma_1 \cap \sigma_2$. And by part 1, $\Gamma, (x\colon \sigma_1 \cap \sigma_2) \vdash M\colon \tau_i$ for each i, so that $\Gamma, (x\colon \sigma_1 \cap \sigma_2) \vdash M\colon \tau$ as desired.

For part 4 it suffices to prove the following generalization: *For all* $\gamma_1, \ldots, \gamma_k$, *if* $\Gamma \vdash \lambda x.B\colon (\sigma \to \tau) \cap \gamma_1 \cap \cdots \cap \gamma_k$ *then* $\Gamma, (x\colon \sigma) \vdash B\colon \tau$. We prove this by induction on typing derivations. If the last inference is an instance of $\to$-E then $k = 0$ and the result is immediate. If the last inference is an instance of $\cap$-E we apply the induction hypothesis to the sole premise of the inference. If the last inference is an instance of $\cap$-I the induction hypothesis applies to one of the premises.

□

6. Normalization for typable terms

In this section we demonstrate that, as in the classical λ-calculus, type assignments for Λx terms guarantee certain reduction properties. The framework is a standard one, interpreting terms in a semantics which uses the notion of saturated set in an essential way: see (Tait 1967, Girard 1972).

Definition 6.1 (Function space). If $\mathcal{A}$ and $\mathcal{B}$ are sets of terms then $\mathcal{A} \twoheadrightarrow \mathcal{B}$ is $\{F \mid \forall A \in \mathcal{A}, (FA) \in \mathcal{B}\}$.

Definition 6.2. An interpretation $\mathcal{I}$ is a function from types to sets of terms obeying the following
— $\mathcal{I}_\omega = \Lambda$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. The following are very easy consequences of the definition of saturation.

Lemma 6.3. 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 $\mathcal{A} \cap \mathcal{B}$.

Corollary 6.4. 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.

Proof. Immediate from Lemma 6.3 and the fact (when $\tau = \omega$) that Λx is itself $\mathcal{X}$-saturated. □

As with any well-behaved type system, types are preserved by reduction. This plays an essential role in the Soundness Theorem below.

Theorem 6.5 (Subject Reduction). In either of the systems $\mathcal{D}_\omega$ or $\mathcal{D}$: suppose $\Gamma \vdash M : \tau$ and $M \longrightarrow M_1$. Then $\Gamma \vdash M_1 : \tau$.

Proof. By induction over the derivation of $\Gamma \vdash M : \tau$. If the last inference of this derivation is ω-introduction the result is immediate; if the last inference is one of $\cap$-introduction or $\cap$-elimination an easy application of the induction hypothesis suffices. We organize the rest of the argument by cases according to the shape of the term M.

If M is an abstraction $\lambda x.B$ then M_1 is of the form $\lambda x.B_1$ with $B \longrightarrow B_1$, and we need only consider the situation which the typing of M ends with $\rightarrow$introduction. Then $\tau = (\tau_1 \rightarrow \tau_2)$ and $\Gamma, y : \tau_1 \vdash B : \tau_2$. The induction hypothesis yields the same typing for B_1 and then we may type M_1 as desired.

In case M is FA and was typed by

$$\rightarrow\mathrm{E} \quad \frac{\Gamma \vdash F : \sigma \rightarrow \tau \qquad \Gamma \vdash A : \sigma}{\Gamma \vdash (FA) : \tau}$$

there are three cases for the reduction to M_1. If M_1 is $F_1 A$ with $F \longrightarrow F_1$ or M_1 is FA_1 with $A \longrightarrow A_1$ then we may apply the induction hypothesis to the typing of F or A as appropriate. So suppose M is $(\lambda x.B)A$ and M_1 is $B\langle x = A\rangle$. We have $\Gamma \vdash \lambda x.B : \sigma \rightarrow \tau$ and $\Gamma \vdash A : \sigma$; by Lemma 5.3.4 we have $\Gamma, (x : \sigma) \vdash B : \tau$, and so $\Gamma \vdash B\langle x = A\rangle : \tau$ by a cut.

The remaining case is when M is a closure, typed by the cut rule:

$$\frac{\Gamma, (x : \sigma) \vdash P : \tau \qquad \Gamma \vdash Q : \sigma}{\Gamma \vdash P\langle x = Q\rangle : \tau}$$

If M_1 is obtained by a reduction inside of P or Q then as usual we use a simple application of the induction hypothesis. The interesting sub-cases are when M_1 is produced by a VarK or gc, Varl, Abs, or App reduction. Of course the gc sub-case subsumes that of VarK; we treat this case first.

When $P\langle x = Q\rangle \longrightarrow P$: Since x is not free in P we may apply Lemma 5.3.2 to the judgement $\Gamma, (x : \sigma) \vdash P : \tau$.

