Reductions, intersection types, and explicit substitutions *

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

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

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

Abstract. 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 an 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 the primitive notions of abstraction and application in terms of *substitution*. Classical λ-calculus treats substitution as an atomic operation, but in the presence of variable-binding substitution it is a complex operation to define and to implement. So a more careful analysis is required if one is to reason about the correctness of compilers, theorem provers, or proof-checkers. Furthermore the actual cost of performing substitution should be considered when reasoning about complexity of implementations.

Abadi, Cardelli, Curien, and Lévy [1] defined a calculus of *explicit substitutions* to serve as a more faithful model of implementations of the λ-calculus. Since then a variety of explicit substitutions calculi have been defined.

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

The original motivation for the Abadi-Cardelli-Curien-Lévy calculus was pragmatic, but there is another point of view one may take on such a calculus, namely that making substitution explicit permits a more refined analysis of substitution than does classical λ-calculus.

As historical context we note that in their book [14] Curry and Feys insist on the importance of substitution in logic in general and especially in the framework of λ-calculus. They write [page 6] that the synthetic theory of combinators "gives the ultimate analysis of substitutions in terms of a system of extreme simplicity. The theory of lambda-conversion is intermediate in character between synthetic theories and ordinary logics. Although its analysis is in some ways less profound—many of the complexities in regard to variables are still unanalyzed there—yet it is none the less significant; and it has the advantage of departing less radically from our intuition."

From this point of view one can see an explicit substitution calculus as an improvement on both the system of combinators and the classical λ-calculus, since it is a system whose mechanics are first-order and as simple as those of combinatory logic yet which retains the same intensional character as traditional λ-calculus. In particular we may view explicit substitution calculi as primary and see the classical λ-calculus as a subsystem of these systems, defined by a particular strategy of "eagerly" evaluating the substitution constructed by contracting a β-redex. In this way the study of explicit substitutions represents a deeper examination of the relationship between abstraction and application. This setting invites the programme of refining the results of the classical λ-calculus by finding proofs of their explicit-substitutions analogues *in the explicit substitutions system itself.* One can reasonably expect in this way to gain insight into the original λ-calculus. As a case study, in this paper we present a systematic study of the relation between normalization and types.

In many calculi of explicit substitutions, including the original Abadi, Cardelli, Curien, Lévy system, substitutions are first-class citizens and there is an algebraic/computational structure on the substitutions themselves, reflecting the fact that composition is a natural operation on substitutions. Melliès [27] 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 [25, 8, 5] 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 [28] where a similar issue is discussed.

Surprisingly, the situation for the strongly normalizing terms diverges from the classical λ-calculus. For the natural generalization of the classical type system we prove that typable terms are strongly normalizing. But the converse fails: see Section 7.

In addition to their theoretical and methodological interest our results have consequences for the study of the implementation of functional programming languages. Recall that the theoretical foundation for the correctness of the standard evaluation strategy for functional languages is the classical theorem that leftmost reduction is normalizing (see for example [29] Prop. 2.4.12). When explicit substitutions are offered as a basis for an implementation one should define and analyze a corresponding notion of "leftmost" reduction. To our knowledge this analysis has not been previously done. The natural notion of leftmost reduction we define here is related to, but a refinement of, the classical notion; the non-determinism in leftmost reduction here corresponds to a choice between certain standard implementation strategies ([6]).

The proofs we present here readily yield the results that a term is (strong-, leftmost-, or head-) normalizing iff it is so in the calculus extended by garbage-collection. Our results support the claim that garbage-collection is a very natural addition to the system, even from a purely theoretical point of view, since the resulting calculus has more convenient closure properties than the simple calculus (Lemma 2 is an example).

The intersection type systems we study are natural generalizations of the corresponding classical systems, and in fact the global structure of the proofs follow a standard paradigm (as in [4]). But the explicit reductions involving substitutions lead to combinatorial complications not arising in the traditional treatments and the proofs require some new techniques.

