An improved system of intersection types for explicit substitutions

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

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

[‡] École Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon 07, FRANCE
E-mail: {Stephane.Lengrand,Pierre.Lescanne}@ens-lyon.fr

Abstract. We characterize those terms which are strongly normalizing in a composition-free calculus of explicit substitutions by defining a suitable type system using intersection types. The key idea is the notion of *available* variable in a term, which is a generalization of the classical notion of free variable.

1 Introduction

An explicit substitutions calculus is a refinement of traditional λ-calculus in which substitution is not treated as a meta-operation on terms but rather as an operation of the calculus itself. The inspiration for such a study is the observation that, in the presence of variable-binding, substitution is a complex operation to define and to implement, so that making substitutions explicit leads to a more pertinent analysis of the correctness and efficiency of compilers, theorem provers, and proof-checkers. Abadi, Cardelli, Curien, and Lévy defined the first calculus of explicit substitutions in [1].

The original motivation for the Abadi-Cardelli-Curien-Lévy system was pragmatic, but there is another point of view one may take on such a calculus, namely that making substitution explicit reflects a more refined analysis of substitution than that of the classical λ-calculus. As historical context we note that Curry and Feys emphasize, in their book [13], 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.*

With this perspective 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. Observe that the classical λ-calculus is a subsystem of explicit substitution systems, defined by the strategy of "eagerly" applying the substitution induced by contracting a β-redex. In this sense explicit substitutions calculi are logically prior to classical λ-calculus, and the study of explicit substitutions represents a deeper examination of the relationship between abstraction and application. This point of view 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.

A fundamental property of classical typed lambda-calculi is strong normalization: no term admits an infinite reduction sequence. Melliès [27] made the somewhat surprising discovery that strong normalization fails even for simply-typed terms of the calculi of [1] and [12].

Given the central place that strong normalization occupies in the theory and application of classical lambda calculus it is important to study this property in systems of explicit substitutions. Melliés' result exploits the existence of a *composition* operator on substitutions, so there are two obvious and complementary research directions. The first is to define classes of reduction strategies in the original calculus which support strong normalization; a notable example of work in this area is that of Eike Ritter [29]. The second direction is to investigate calculi in which substitutions are explicit but composition is absent; the current paper is part of this effort.

Composition-free calculi of explicit substitutions have been studied in [25, 8, 21, 7, 5] among others. Here we work in the composition-free calculus $\lambda\mathtt{x}$ [8] and the calculus $\lambda\mathtt{x}_{gc}$ obtained by adding explicit garbage collection to $\lambda\mathtt{x}$.

In previous work [17, 18] we explored some reduction properties of this system using intersection types. Working with the natural generalization of the classical type systems we were able to characterize the sets of normalizing and head-normalizing terms in terms of typability. But it was shown in [17] that the naive generalization of the classical system did not characterize the strongly normalizing terms. Typable terms were strongly normalizing but the converse fails.

Example 1 Let S be the term $\lambda u.uu$. Consider the terms

$$M_1 \equiv ((\lambda y.z)xx)\langle x = S\rangle \longrightarrow M_2 \equiv z\langle y = xx\rangle\langle x = S\rangle$$

The term M_2 is readily seen to be strongly normalizing. But M_2 is not typable in the system $\mathcal{D}$ of [17]: it is obtained from the (non-SN, hence untypable) term M_1 by contracting a β-redex, and such a contraction does not change the typing behavior of terms under $\mathcal{D}$. Finding a type system characterizing the strongly normalizing terms was left as an open problem in [17].

Main results. In this paper we solve the aforementioned problem: we define an extension $\mathcal{E}$ of system $\mathcal{D}$ which types precisely the strongly normalizing terms. Furthermore when a universal type ω is added the resulting system $\mathcal{E}_\omega$ satisfies the same theorems as those in [17] characterizing the weakly normalizing, head normalizing, and solvable terms. Our claim, then, is that the system presented here — with or without a universal type — is a robust type system appropriate for analyzing reduction properties in explicit substitutions calculi.

The key insight for the solution is a new notion, that of *available* variable occurrence in a term (Definition 3). This is a refinement of the notion of free variable and is the key to extending $\mathcal{D}$. As a corollary of our approach we are able to define a somewhat more general notion of garbage collection than has been studied in the literature of $\lambda\mathtt{x}$ and show that adding a reduction for garbage-collection does not change the set of strongly normalizing terms.