When $x\langle x = Q\rangle \longrightarrow Q$: The derivation $\Gamma, (x : \sigma) \vdash x : \tau$ is either an instance of the

start rule or has $\cap$-introduction or $\cap$-elimination as its last inference. In the first case $\sigma = \tau$ and we are done. If it is $\cap$-introduction we have $\tau = \tau_1 \cap \tau_2$, $\Gamma, (x\colon \sigma) \vdash x\colon \tau_i$ for $i = 1, 2$ and so by induction (twice) we have $\Gamma \vdash Q\colon \tau_i$ for $i = 1, 2$, whence $\Gamma \vdash Q\colon \tau$. The case of $\cap$-elimination is similar.

When $(\lambda y.C)\langle x = Q\rangle \longrightarrow \lambda y.(C\langle x = Q\rangle)$*:* The hypotheses of the cut rule are that $\Gamma, (x\colon \sigma) \vdash \lambda y.C\colon \tau$ and $\Gamma \vdash Q\colon \sigma$; we seek $\Gamma \vdash \lambda y.(C\langle x = Q\rangle)\colon \tau$. If the last inference in $\Gamma, (x\colon \sigma) \vdash \lambda y.C\colon \tau$ is $\cap$-introduction or $\cap$-elimination then it is straightforward to rearrange the deductions just as in the previous case. Otherwise τ is $\tau_1 \to \tau_2$ and $\Gamma, (y\colon \tau_1), (x\colon \sigma) \vdash C\colon \tau_2$. Certainly $\Gamma, (y\colon \tau_1) \vdash Q\colon \sigma$, so an application of cut yields $\Gamma, (y\colon \tau_1) \vdash C\langle x = Q\rangle\colon \tau_2$. A final application of $\to$introduction finishes the argument.

When $(UV)\langle x = Q\rangle \longrightarrow (U\langle x = Q\rangle)(V\langle x = Q\rangle)$*:* We have $\Gamma, (x\colon \sigma) \vdash (UV)\colon \tau$ and we seek $\Gamma \vdash (U\langle x = Q\rangle)(V\langle x = Q\rangle)\colon \tau$. If the last inference in $\Gamma, (x\colon \sigma) \vdash (UV)\colon \tau$ is $\cap$-introduction or $\cap$-elimination then it is straightforward to rearrange the deductions using induction. Otherwise for some φ we have $\Gamma, (x\colon \sigma) \vdash U\colon \varphi \to \tau$ and $\Gamma, (x\colon \sigma) \vdash V\colon \varphi$ so that by induction $\Gamma \vdash U\langle x = Q\rangle\colon \varphi \to \tau$ and $\Gamma \vdash V\langle x = Q\rangle\colon \varphi$. □

Theorem 6.6 (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 closed and typable with type τ. Then $M \in \mathcal{I}_\tau$.

Proof. Throughout the proof we will use the fact that each $\mathcal{I}_\tau$ is $\mathcal{X}$-saturated and the fact that if M is typable with type τ then so are its reducts.

It is convenient to prove the following more general statement: *Let $\mathcal{I}$ and $\mathcal{X}$ be as in the statement of the theorem, and suppose*

— $(x_1\colon \alpha_1), (x_2\colon \alpha_2), \ldots, (x_n\colon \alpha_n) \vdash M\colon \tau$ *and*
— $A_i \in \mathcal{I}_{\alpha_i}$ *for* $1 \le i \le n$*, with* $x_{i+j} \notin FV(A_i)$ *for* $1 \le i \le n$ *and* $j \ge 0$.

Then $M\langle x_1 = A_1\rangle \ldots \langle x_n = A_n\rangle \in \mathcal{I}_\tau$.

Below, write Γ for the basis $(x_1\colon \alpha_1), (x_2\colon \alpha_2), \ldots, (x_n\colon \alpha_n)$. The proof is by induction on the derivation of $\Gamma \vdash M\colon \tau$; we organize the argument according to the last inference in the derivation.

The ω rule (when the system under consideration is $\mathcal{D}_\omega$). This case is trivial since $\mathcal{I}_\omega = \Lambda\mathrm{x}$.

The start rule. Here, for some i, $M \equiv x_i$ and $\tau = \alpha_i$. To show that $x_i\langle x_1 = A_1\rangle \ldots \langle x_n = A_n\rangle \in \mathcal{I}_\tau$ we induct on n; we identify two cases. If $i = 1$ then by saturation rule sat-I it suffices to check that $A_1\langle x_2 = A_2\rangle \cdots \langle x_n = A_n\rangle$ is in $\mathcal{I}_\tau$. But since none of the indicated x_i is free in A_1 and each $A_j \in \mathcal{I}_{\alpha_j} \subseteq \mathcal{X}$, we may repeat application of sat-gc and reduce the assertion to the claim that $A_1 \in \mathcal{I}_\tau$, which holds by hypothesis. If $i \neq 1$ then after a single application of sat-gc it suffices to check $x_i\langle x_2 = A_2\rangle \cdots \langle x_n = A_n\rangle$, which submits to the sub-induction hypothesis.

The rules $\cap$-I and $\cap$-E. Each of these is a very easy application of the induction hypothesis.

The $\to$E rule. We have

$$\frac{\Gamma \vdash U : \alpha \to \beta \qquad \Gamma \vdash V : \alpha}{\Gamma \vdash UV : \beta}$$

To show that $(UV)\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_\beta$ it suffices, by iterating rule sat-App, to argue that $(U\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle)(V\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle) \in \mathcal{I}_\beta$.