The first result on strong normalization of calculi of explicit substitution was the so-called *preservation of strong normalization:* a pure (substitution-free) term is strongly normalizing under reduction in the presence of explicit substitutions if and only if it is strongly normalizing under β-reduction. We stress that, in keeping with our aim of treating the explicit substitutions calculus as logically prior to the traditional λ-calculus, we develop the machinery needed for direct proofs which do not depend on results from the theory of β-reduction.

Related work

The system of intersection types is due to Coppo and Dezani [12] and Sallé [33]. The fact that the strongly normalizing terms are precisely the typable terms seems to have been first proved in [30]. Other notable works in this area include [24, 34, 17], and the books [23] and [2].

The first proof — actually a sketch of a proof — of preservation of strong normalization (PSN) appears in [26] for $\lambda\upsilon$ ([5] has the complete proof) and at almost the same time Bloo and Rose [9] proposed a proof of PSN for $\lambda\mathbf{x}$ (see also [7, 8, 32]). Kamareddine and Rios [20] gave proofs of PSN for another calculus of explicit substitutions called λs. The proof of Di-Cosmo and Kesner [16] relies on properties of Girard's proof nets for linear logic. Ritter [31] sketches a generic proof of PSN for a general family of systems of explicit substitutions. Bonelli [10] proposes a proof of strong normalization for $\lambda\mathbf{x}_{gc}$ inspired by the proof of [5] 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 and he characterizes SN-terms inductively [35]. From this, he deduces a proof of SN for the terms typable in a system of polymorphic types. Herbelin [19] is interested in reduction and strategies and like us, he uses reducibility. Melliès [27] showed that $\lambda\sigma$ does not have PSN; Guillaume [18] 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 [4], to which we refer the reader for background.

2 Terms and reduction strategies

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

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

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

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

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

We assume Barendregt's [3] convention, namely that *a variable does not occur free and bound in the same term*. For instance, we assume that x does not occur free in N in the term $M\langle x = N\rangle$. The rules we define further assume this convention and the reader should keep this fact in mind when reading them and certain forthcoming lemmas.

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

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

$\lambda x.B$	$(\lambda y.B)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\,\boldsymbol{T}$
$(\lambda x.B)AT_1\cdots T_n$ *with* $n \geq 0$	$(UV)\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\,\boldsymbol{T}$
$xT_1\cdots T_n$ *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 1. On the other hand, each term M has this form. If M is an abstraction, this is covered by Lmb. If M is a variable, this is covered by VarHd with $n = 0$. If M is a closure, then this is covered by one of the last four cases (I Clo, K Clo, AbsClo, AppClo) as we will see below. If M is an application, M is of the form $N\overrightarrow{T}$ where N is either a variable (VarHd) or an abstraction (βHd) or a closure. If N is a closure, N is of the form $P\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\,\boldsymbol{T}$ where P is either a variable (I Clo or K Clo) or an abstraction (AbsClo) or an application (AppClo). ///

Following Barendregt [3] 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 [6]. 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 [9, 32, 7]. 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. *The notions of reduction* $\lambda\mathtt{x}$ *and* $\lambda\mathtt{x}_{gc}$ *are induced by the rules in Table 1: the notion of reduction* $\lambda\mathtt{x}$ *is obtained by deleting the rule* gc*, and the notion of reduction* $\lambda\mathtt{x}_{gc}$ *is obtained by deleting the rule* VarK*.*

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

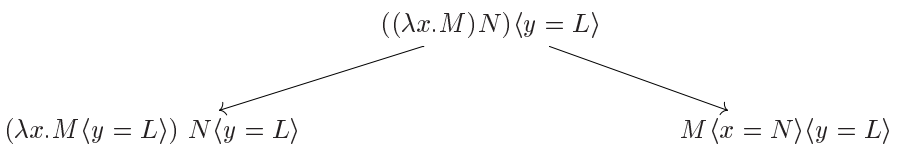
(B)	$(\lambda x B)\, A$	$\to$	$B\langle x = A\rangle$	
(App)	$(MN)\langle x = A\rangle$	$\to$	$M\langle x = A\rangle\, N\langle x = A\rangle$	
(Abs)	$(\lambda y M)\langle x = N\rangle$	$\to$	$\lambda y M\langle x = N\rangle$	
(Varl)	$x\langle x = N\rangle$	$\to$	N	
(VarK)	$y\langle x = N\rangle$	$\to$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

Table 1. The reduction rules.