Related work. Independently of the present work, Mariangiola Dezani and Steffen van Bakel [32] have defined a type system which also characterizes the strongly normalizing terms. Their system is different from ours and the technical development of their argument proceeds along different lines. In section 6 we present a comparison of our work with theirs.

The system of intersection types is due to Coppo and Dezani [11]. The fact that the strongly normalizing terms in the classical λ-calculus are precisely the typable terms was first proved in [28]. Other notable works in this area include [24, 31, 19], and the books [23] and [2].

Explicit substitutions calculi without composition typically enjoy the *preservation of strong normalization* property: a pure term is strongly normalizing in the presence of explicit substitutions if it is so under β-reduction [26, 5, 8, 6, 7, 30, 16]. It follows that the classical intersection types system does characterize strong normalization for pure terms. In contrast, the current results provide information about *all* terms. Perhaps more significant is the fact that the proofs here are direct, involving reasoning in the explicit substitutions calculus itself, not passing through the indirection of an argument about β-reduction.

Plan of the paper. Section 2 presents the syntax of $\lambda\mathtt{x}$; section 3 defines the type systems $\mathcal{E}$ and $\mathcal{E}_\omega$, and verifies that the results of [17] extend to system $\mathcal{E}_\omega$. In section 4 we prove that all strongly normalizing terms are typable in system $\mathcal{E}$, and in section 5 we show the converse. There is one

subtlety to be noted: in section 4 we treat reduction without garbage collection; while in section 5 we admit garbage collection. Since it is clear that the strongly normalizing terms in the presence of garbage collection are strongly normalizing in the simpler system we obtain as a corollary the fact that garbage collection does not affect strong normalization.

Our notation is consistent with that of [4], to which we refer the reader for background on the classical λ-calculus.

2 The calculus λx

Definition 2 The set Λx of terms with explicit substitutions is defined as follows :

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

One defines the notions of free and bound variable occurrences in a term as usual. But it turns out that in the presence of explicit substitutions a refinement of the notion of free variable, called *available* variable occurrence, is key.

Definition 3 The *free* variables in a term are:

$$\begin{cases} FV(x) & := \{x\} \\ FV(\lambda x.M) & := FV(M)\backslash\{x\} \\ FV(M\ N) & := FV(M) \cup FV(N) \\ FV(M\langle x = N\rangle) & := (FV(M)\backslash\{x\}) \cup FV(N) \end{cases}$$

The *available* variables in a term are:

$$\begin{cases} AV(x) & := \{x\} & \\ AV(\lambda x.M) & := AV(M)\backslash\{x\} & \\ AV(M\ N) & := AV(M) \cup AV(N) & \\ AV(M\langle x = N\rangle) & := (AV(M)\backslash\{x\}) \cup AV(N) & \text{if } x \in AV(M) \\ AV(M\langle x = N\rangle) & := AV(M) & \text{if } x \notin AV(M) \end{cases}$$

A variable occurrence which is not free is a *bound* occurrence.

For pure terms the notions of freeness and availability coincide. But availability differs from freeness in that the available variables of $M\langle x = N\rangle$, where x is not available in M, are exactly those of M, whereas the free variables in any case are those of M and of N. The intuition is that x is not available just when the term N disappears in the course of fully applying the substitutions in $M\langle x = N\rangle$.

Further discussion of the motivation for defining available variable occurrences will be given after we present our type system. For now we can observe, referring to Example 1, that in the term $z\langle y = xx\rangle$ the variable x is free, but is not available.

From the definitions of $AV(M)$ and $FV(M)$, an easy induction over the structure of M shows that the available variable occurrences in a term are a subset of the free variable occurrences.

In what follows we consider terms up to a α-conversion. Moreover, when one chooses a representative in a term, one does that in such a way that the Barendregt convention [3] is fulfilled: *no variable occurs both free and bound.* Since available variables are free it follows that we may assume that no variable occurs both available and bound.

As usual we often treat *contexts,* terms $C[\ ]$ with a designated variable $[\ ]$ called a *hole*; terms can be "grafted" into the hole with variable-capture permitted (see [3]).

Definition 4 (λx and λx_{gc}) We identify the following reduction rules on λx terms.

$$\begin{array}{lll}
(\lambda x.M)\,A & \longrightarrow M\langle x = A\rangle & \mathsf{B} \\
(M\,N)\langle x = A\rangle & \longrightarrow M\langle x = A\rangle\, N\langle x = A\rangle & \mathsf{App} \\
(\lambda y.M)\langle x = A\rangle & \longrightarrow \lambda y.(M\langle x = A\rangle) & \mathsf{Abs} \\
x\langle x = A\rangle & \longrightarrow A & \mathsf{VarI} \\
y\langle x = A\rangle & \longrightarrow y & \mathsf{VarK} \\
M\langle x = A\rangle & \longrightarrow M \quad \text{if } x \notin AV(M) & \mathsf{gc}
\end{array}$$

The notion of reduction λx is obtained by deleting the rule gc, and the notion of reduction λx_{gc} is obtained by deleting the rule VarK.