But by induction $(U\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle) \in \mathcal{I}_{\alpha\to\beta}$ and $(V\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle) \in \mathcal{I}_\alpha$ so the result follows by definition of $\mathcal{I}_{\alpha\to\beta}$.

The $\to$I rule. We have

$$\frac{\Gamma, (x : \alpha) \vdash B : \beta}{\Gamma \vdash \lambda x.B : \alpha \to \beta}$$

We may assume that the variable x is not free in any of the A_i. To show that $(\lambda x.B)\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_{\alpha\to\beta}$ we may iterate rule sat-Abs and argue that $\lambda x.B\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_{\alpha\to\beta}$. So choose $A \in \mathcal{I}_\alpha$, we seek $(\lambda x.B\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle)A \in \mathcal{I}_\beta$. Now by sat-B it suffices to see that $B\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle\langle x = A\rangle \in \mathcal{I}_\beta$. But (since x is not free in any A_i) this is an application of the induction hypothesis applied to the derivation of $x : \alpha, \Gamma \vdash B : \beta$.

The cut rule. Here we have

$$\frac{\Gamma, (x : \sigma) \vdash P : \tau \qquad \Gamma \vdash Q : \sigma}{\Gamma \vdash P\langle x = Q\rangle : \tau}$$

We wish to show that $P\langle x = Q\rangle\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_\tau$; we may assume without loss of generality that x is not free in any of the A_i. By iterating rule sat-comp we see that it suffices to show the following term to be in $\mathcal{I}_\tau$:

$$P\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle\langle x = Q\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle\rangle$$

By the induction hypothesis applied to the derivation $\Gamma \vdash Q : \sigma$, the term $Q\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle$ is in $\mathcal{I}_\sigma$. Then since x is not free in any of the A_i, we may use the induction hypothesis applied to the derivation $\Gamma, (x : \sigma) \vdash P : \tau$ to finish the argument. □

The following definitions are due to Cardone and Coppo (Cardone & Coppo 1990).

Definition 6.7. A type is *proper* if it has no positive occurrence of ω and *antiproper* if it has no negative 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 6.8. Let $\mathcal{H}$ be the interpretation which maps each type variable to the set $\mathcal{HN}$ of head normalizing terms.

Lemma 6.9. $\mathcal{H}_\tau \subseteq \mathcal{HN}$ for each non-trivial type τ.

Proof. We prove the following two statements simultaneously by induction on types.

1 $\mathcal{H}_\tau \subseteq \mathcal{HN}$ when τ is non-trivial
2 $(xT_1 \dots T_n) \in \mathcal{H}_\tau$ for all T_i.

At type-variables t the first claim holds by definition, and the second holds since $(xT_1 \dots T_n)$ is in head normal form. At type ω the first claim is vacuous and the second is immediate since $\mathcal{H}_\omega = \Lambda\mathrm{x}$. At intersection-types the first claim follows by induction since at least one of the conjuncts of τ must be non-trivial; the second claim is an immediate consequence of the induction hypothesis.

When τ is $\alpha \to \beta$: To establish the first claim, suppose $M \in \mathcal{H}_{\alpha\to\beta}$, and note that as a consequence of the second claim at type α, each variable is in $\mathcal{H}_\alpha$. So $Mx \in \mathcal{H}_\beta$. The type β is non-trivial, so by induction at type β, $Mx \in \mathcal{HN}$. This implies $M \in \mathcal{HN}$. The second claim follows easily from the definition of $\mathcal{H}_{\alpha\to\beta}$. □

Corollary 6.10. If M is typable in $\mathcal{D}_\omega$ with a non-trivial type then M is head normalizing.

Proof. It suffices to consider closed M. The hypotheses of the Soundness Theorem 6.6 hold for the interpretation $\mathcal{H}$ when $\mathcal{X}$ is $\Lambda\mathrm{x}$: $\mathcal{H}_t$ is $\Lambda\mathrm{x}$-saturated by Lemma 4.2, and the second condition is immediate. So if M is typable with type τ then $M \in \mathcal{H}_\tau$ (for any type τ). Now apply Lemma 6.9. □

The converse of this result will be established in Section 7.

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

Lemma 6.12. $\mathcal{L}_\tau \subseteq \mathcal{LN}$ for each proper type τ.

Proof. We prove the following two statements simultaneously by induction on types.

1 $\mathcal{L}_\tau \subseteq \mathcal{LN}$ when τ is proper, and
2 $(xT_1 \dots T_n) \in \mathcal{L}_\tau$ if each $T_i \in \mathcal{LN}$ and τ is antiproper.

At type-variables t the first claim holds by definition, and the second holds since $(xT_1 \ldots T_n) \in \mathcal{LN} = \mathcal{L}_t$ if each T_i is in $\mathcal{LN}$. At type ω the first claim is vacuous and the second is immediate since $\mathcal{L}_\omega = \Lambda x$. At proper intersection-types the first claim follows by induction since each of the conjuncts of τ is proper; the second claim is vacuously true. At antiproper intersection-types the first claim is vacuously true; the second claim is a consequence of the induction hypothesis since each conjunct of τ is antiproper.