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

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

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



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

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

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

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

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

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

Table 2. Head reduction

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

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

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

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

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

Table 3. Leftmost reduction

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

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

3 Two fundamental Lemmas

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

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

and

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

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

A remark on the classical Substitution Lemma

The Substitution Lemma of the classical λ-calculus [3] states a fundamental property of (implicit) substitutions, namely that, when x is not free in L

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

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

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

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

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

3.1 The Composition Lemma

Let us consider the following rule

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

which we call *composition*. It abstracts the composition one finds in systems like $\lambda\sigma$ [1,13] (namely the rule called *Map*) or in the extension $\lambda\mathbf{x}\|c$ of $\lambda\mathbf{x}$ [7,8,21,22] (namely the rule $\xrightarrow[\|c]{}$ of [7], see also [32] page 75).

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

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

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

$$[\mathsf{Cref}] \quad M \Rightarrow M$$

$$[\mathsf{Cabs}] \quad \frac{B \Rightarrow B'}{\lambda x.B \Rightarrow \lambda x.B'} \qquad [\mathsf{Cpabs}] \quad \frac{B\langle y = Q\rangle \Rightarrow B^+}{(\lambda x.B)\langle y = Q\rangle \Rightarrow \lambda x.B^+}$$

$$[\mathsf{Capp}] \quad \frac{U \Rightarrow U' \qquad V \Rightarrow V'}{UV \Rightarrow U'V'} \qquad [\mathsf{Cpapp}] \quad \frac{U\langle y = Q\rangle \Rightarrow U^+ \qquad V\langle y = Q\rangle \Rightarrow V^+}{(UV)\langle y = Q\rangle \Rightarrow (U^+V^+)}$$

$$[\mathsf{Cclo}] \quad \frac{B \Rightarrow B' \qquad A \Rightarrow A'}{B\langle z = A\rangle \Rightarrow B'\langle z = A'\rangle} \qquad [\mathsf{Cpclo}] \quad \frac{M\langle y = Q\rangle \Rightarrow M^+ \qquad P\langle y = Q\rangle \Rightarrow P^+}{M\langle x = P\rangle\langle y = Q\rangle \Rightarrow M^+\langle x = P^+\rangle}$$

Table 4. The rules for $\Rightarrow$

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

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

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 [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

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

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

$M \Rightarrow M'$ is Capp

There are two sub-cases:

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

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

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

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

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

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

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

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

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

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 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:

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

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:

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

Complete the square by:

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

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

$M \Rightarrow M'$ is Cpclo

Here we have

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

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

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

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 Q. 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 & \cdots\cdots\cdots\

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

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

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

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

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

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

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

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

3.2 The Closure lemma

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

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

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

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

– *if* $C[\![M_1,...,M_n]\!] \in \mathcal{HN}$ *then* $C[\![M_1\langle x = A_1\rangle,...,M_n\langle x = A_n\rangle]\!] \in \mathcal{HN}$.
– *if* $C[\![M_1,...,M_n]\!] \in \mathcal{LN}$ *then* $C[\![M_1\langle x = A_1\rangle,...,M_n\langle x = A_n\rangle]\!] \in \mathcal{LN}$.
– *if* $C[\![M_1,...,M_n]\!] \in \mathcal{SN}$ *and* $A_i \in \mathcal{SN}$ *for* $1 \leq i \leq n$ *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 the three statements is the same, so let us introduce notation to avoid repetition of the argument. We let $\mathcal{N}$ stand any of for $\mathcal{HN}$, $\mathcal{LN}$, or $\mathcal{SN}$, 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; we let $\sqsupset^m$ be the multiset extension [15] of the superterm order $\sqsupset$ and $\longrightarrow^m$ 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 (by hypothesis); $\boldsymbol{M}$ will be $\{M_1,...,M_n\}$; $\boldsymbol{A}$ will be $\{A_1,...,A_n\}$ and its $\lambda\mathbf{x}_{gc}$-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.