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

In contrast with the classical λ-calculus we are considering a rewrite system with several rules, which in fact interact with each other in interesting ways. For example there is a *critical pair* formed by the rules B and App, which is responsible for much of the complexity in analyzing the theory.

Definition 5 (Reduction) Let $l \longrightarrow r$ be a reduction rule; we refer to an instantiation $s(l)$ of l as a *redex*. A (unconstrained) *reduction* is determined by a redex occurrence in a term $C[s(l)]$ and gives rise to the ordered pair $C[s(l)] \longrightarrow C[s(r)]$.

We write $\mathcal{SN}$ for the set of strongly normalizing terms under λx, and $\mathcal{SN}_{\mathsf{gc}}$ for the corresponding set under λx_{gc}.

The notion of garbage collection in this paper is more liberal than that originally defined by Bloo and Rose [8] and treated in [17]: here we define "garbage" in terms of available occurrences rather than free occurrences. Our results will imply that a term is strongly normalizing under λx_{gc} if and only if it is strongly normalizing under λx (a result shown directly in [8] for their notion of garbage-collection).

The following easy observations will be useful later. They apply to each of λx and λx_{gc}.

Lemma 6 *If $M\langle x = N\rangle$ is not a redex then $M \equiv M'\langle y = N'\rangle$. In particular $M\langle x = N\rangle$ is never a normal form.*

We will also need the fact that if we omit rule B then the resulting reduction is strongly normalizing, so that an infinite derivation contains infinitely many applications of rule B.

To prove that strongly normalizing terms are typable, we will induct over the reduction relation, and so we will want to show that the converse of the reduction relation preserves typability. Of course this is not true in full generality, and we restrict attention to reductions following a certain strategy, a *leftmost-outermost* strategy. As discussed in [17] the notion of leftmost reduction is not as straightforward as in classical λ-calculus, in particular the notion there is a non-deterministic strategy. The strategy defined below is a deterministic restriction. It makes sense for each of λx and λx_{gc}, although we make use of it only in Section 4, where we consider only λx.

Definition 7 For any term not in normal form, the *leftmost-outermost* strategy reduces the leftmost-outermost redex, where the leftmost-outermost redex of M is :

$$\begin{cases}
M & \text{if } M \text{ is a redex,} \\
\text{otherwise:} & \\
\text{the leftmost-outermost redex of } N_1 & \text{if } M \equiv N_1\ N_2 \text{ where } N_1 \text{ is not a normal form} \\
\text{the leftmost-outermost redex of } N_2 & \text{if } M \equiv N_1\ N_2 \text{ where } N_1 \text{ is a normal form} \\
\text{the leftmost-outermost redex of } N & \text{if } M \equiv N\langle x = A\rangle \text{ since } N \text{ is not a normal form} \\
\text{the leftmost-outermost redex of } N & \text{if } M \equiv \lambda x.N.
\end{cases}$$

Note that the leftmost-outermost redex in a term $N\langle x = A\rangle$ is never a B-redex.

3 The system $\mathcal{E}$ of intersection types

Definition 8 The set of *types* is inductively defined as follows.

$$\tau_1, \tau_2 \ := \ \sigma \mid \tau_1 \cap \tau_2 \mid \tau_1 \to \tau_2$$

The standard ordering $\leq$ on types is the smallest transitive and reflexive relation such that

$$\begin{cases} \tau_1 \cap \tau_2 \ \leq \ \tau_1 \\ \tau_1 \cap \tau_2 \ \leq \ \tau_2 \\ \text{if } \sigma \leq \tau_1 \text{ and } \sigma \leq \tau_2 \text{ then } \sigma \leq \tau_1 \cap \tau_2 \end{cases}$$

Definition 9 An *environment* is an assignment from variables to types, where each individual assignment is written $(x : \tau)$. Environments are partially ordered as follows.

$$\Gamma \leq \Gamma' \quad \text{iff} \quad (x : \tau') \in \Gamma' \Longrightarrow (\exists \tau)\ (x : \tau) \in \Gamma \text{ and } \tau \leq \tau'$$