When τ is $\alpha \to \beta$: Suppose τ is proper. To establish the first claim, note that α is antiproper and so as a consequence of the second claim at type α, each variable is in $\mathcal{L}_\alpha$. Then if $M \in \mathcal{L}_{\alpha\to\beta}$, $Mx \in \mathcal{L}_\beta$. The type β is proper, so by induction at type β, $Mx \in \mathcal{LN}$. This implies that $M \in \mathcal{LN}$.

Next suppose that τ is antiproper. To establish the second claim, we want to show $(xT_1 \ldots T_n) \in \mathcal{L}_{\alpha\to\beta}$ given that each $T_i \in \mathcal{LN}$. Let $A \in \mathcal{L}_\alpha$, we want to show $(xT_1 \ldots T_n A) \in \mathcal{L}_\beta$. But since α is proper $A \in \mathcal{LN}$ by induction at α. Since β is antiproper we may apply the induction hypothesis at β to complete the argument. □

Corollary 6.13. If M is typable in $\mathcal{D}_\omega$ with a proper type then M is leftmost normalizing.

Proof. It suffices to consider closed M. The hypotheses of the Soundness Theorem 6.6 hold for the interpretation $\mathcal{L}$ when $\mathcal{X}$ is Λx: $\mathcal{L}_t$ is Λx-saturated by Lemma 4.2, and the second condition is immediate. So if M is typable with type τ then $M \in \mathcal{L}_\tau$ (for any type τ). Now apply Lemma 6.12. □

The converse of this result will be established in Section 7.

Strong normalization

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

Definition 6.14. Let $\mathcal{S}$ be the interpretation which maps each type variable to the set $\mathcal{SN}$ of strongly normalizing terms.

Lemma 6.15. $\mathcal{S}_\tau \subseteq \mathcal{SN}$ for each type τ.

Proof. We prove the following two statements simultaneously by induction on types.

1 $\mathcal{S}_\tau \subseteq \mathcal{SN}$

2 $(xT_1 \ldots T_n) \in \mathcal{S}_\tau$ whenever each $T_i \in \mathcal{SN}$

At type-variables t the first claim holds by definition, and the second is just the statement that $(xT_1 \ldots T_n) \in \mathcal{SN}$. At intersection-types each claim is an immediate consequence of the induction hypothesis.

When τ is $\alpha \to \beta$: To establish the first claim, suppose $M \in \mathcal{S}_{\alpha\to\beta}$, and note that as a consequence of the second claim at type α, each variable is in $\mathcal{S}_\alpha$. So $Mx \in \mathcal{S}_\beta$. Since this is in $\mathcal{SN}$ by induction at type β, M is $\mathcal{SN}$.

For the second claim let $(xT_1 \ldots T_n)$ be given with each $T_i \in \mathcal{SN}$ and let A be in $\mathcal{S}_\alpha$: we seek $(xT_1 \ldots T_n A) \in \mathcal{S}_\beta$. But $A \in \mathcal{SN}$ by induction hypothesis at type α and so $(xT_1 \ldots T_n A) \in \mathcal{S}_\beta$ by induction hypothesis at type β. □

Corollary 6.16. If M is typable in $\mathcal{D}$ then M is strongly normalizing.

Proof. It suffices to consider closed M. The hypotheses of the Soundness Theorem 6.6 hold for the interpretation $\mathcal{S}$ when $\mathcal{X}$ is the set $\mathcal{SN}$: the set $\mathcal{S}_t$ is $\mathcal{SN}$-saturated by Lemma 4.2, and part 1 of the previous lemma established the second condition. So if M is typable with type τ then $M \in \mathcal{S}_\tau$. Now apply Lemma 6.15. □

The converse of this result is false, as will be demonstrated in Section 7 (see Example 7.4).

7. Typings for normalizable terms

In this section we examine how type assignments for Λx terms *reflect* their reduction properties.

First we repeat the classical observations concerning the typings for normal forms and head normal forms.

Proposition 7.1.

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

Proof.

1 By induction on N; let N be $\lambda x_1 \ldots x_n.x_e N_1 \cdots N_k$. For each $1 \leq i \leq k$ N_i is in normal form, so by induction we have Γ_i and τ_i with $\Gamma_i \vdash N_i : \tau_i$. Let t be a type variable and let Γ_0 be the singleton basis $(x_e : \tau_1 \to \cdots \to \tau_k \to t)$, and set Γ to be $\Gamma_0 \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_k$. Then $\Gamma \vdash x_e N_1 \cdots N_k : t$. It follows easily that N is $\mathcal{D}$-typable.
2 By induction on H; let H be $\lambda x_1 \ldots x_n.x_e H_1 \cdots H_k$. Let t be a type variable, let τ_e be the type $(\omega \to \cdots \to \omega \to t)$ (with k occurrences of ω), and let τ_j be an arbitrary type for $j \neq e$. Set Γ to be the basis assigning τ_i to x_i for $1 \leq i \leq n$. Certainly for each i we have $\Gamma \vdash H_i : \omega$, so $\Gamma \vdash x_e H_1 \cdots H_k : t$. Thus $\Gamma \vdash H : \tau_1 \to \cdots \to \tau_n \to t$. This type is non-trivial.