Assume the induction hypothesis and that $C[\![M_1,...,M_n]\!] \in \mathcal{N}$ (and that for $1 \leq i \leq n$, $A_i \in \mathcal{SN}$ 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 2. *Let $M \equiv N\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$ with $x \notin FV(N)$ and let $M' \equiv N\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}$,*

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

4 Saturated Sets

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

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 $\dfrac{N\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T} \qquad A \in \mathcal{X} \text{ and } x \notin FV(N)}{N\langle x = A\rangle\langle \boldsymbol{z} = \boldsymbol{S}\rangle\, \boldsymbol{T}}$

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

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

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

The following are very easy consequences of the definition.

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

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

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

Lemma 5.

- *$\mathcal{SN}$ is $\mathcal{SN}$-saturated.*
- *$\mathcal{HN}$ is $\Lambda\boldsymbol{x}$-saturated.*
- *$\mathcal{LN}$ is $\Lambda\boldsymbol{x}$-saturated.*

Proof. We wish to show that each of $\mathcal{SN}$, $\mathcal{HN}$, and $\mathcal{LN}$ is closed under the rules in Definition 8.

It suffices to show that if M fails to be in $\mathcal{SN}$ (or $\mathcal{HN}$ or $\mathcal{LN}$ 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-App, sat-gc and sat-gc this is easy to see. Corollary 2 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 Corollary 1 precludes. Corollary 1 directly handles the case of sat-comp. ///

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

5 Types

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

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

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

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

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

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

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

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

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

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 11. *Let Γ_1 and Γ_2 be bases. The basis $\Gamma_1 \sqcap \Gamma_2$ contains $x : \sigma$ if either:*

- *$(x : \sigma)$ is in Γ_1 and $x \notin \mathcal{D}om(\Gamma_2)$, or*
- *$(x : \sigma)$ is in Γ_2 and $x \notin \mathcal{D}om(\Gamma_1)$, or*
- *$\sigma = \sigma_1 \cap \sigma_2$ with $(x : \sigma_1) \in \Gamma_1$ and $(x : \sigma_2) \in \Gamma_2$.*

Lemma 6. *In each of $\mathcal{D}$ and $\mathcal{D}_\omega$:*

1. *If $\Gamma \vdash M : \tau$ and $\Gamma \subseteq \Gamma'$ then $\Gamma' \vdash M : \tau$.*
2. *If $\Gamma \vdash M : \tau$ and Γ' agrees with Γ on the free variables of M then $\Gamma' \vdash M : \tau$.*
3. *if $\Gamma \vdash M : \tau$ then for all Γ', $\Gamma \sqcap \Gamma' \vdash M : \tau$.*

Proof. Each of these is an easy induction over typing derivations. ///

Soundness

Here we consider the type systems $\mathcal{D}$ and $\mathcal{D}_\omega$ simultaneously.

Definition 12. *An interpretation $\mathcal{I}$ is a function from types to sets of terms obeying the following*

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

Obviously an interpretation is completely determined by its value on the type variables.

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

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

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

Proof. Throughout the proof we will use the fact that each $\mathcal{I}_\tau$ is $\mathcal{X}$-saturated.

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
- $(x_1 : \alpha_1), (x_2 : \alpha_2), \ldots, (x_n : \alpha_n) \vdash M : \tau$ 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 $(x_1 : \alpha_1), (x_2 : \alpha_2), \ldots, (x_n : \alpha_n)$. The proof is by induction on the derivation of $\Gamma \vdash M : \tau$.

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

The start rule. Here, for some i, $M \equiv x_i$ and $\tau = \alpha_i$. To show that $x_i\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_\tau$ 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 $\rightarrow$E rule. We have

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

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 $(U\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle) \in \mathcal{I}_{\alpha\rightarrow\beta}$ 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 $\mathcal{I}_{\alpha\rightarrow\beta}$.

The $\rightarrow$I rule. We have

$$\frac{x : \alpha, \Gamma \vdash B : \beta}{\Gamma \vdash \lambda x.B : \alpha \rightarrow \beta}$$

We may assume that the variable x is not free in any of the A_i. To show that $\lambda x.B\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_{\alpha\rightarrow\beta}$ we may iterate rule sat-Abs and argue that $\lambda x.B\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_{\alpha\rightarrow\beta}$. 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

$$\frac{x : \sigma, \Gamma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\langle x = N\rangle : \tau}$$

We wish to show that $M\langle x = N\rangle\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle \in \mathcal{I}_\tau$; 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 $\mathcal{I}_\tau$:

$$M\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle\langle x = N\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle\rangle$$

By the induction hypothesis applied to the derivation $\Gamma \vdash N : \sigma$, the term $N\langle x_1 = A_1\rangle ... \langle x_n = A_n\rangle$ 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 $x : \sigma, \Gamma \vdash M : \tau$ to finish the argument. ///

6 Normalization

The following definitions are due to Cardone and Coppo [11].

Definition 13. *A type is* proper *if it has no positive occurrence of* ω*,* antiproper *if it has no negative occurrence of* ω*, and* strictly proper *if it has no occurrence of* ω*.*

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

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

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

Lemma 7. $\mathcal{H}_\tau \subseteq \mathcal{HN}$ *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 $\mathcal{H}_\omega = \Lambda\mathsf{x}$. At intersection-types the first claim holds 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 3. *If* M *is typable in* $\mathcal{D}_\omega$ *with a non-trivial type then* M *is head normalizing.*

Proof. The hypotheses of the Soundness Theorem hold for the interpretation $\mathcal{H}$ when $\mathcal{X}$ is $\Lambda\mathsf{x}$: $\mathcal{H}_t$ is $\mathcal{HN}$-saturated by Lemma 5, 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 7. ///

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

Lemma 8. *$\mathcal{L}_\tau \subseteq \mathcal{LN}$ 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 $(xT_1 \dots T_n) \in \mathcal{LN} = \mathcal{L}_t$ if each T_i is in $\mathcal{LN}$. At type ω the first claim is vacuous and the second is immediate since $\mathcal{L}_\omega = \Lambda\mathtt{x}$. 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 $(xT_1 \dots T_n) \in \mathcal{L}_{\alpha\to\beta}$ given that each $T_i \in \mathcal{LN}$. Let $A \in \mathcal{L}_\alpha$, we want to show $(xT_1 \dots T_nA) \in \mathcal{L}_\beta$. 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 4. *If M is typable in $\mathcal{D}_\omega$ with a proper type then M is leftmost normalizing.*

Proof. The hypotheses of the Soundness Theorem hold for the interpretation $\mathcal{L}$ when $\mathcal{X}$ is $\Lambda\mathtt{x}$: $\mathcal{L}_t$ is $\Lambda\mathtt{x}$-saturated by Lemma 5, 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 8. ///

6.3 Strong normalization

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

Definition 16. *Let $\mathcal{S}$ be the interpretation which maps each type variable to the set $\mathcal{SN}$ of strongly normalizing terms.*

Lemma 9. *$\mathcal{S}_\tau \subseteq \mathcal{SN}$ for each type τ.*

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

1. $\mathcal{S}_\tau \subseteq \mathcal{SN}$
2. $(xT_1 \dots T_n) \in \mathcal{S}_\tau$ 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 $(xT_1 \dots T_n) \in \mathcal{SN}$. 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 $(xT_1 \dots T_n)$ be given with each $T_i \in \mathcal{SN}$ and let A be in $\mathcal{S}_\alpha$: we seek $(xT_1 \dots T_n A) \in \mathcal{S}_\beta$. But $A \in \mathcal{SN}$ by induction hypothesis at type α and so $(xT_1 \dots T_n A) \in \mathcal{S}_\beta$ by induction hypothesis at type β. ///

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

Proof. We first verify the hypotheses of the Soundness Theorem 1 for the interpretation $\mathcal{S}$ when $\mathcal{X}$ is the set $\mathcal{SN}$. The set $\mathcal{S}_t$ is $\mathcal{SN}$-saturated by Lemma 5, and the previous lemma established the second condition. So if M is typable with type τ then $M \in \mathcal{S}_\tau$. Now apply Lemma 9. ///

7 Typings for normalizable terms

Proposition 1.