□

In system $\mathcal{D}_\omega$ typings are preserved by the converse of reduction as well.

Theorem 7.2 (Subject Expansion for $\mathcal{D}_\omega$). In system $\mathcal{D}_\omega$: suppose $\Gamma \vdash M : \tau$ and $M_0 \longrightarrow M$. Then $\Gamma \vdash M_0 : \tau$.

Proof. The proof is by induction over the derivation of $\Gamma \vdash M : \tau$.

If the last inference of this derivation is ω-introduction the result is immediate; if the last inference is one of $\cap$-introduction or $\cap$-elimination an easy application of the induction hypothesis suffices. We organize the rest of the argument by cases according to the shape of the term M_0.

If M_0 is an abstraction $\lambda x.B_0$ then M is of the form $\lambda x.B$ with $B_0 \longrightarrow B$; we need only consider the situation in which the typing of M ends with $\to$introduction. Then $\tau = (\tau_1 \to \tau_2)$ and $\Gamma, (x : \tau_1) \vdash B : \tau_2$. The induction hypothesis yields $\Gamma, (x : \tau_1) \vdash B_0 : \tau_2$ then we may type M_0 as desired.

In case M_0 is $F_0 A_0$ there are three cases for the reduction to M. Suppose M is $F_0 A$ with $F_0 \longrightarrow F$ or M is $F A_0$ with $A_0 \longrightarrow A$. As the typing of M is known here to conclude with $\rightarrow$elimination we may apply the induction hypothesis to the typing of F or A as appropriate. The other case is that M_0 is $(\lambda x.B)A$ and M is $B\langle x = A\rangle$, with M typed by the cut rule.

$$\text{cut} \quad \frac{\Gamma, (x\colon \sigma) \vdash B\colon \tau \qquad \Gamma \vdash A\colon \sigma}{\Gamma \vdash B\langle x = A\rangle\colon \tau}.$$

The induction hypothesis is not needed here, since we may type M_0 by expanding the cut. An application of $\rightarrow$introduction types $(\lambda x.B)$ under Γ and then $\rightarrow$elimination types $(\lambda x.B)A$.

The remaining case is when M_0 is a closure $P\langle x = Q\rangle$. If M is obtained by a reduction inside of P or Q then as usual we use a simple application of the induction hypothesis. The interesting sub-cases are when M is produced by a VarK or gc, Varl, Abs, or App reduction. None of these cases involves the induction hypothesis. Of course the gc subcase subsumes that of VarK; we treat this case first.

When $P\langle x = Q\rangle \longrightarrow P$*:* By our variable convention we may assume $x \notin FV(P)$. Let Γ' be Γ restricted to the free variables of P; then $\Gamma' \vdash P\colon \tau$. So $\Gamma', (x\colon \omega) \vdash P\colon \tau$ and $\Gamma' \vdash Q\colon \omega$. By cut, $\Gamma' \vdash P\langle x = Q\rangle\colon \tau$, and so $\Gamma \vdash P\langle x = Q\rangle\colon \tau$.

When $x\langle x = Q\rangle \longrightarrow Q$*:* Let Γ' be Γ restricted to the free variables of Q; then $\Gamma' \vdash Q\colon \tau$. Since $\Gamma', (x\colon \tau) \vdash x\colon \tau$ an application of cut yields $\Gamma' \vdash x\langle x = Q\rangle\colon \tau$, so indeed $\Gamma \vdash x\langle x = Q\rangle\colon \tau$.

When $(\lambda y.C)\langle x = Q\rangle \longrightarrow \lambda y.(C\langle x = Q\rangle)$*:* We may assume M is typed by $\rightarrow$introduction, so that τ is $\tau_1 \rightarrow \tau_2$ and $\Gamma, (y\colon \tau_1) \vdash C\langle x = Q\rangle\colon \tau_2$. By Lemma 5.3.3 there is a σ such that $\Gamma, (y\colon \tau_1), (x\colon \sigma) \vdash C\colon \tau_2$ and $\Gamma, (y\colon \tau_1) \vdash Q\colon \sigma$, and by the variable convention we may assume that y is not free in Q so that in fact $\Gamma \vdash Q\colon \sigma$. Then since $\Gamma, (x\colon \sigma) \vdash \lambda y.C\colon \tau_1 \rightarrow \tau_2$ we may type $(\lambda y.C)\langle x = Q\rangle$ by the cut rule.

When $(UV)\langle x = Q\rangle \longrightarrow (U\langle x = Q\rangle)(V\langle x = Q\rangle)$*:* We may assume M is typed by $\rightarrow$elimination, so there exists δ with $\Gamma \vdash U\langle x = Q\rangle\colon \delta \rightarrow \tau$ and $\Gamma \vdash V\langle x = Q\rangle\colon \delta$. By Lemma 5.3.3 we then obtain σ_1 and σ_2 with $\Gamma, (x\colon \sigma_1) \vdash U\colon \delta \rightarrow \tau$ and $\Gamma \vdash Q\colon \sigma_1$ from the first, and $\Gamma, (x\colon \sigma_2) \vdash V\colon \delta \rightarrow \tau$ and $\Gamma \vdash Q\colon \sigma_2$ from the second. So $\Gamma \vdash Q\colon \sigma_1 \cap \sigma_2$. By Lemma 5.3.1 $\Gamma, (x\colon \sigma_1 \cap \sigma_2) \vdash U\colon \delta \rightarrow \tau$, and $\Gamma, (x\colon \sigma_1 \cap \sigma_2) \vdash V\colon \delta$. Thus $\Gamma, (x\colon \sigma_1 \cap \sigma_2) \vdash (UV)\colon \tau$ and we may type M by a cut.

□

Corollary 7.3 (Subject Conversion for $\mathcal{D}_\omega$). Suppose $\Gamma \vdash M\colon \tau$ in system $\mathcal{D}_\omega$ and $M \twoheadleftarrow\!\!\!\!\!\rightarrow M'$. Then $\Gamma \vdash M'\colon \tau$.

Proof. This follows immediately from the Subject Reduction and Expansion theorems. □

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

In the classical λ-calculus a weak version of the Subject Expansion theorem holds for system $\mathcal{D}$ (under an additional hypothesis that a subterm which is erased by a β-reduction

is typable). Using this obtains a characterization of the strongly normalizing classical terms.

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

Example 7.4. Let S be the term $\lambda u.uu$. Consider the terms

$$\begin{aligned} M_1 &\equiv ((\lambda y.z)xx)\langle x = S\rangle \longrightarrow \\ M_2 &\equiv z\langle y = xx\rangle\langle x = S\rangle \longrightarrow \\ M_3 &\equiv z\langle x = S\rangle. \end{aligned}$$

M_3 and S are easily seen to be $\mathcal{D}$-typable. But M_2 is not $\mathcal{D}$-typable. To see this, first note that the typing behavior of M_2 is the same as that of M_1 since they are related by a B-reduction. But M_1 is not typable since it is not strongly normalizing.

So the reduction from M_2 to M_3 witnesses the failure of Subject Expansion; and indeed the term M_2 witnesses the failure of the converse to "$\mathcal{D}$-typable implies $\mathcal{SN}$."

8. Main Results

In this section we collect the results of the previous sections and derive the main results of the paper; we 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 paradigm 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 continue to hold for the "elementary" calculus without rule gc. It turns out our main results relating various normalization properties and typing properties can be established for the elementary calculus as well with essentially no extra work. This is shown in the three theorems of this section.

Theorem 8.1. 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 $\lambda\mathtt{x}$ (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. The implication from 1 to 2 is Corollary 6.10; the implications from 2 to 3 and from 3 to 4 are immediate. To see that 4 implies 1 suppose $M \twoheadleftarrow\!\!\!\!\rightarrow H$ with H in head-normal form. By Proposition 7.1 H is typable in system $\mathcal{D}_\omega$ with a non-trivial type, and by Corollary 7.3 so is M.

Now 4 implies 5: if $M \in \mathcal{HN}$ then M (head-) reduces to a term of the form

$$\lambda x_1 \ldots x_n.x_i M_1 \cdots M_k.$$

Take X_i to be the constant function taking k arguments and returning I, and take the other X_j to be arbitrary: M is solvable. Finally, 5 implies 2: if $MX_1 \cdots X_n = \lambda x.x$ then $MX_1 \cdots X_n$ has a head normal form, and so $MX_1 \cdots X_n \in \mathcal{HN}$. This implies that $M \in \mathcal{HN}$. □

It is worth emphasizing the fact that the implications 4 to 2 and 4 to 3 state that in $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$ head reduction is a correct strategy for deriving head-normal forms. In light of the fact that head-reduction on $\Lambda\mathtt{x}$ is non-deterministic, these generalize the head-normalization theorem for the classical λ-calculus.

Theorem 8.2. 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. The implication from 1 to 2 is immediate; the implication from 2 to 3 is Corollary 6.13. The implications from 3 to 4 and from 4 to 5 are immediate. Now to see that 5 implies 1 suppose $M \twoheadleftarrow\!\!\!\!\rightarrow N$ with N in normal form. By Proposition 7.1 N is typable in system $\mathcal{D}$, and so in particular is typable with a type τ not involving ω. We may consider this typing to be in system $\mathcal{D}_\omega$ and apply Corollary 7.3 to conclude that M is $\mathcal{D}_\omega$-typable with type τ. □

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. In light of the fact that leftmost reduction on $\Lambda\mathtt{x}$ is non-deterministic, these generalize the leftmost normalization theorem for the classical λ-calculus.

Theorem 8.3. 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 $\mathcal{D}$.

Proof. The first assertion was originally established by Rose, in (Rose 1996). The second result is Corollary 6.16. The third fact follows from the corresponding result for the classical λ-calculus and the preservation of strong normalization (Lescanne & Rouyer-Degli 1994, Bloo & Rose 1995, Bloo 1997, Bloo & Geuvers 1999, Rose 1996). □

The requirement that M be a pure term in part 3 is necessary, as shown by Example 7.4.