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

Proof.

1. By induction on H; let H be $\lambda x_1 \dots x_n . x_e H_1 \cdots H_k$. 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.
2. By induction on N; let N be $\lambda x_1 \dots x_n . x_e N_1 \cdots N_k$. 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 $\Gamma_0 \sqcap \Gamma_1 \sqcap \dots \sqcap \Gamma_k$. Then $\Gamma \vdash x_e N_1 \cdots N_k : t$. It follows easily that N is $\mathcal{D}$-typable.

///

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

Proof. By induction over 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 $B \longrightarrow B_1$. We show that $\Gamma \vdash M : \tau$ implies $\Gamma \vdash M_1 : \tau$; we need only consider the situation which the typing of M ends with $\to$introduction. Then $\tau = (\tau_1 \to \tau_2)$ and $\Gamma, y : \tau_1 \vdash B : \tau_2$. The induction hypothesis yields the desired typing for B_1 and then we may type M_1 as desired.

In case M is FA and was typed by

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

there are three cases for the reduction to M_1. If M_1 is F_1A 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. On the other hand, if M is $(\lambda x.B)A$ and M_1 is $B\langle x = A\rangle$ then we may type M_1 as follows

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

Since $\Gamma \vdash \lambda x.B\colon \sigma\rightarrow\tau$ we can be sure that $x\colon \sigma, \Gamma \vdash B\colon \tau$.

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

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

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 $Y\langle x = Q\rangle \longrightarrow Y$*:* We have $x\colon \sigma, \Gamma \vdash Y\colon \tau$ and so $\Gamma \vdash y\colon \tau$ as desired.

When $x\langle x = Q\rangle \longrightarrow Q$*:* The derivation $\Gamma, x\colon \sigma \vdash x\colon \tau$ 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 $\Gamma, x\colon \sigma \vdash x\colon \tau_i$ for $i = 1, 2$ and so by induction (twice) we have $\Gamma \vdash Q\colon \tau_i$ for $i = 1, 2$, and so $\Gamma \vdash Q\colon \tau$. 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 $x\colon \sigma, \Gamma \vdash \lambda y.C\colon \tau$ and $\Gamma \vdash Q\colon \sigma$; we seek $\Gamma \vdash \lambda y.(C\langle x = Q\rangle)\colon \tau$. If the last inference in $x\colon \sigma, \Gamma \vdash \lambda y.C\colon \tau$ is $\cap$-introduction or elimination then it is straightforward to rearrange the deductions just as in the previous case. Otherwise τ is $\tau_1\rightarrow\tau_2$ and $y\colon \tau_1, x\colon \sigma, \Gamma \vdash C\colon \tau_2$. Certainly $y\colon \tau_1, \Gamma \vdash Q\colon \sigma$, so an application of cut yields $y\colon \tau_1 \vdash P\langle x = Q\rangle\colon \tau_2$. A final application of $\rightarrow$introduction finishes the argument.

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

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

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

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 show that $\Gamma \vdash M\colon \tau$ implies $\Gamma \vdash M_0\colon \tau$; we need only consider the situation which the typing of M ends with $\rightarrow$introduction. Then $\tau = (\tau_1 \rightarrow \tau_2)$ and $\Gamma, x\colon \tau_1 \vdash B\colon \tau_2$. The induction hypothesis yields $\Gamma, x\colon \tau_1 \vdash B_0\colon \tau_2$ then we may type M_0 as desired.

In case M_0 is F_0A_0 there are three cases for the reduction to M. Suppose M is F_0A with $F_0 \longrightarrow F$ or M is FA_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 $(\lambda x.B)A_0$ and M is $B\langle x = A_0\rangle$, with M typed by the cut rule.

$$\text{cut}\quad \frac{\Gamma, x\colon \sigma \vdash B\colon \tau \qquad \Gamma \vdash A_0\colon \tau}{\Gamma \vdash B\langle x = A_0\rangle\colon \tau}.$$

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 $(\lambda x.B)A_0$.

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 sub-case subsumes that of VarK; we treat this case first.