9. Conclusions and future work

In this paper we defined an intersection-types system for terms in the explicit substitution calculi $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$ which is a natural generalization of the system for the classical

lambda-calculus. We defined notions of head and leftmost reduction which generalize classical head reduction yet are more flexible in the sense that they are not deterministic.

Characterizations of the head- and leftmost-normalizing terms were obtained in terms of their typing behavior. Leftmost reduction is a normalizing strategy. The nondeterminism in leftmost reduction means that one has a choice of reduction strategy, for example in a programming language implementation, while retaining computational completeness.

Terms typable in the system $\mathcal{D}$ without type ω were shown to be strongly normalizing; the converse fails. For pure terms the strong normalization of typable terms follows from the corresponding result for the classical λ-calculus and the previously established "preservation of strong normalization" results for $\lambda\mathtt{x}$; our treatment here is direct and makes no reference to analysis in the classical lambda-calculus.

All of our results hold for both reduction systems, $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$.

It is not true that all strongly normalizing terms are typable in system $\mathcal{D}$. This surprising failure of the most natural generalization of the classical characterization of $\mathcal{SN}$ terms gives rise to two complementary questions:

— Which terms are typable in system $\mathcal{D}$? For example, is there a reduction relation on the terms of $\Lambda\mathtt{x}$ for which system $\mathcal{D}$ captures the SN terms?

— Is there a type system such that the typable terms are precisely the $\lambda\mathtt{x}_{gc}$-$\mathcal{SN}$ terms?

Recall Example 7.4. The terms M_1 and M_2 there show that if we are to have a type system which characterizes strong normalization then we apparently 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.

Acknowledgements

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

References

Abadi, M., Cardelli, L., Curien, P.-L. & Lévy, J.-J. (1991), 'Explicit substitutions', *Journal of Functional Programming* **1**(4), 375–416.

Amadio, R. & Curien, P.-L. (1998), *Domains and lambda-calculi*, Cambridge University Press.

Barendregt, H. P. (1984), *The Lambda-Calculus, its syntax and semantics*, Studies in Logic and the Foundation of Mathematics, Elsevier Science Publishers B. V. (North-Holland), Amsterdam. Second edition.

Barendregt, H. P. (1992), Lambda calculi with types, *in* S. Abramsky, D. M. Gabby & T. S. E. Maibaum, eds, 'Handbook of Logic in Computer Science', Vol. 2, Oxford University Press, chapter 2, pp. 117–309.

Benaissa, Z., Briaud, D., Lescanne, P. & Rouyer-Degli, J. (1996), 'λv, a calculus of explicit substitutions which preserves strong normalisation', *Journal of Functional Programming* **6**(5), 699–722.

Benaissa, Z., Lescanne, P. & Rose, K. H. (1996), Modeling sharing and recursion for weak reduction strategies using explicit substitution, *in* H. Kuchen & D. Swierstra, eds, 'PLILP '96—Eighth International Symposium on Programming Languages: Implementation, Logics and Programs', number 1140 *in* 'Lecture Notes in Computer Science', Springer-Verlag, Aachen, Germany, pp. 393–407.
*ftp://ftp.ens-lyon.fr/ pub/users/ LIP/krisrose/ PAPERS/benaissa+lescanne+rose-plilp96.ps.gz

Bloo, R. (1997), Preservation of Termination for Explicit Substitution, PhD thesis, Technische Universiteit Eindhoven. IPA Dissertation Series 1997-05.

Bloo, R. & Geuvers, J. H. (1999), 'Explicit substitution: on the edge of strong normalization', *Theoretical Computer Science* **211**, 375 – 395.

Bloo, R. & Rose, K. H. (1995), Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection, *in* 'CSN '95—Computing Science in the Netherlands', Koninklijke Jaarbeurs, Utrecht, pp. 62–72.
*ftp:// ftp.diku.dk/ diku/ semantics/ papers/ D-246.ps

Bonelli, E. (2001), 'Perpetuality in a named lambda calculus with explicit substitutions and some applications', *Mathematical Structures in Compuer Science* . to appear.

Cardone, F. & Coppo, M. (1990), Two extension of Curry's type inference system, *in* P. Odifreddi, ed., 'Logic and Computer Science', Vol. 31 of *APIC Series*, Academic Press, New York, NY, pp. 19–75.

Coppo, M. & Dezani-Ciancaglini, M. (1978), 'A new type assignment for lambda-terms', *Archiv für mathematische Logik und Grundlagenforschung* **19**, 139–156.

Curien, P.-L., Hardin, T. & Lévy, J.-J. (1996), 'Confluence properties of weak and strong calculi of explicit substitutions', *Journal of the ACM* **43**(2), 362–397.

Curry, H. B. & Feys, R. (1958), *Combinatory Logic I*, North-Holland, Amsterdam.

Dershowitz, N. & Manna, Z. (1979), 'Proving termination with multiset orderings', *Communications of the ACM* **22**(8), 465–476.