When $Y\langle x = Q\rangle \longrightarrow Y$*:* Since $x \notin Var(Y)$, without loss of generality we may assume $x \notin \Gamma$. Then $\Gamma, x\colon \omega \vdash Y\colon \tau$. Moreover $\Gamma \vdash Q\colon \omega$. Then $\Gamma \vdash Y\langle x = Q\rangle\colon \tau$ by the cut rule.

When $x\langle x = Q\rangle \longrightarrow Q$*:* Assume $\Gamma \vdash Q\colon \tau$. Without loss of generality $x \notin \Gamma$, so $\Gamma, x\colon \tau \vdash x\colon \tau$, and an application of cut yields $\Gamma \vdash x\langle x = Q\rangle\colon \tau$.

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 $\Gamma, y\colon \tau_1 \vdash C\langle x = Q\rangle\colon \tau_2$. We know that there is a σ such that $\Gamma, y\colon \tau_1, x\colon \sigma \vdash C\colon \tau_2$ and $\Gamma, y\colon \tau_1 \vdash Q\colon \sigma$. The first of these yields $\Gamma, x\colon \sigma \vdash \lambda y.C\colon \tau_1 \rightarrow \tau_2$. By the variable convention we may assume that y is not free in Q so that in fact $\Gamma \vdash Q\colon \sigma$. Then 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 $\Gamma \vdash V\langle x = Q\rangle\colon \delta$. 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 $\Gamma, x\colon \sigma_2 \vdash V\colon \delta$ and $\Gamma \vdash Q\colon \sigma_2$ from the second. So $\Gamma \vdash Q\colon \sigma_1 \cap \sigma_2$. By Lemma 6 $\Gamma, x\colon \sigma_1 \cap \sigma_2 \vdash U\colon \delta \rightarrow \tau$, 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 6. *Suppose* $\Gamma \vdash M\colon \tau$ *in system* $\mathcal{D}_\omega$ *and* $M \twoheadleftarrow\!\!\!\!\twoheadrightarrow M'$*. Then* $\Gamma \vdash M'\colon \tau$*.*

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

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

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

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

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

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

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

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

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

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

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

8 Summary

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

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

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

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

8.1 Head normalization

Theorem 4. *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 3; the implications from 2 to 3 and from 3 to 4 are immediate. To see that 4 implies 1 suppose $M \twoheadleftarrow\!\!\!\!\twoheadrightarrow H$ with H in head-normal form. By Proposition 1 H is typable in system $\mathcal{D}_\omega$ with a non-trivial type, and by Corollary 6 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 $M \in \mathcal{HN}$. ///

8.2 Leftmost normalization

Theorem 5. *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 4. The implications from 3 to 4 and from 4 to 5 are immediate. Now to see that 5 implies 1 suppose $M \twoheadleftarrow\!\!\!\!\twoheadrightarrow N$ with N in normal form. By Proposition 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 6 to conclude that M is $\mathcal{D}_\omega$-typable with type τ. ///

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

8.3 Strong normalization

Theorem 6. *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 is due to Rose, in [32]. The second result is Corollary 5.The third fact follows from the correspondng result for the classical λ-calculus and the preservation of strong normalization [26, 9, 7, 8, 32]. ///

As described in the previous section we do not have the implication "$M \in \mathcal{SN}$ implies M is typable in system $\mathcal{D}$." As is well-known this result holds for pure terms under β-reduction. It then follows from the preservation of strong normalization in $\lambda\mathtt{x}$ that the result holds for pure terms under $\lambda\mathtt{x}_{gc}$-reduction.

The problem of finding a reasonable type system characterizing the strongly normalizing terms in $\Lambda\mathtt{x}$ is left as an open problem.

Acknowledgements

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

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
2. R. Amadio and P.-L. Curien. *Domains and lambda-calculi*. Cambridge University Press, 1998.
3. H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.
4. H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabby, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 2, pages 117–309. Oxford University Press, 1992.
5. Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, September 1996.
6. Z. Benaissa, P. Lescanne, and K. H. Rose. Modeling sharing and recursion for weak reduction strategies using explicit substitution. In H. Kuchen and D. Swierstra, editors, *PLILP '96—Eighth International Symposium on Programming Languages: Implementation, Logics and Programs*, number 1140 in Lecture Notes in Computer Science, pages 393–407, Aachen, Germany, September 1996. Springer-Verlag.
7. R. Bloo. *Preservation of Termination for Explicit Substitution*. PhD thesis, Technische Universiteit Eindhoven, 1997. IPA Dissertation Series 1997-05.
8. R. Bloo and J. H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211:375 – 395, 1999.
9. R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95—Computing Science in the Netherlands*, pages 62–72, Koninklijke Jaarbeurs, Utrecht, November 1995.
10. E. Bonelli. Perpetuality in a named lambda calculus with explicit substitutions and some applications. Technical Report RR 1221, LRI, University of Paris-Sud, 1999. to appear in MSCS, 2000.
11. F. Cardone and M. Coppo. Two extension of Curry's type inference system. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Series*, pages 19–75. Academic Press, New York, NY, 1990.
12. M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archive f. math. Logic u. Grundlagenforschung*, 19:139–156, 1978.
13. P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996.
14. H. B Curry and R. Feys. *Combinatory Logic I*. North-Holland, Amsterdam, 1958.

15. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
16. R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *LICS '97—Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 35–46, Warsaw, Poland, June-July 1997. Warsaw University, IEEEC Society Press.
17. S. Ghilezan. Strong normalization and typability with intersection types. *Notre Dame J. Formal Loigc*, 37(1):44–52, 1996.
18. B. Guillaume. The $\lambda-s_e$-calculus does not preserve strong normalisation. *Journal of Functional Programming*, 2000. to appear.
19. H. Herbelin. Explicit substitutions and reducibility. In *WESTAPP 2000, Workshop on explicit substitution*, 2000.
20. F. Kamareddine and A. Ríos. Extending a lambda-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, July 1997.
21. F. Kamareddine and A. Ríos. Relating the lambda-sigma and lambda-s styles of explicit substitutions. *Journal of Logic and Computation*, 10(3), 2000. Special issue on Type Theory and Term Rewriting.
22. F. Kamereddine and R.P. Nederpelt. On stepwise explicit substitutions. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.
23. J.-L. Krivine. *Lambda calculus, types and models*. Ellis Horwood, 1993.
24. D. Leivant. Typing and computational properties of lambda expressions. *Theoretical Computer Science*, 44(1):51–68, 1986.
25. P. Lescanne. From $\lambda\sigma$ to λv: a journey through calculi of explicit substitutions. In Hans-J. Boehm, editor, *POPL '94—21st Annual ACM Symposium on Principles of Programming Languages*, pages 60–69, Portland, Oregon, January 1994. ACM.
26. P. Lescanne and J. Rouyer-Degli. The calculus of explicit substitutions λv. Technical Report RR-2222, INRIA-Lorraine, January 1994.
27. P.-A. Melliès. Typed λ-calculi with explicit substitution may not terminate. In M. Dezani, editor, *TLCA '95—Int. Conf. on Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334, Edinburgh, Scotland, April 1995. Springer-Verlag.
28. P.-A. Melliès. Axiomatic rewriting theory III, a factorisation theorem in rewriting theory. In *Proceedings of the 7th Conference on Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*, pages 49–68. Springer-Verlag, 1997.
29. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, MA, 1996.
30. G. Pottinger. A type assignment for the strongly normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–578. Academic Press, 1980.
31. E. Ritter. Characterising explicit substitutions which preserve termination. In *TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.
32. K.H. Rose. *Operational Reduction Models for Functional Programming Languages*. PhD thesis, DIKU, Universitetsparken 1, DK-2100 København Ø, February 1996. DIKU report 96/1.
33. P. Sallé. Une extension de la théorie des types en λ-calcul. In G. Ausiello and C. Boehm, editors, *Fifth Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 398–410. Springer-Verlag, July 1978.
34. S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.
35. F. van Raamsdonk, P. Severi, M. Heine B. Sørensen, and Hongwei Xi. Perpetual reductions in λ-calculus. *Information and Computation*, 149(2):173–225, 15 March 1999.