Di Cosmo, R. & Kesner, D. (1997), Strong normalization of explicit substitutions via cut elimination in proof nets, *in* 'LICS '97—Twelfth Annual IEEE Symposium on Logic in Computer Science', Warsaw University, IEEEC Society Press, Warsaw, Poland, pp. 35–46.
*ftp:// ftp.lri.fr/ LRI/ articles/ kesner/ preliminary-version.ps.gz

Ghilezan, S. (1996), 'Strong normalization and typability with intersection types', *Notre Dame J. Formal Loigc* **37**(1), 44–52.

Girard, J.-Y. (1971), Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, *in* J. Fenstad, ed., '2nd Scandinavian Logic Symposium', North-Holland, Amsterdam, pp. 63–92.

Girard, J.-Y. (1972), 'Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur', These D'État, Université Paris VII.

Guillaume, B. (2000), 'The $\lambda - s_e$-calculus does not preserve strong normalisation', *Journal of Functional Programming* . to appear.

Herbelin, H. (2001), 'Explicit substitutions and reducibility', *J. of Logic and Computation* **11**(3), 429–449.

Kamareddine, F. & Nederpelt, R. (1993), 'On stepwise explicit substitutions', *International Journal of Foundations of Computer Science* **4**(3), 197–240.

Kamareddine, F. & Ríos, A. (1995), A λ-calculus à la de Bruijn with explicit substitutions, *in* 'PLILP '95 – Programming Languages: Implementations, Logics, and Programs', number 982 *in* 'Lecture Notes in Computer Science', Springer-Verlag, pp. 45–62.

Kamareddine, F. & Ríos, A. (1997), 'Extending a lambda-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms', *Journal of Functional Programming* **7**(4), 395–420.

Kamareddine, F. & Ríos, A. (2000), 'Relating the lambda-sigma and lambda-s styles of explicit substitutions', *Journal of Logic and Computation* **10**(3). Special issue on Type Theory and Term Rewriting.

Krivine, J.-L. (1993), *Lambda calculus, types and models*, Ellis Horwood.

Leivant, D. (1986), 'Typing and computational properties of lambda expressions', *Theoretical Computer Science* **44**(1), 51–68.

Lescanne, P. (1994), From $\lambda\sigma$ to λv: a journey through calculi of explicit substitutions, *in* H.-J. Boehm, ed., 'POPL '94—21st Annual ACM Symposium on Principles of Programming Languages', ACM, Portland, Oregon, pp. 60–69.

Lescanne, P. & Rouyer-Degli, J. (1994), The calculus of explicit substitutions λv, Technical Report RR-2222, INRIA-Lorraine.

Melliès, P.-A. (1995), Typed λ-calculi with explicit substitution may not terminate, *in* M. Dezani, ed., 'TLCA '95—Int. Conf. on Typed Lambda Calculus and Applications', Vol. 902 of *Lecture Notes in Computer Science*, Springer-Verlag, Edinburgh, Scotland, pp. 328–334.

Melliès, P.-A. (1997), Axiomatic rewriting theory III, a factorisation theorem in rewriting theory, *in* 'Proceedings of the 7th Conference on Category Theory and Computer Science', Vol. 1290 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 49–68.

Mitchell, J. (1986), A type-inference approach to reduction properties and semantics of polymorphic expressions, *in* 'ACM Conference on LISP and Functional Programming', pp. 308–319. Reprinted with minor revisions in *Logical Foundations of Functional Programming,* ed. G. Huet, Addison-Wesley (1990) 195–212.

Mitchell, J. C. (1996), *Foundations for Programming Languages*, MIT Press, Cambridge, MA.

Pottinger, G. (1980), A type assignment for the strongly normalizable λ-terms, *in* 'To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism', Academic Press, pp. 561–578.

Ritter, E. (1999), Characterising explicit substitutions which preserve termination, *in* 'TLCA'99', Vol. 1581 of *Lecture Notes in Computer Science*, Springer-Verlag.

Rose, K. (1996), Operational Reduction Models for Functional Programming Languages, PhD thesis, DIKU, Universitetsparken 1, DK-2100 København Ø. DIKU report 96/1.
*http://www.diku.dk/research/published/96-1.ps.gz

Sallé, P. (1978), Une extension de la théorie des types en λ-calcul, *in* G. Ausiello & C. Boehm, eds, 'Fifth Colloquium on Automata, Languages and Programming', Vol. 62 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 398–410.

Tait, W. (1967), 'Intensional interpretation of functionals of finite type', *J. Symbolic Logic* **32**, 198–212.

Tait, W. W. (1975), A realizability interpretation of the theory of species, *in* R. Parikh, ed., 'Logic Colloquium', Vol. 453, Springer-Verlag, Boston, pp. 240–251.

van Bakel, S. (1992), 'Complete restrictions of the intersection type discipline', *Theoretical Computer Science* **102**(1), 135–163.

van Raamsdonk, F., Severi, P., Sørensen, M. H. B. & Xi, H. (1999), 'Perpetual reductions in λ-calculus', *Information and Computation* **149**(2), 173–225.