Definition 10

$$\begin{aligned} \Gamma_1 \sqcap \Gamma_2 = \{(x : \tau) \mid (x : \tau) \in \Gamma_1 \wedge (\forall \sigma)(x : \sigma) \notin \Gamma_2\} \\ \bigcup \{(x : \tau) \mid (x : \tau) \in \Gamma_2 \wedge (\forall \sigma)(x : \sigma) \notin \Gamma_1\} \\ \bigcup \{(x : \tau_1 \cap \tau_2) \mid (x : \tau_1) \in \Gamma_1 \text{ and } (x : \tau_2) \in \Gamma_2\} \end{aligned}$$

Lemma 11

- $\Gamma_1 \sqcap \Gamma_2 \leq \Gamma_1$ *and* $\Gamma_1 \sqcap \Gamma_2 \leq \Gamma_2$.
- *If* $\Gamma_1 \leq \Gamma$ *and* $\Gamma_2 \leq \Gamma$ *then* $\Gamma_1 \sqcap \Gamma_2 \leq \Gamma$.

Proof. These are routine verifications. ///

Definition 12 A *judgment* is a triple consisting of an environment Γ, a term M, and a type τ. A judgment is *derivable* in system $\mathcal{E}$, denoted $\Gamma \vdash M : \tau$, if this form can be derived by the rules of inference given in Table 1. A term M is *typable* if for some Γ and τ, $\Gamma \ \vdash \ M : \tau$ is derivable.

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

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

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

$$\mathsf{drop}\ \frac{\Gamma \vdash M : \tau \qquad \text{A typable}}{\Gamma \vdash M\langle x = A\rangle : \tau}\quad x \notin AV(M)$$

$$\cap\text{-I}\ \frac{\Gamma \vdash M : \tau_1 \qquad \Gamma \vdash M : \tau_2}{\Gamma \vdash M : \tau_1 \cap \tau_2} \qquad \cap\text{-E}\ \frac{\Gamma \vdash M : \tau_1 \cap \tau_2}{\Gamma \vdash M : \tau_i}\quad i \in \{1, 2\}$$

Table 1. Typing rules for $\mathcal{E}$.

The innovation in the type system here is the presence of the rule drop. The type system of [17] had no such rule: the point of view taken there was that a closure $M\langle x = N\rangle$ should always have the same typing behavior as the B-redex $(\lambda x.M)N$ which yields it. This is a plausible strategy since B-reduction involves no (immediate) erasing of subterms, even when x is not free in M; and indeed the resulting system — in the presence of a universal type — yields the expected characterizations of head-normalizing and leftmost-normalizing terms. But as we have seen in Example 1 this system failed to provide a characterization of the strongly normalizing terms. This example makes clear that we must allow the type system to distinguish between certain B-redexes and their contractions.

Perhaps one's first instinct is to note that in Example 1 the input variable of the B-redex in M_1 does not occur free in the function body (*i.e.*, we have a "K-redex" in classical λ-calculus). This suggests modifying the cut-rule to obtain one which, when typing $M\langle x = N\rangle$ with x not free in M, relaxes the typing hypothesis for N to merely ask that it be typable under *some* environment. This seems particularly appropriate since it echoes the hypotheses of the Subject Expansion Theorem in treatments of intersection types for classical λ-calculus. But such a rule doesn't work: it is still too restrictive. For example, the reader can easily check that the term $M_2' \equiv x\langle y = xx\rangle\langle x = S\rangle$ cannot be typed in such a system, but is clearly strongly normalizing. This last example should motivate our notion of *available* variable occurrence and the corresponding typing rule drop.

A good exercise at this point is to check that the terms M_2 and M_2' can be typed in system $\mathcal{E}$. On another hand, notice that rule cut has no side condition, therefore when $x \notin AV(M)$ and $\Gamma \vdash A : \sigma$, one can freely use cut or drop.

The following are some elementary properties of the type system.

Lemma 13

1. *If $x \notin AV(M)$, then for all types σ, $\Gamma \vdash M : \tau$ if and only if $\Gamma, (x : \sigma) \vdash M : \tau$.*
2. *If $\tau \leq \tau'$ then $(x : \tau) \vdash x : \tau'$*
3. *If $\Gamma \leq \Gamma'$ and $\Gamma' \vdash M : \tau$ then $\Gamma \vdash M : \tau$*

Proof. The first is a structural induction over type derivations. The second is an induction over the definition of $\leq$ as the reflexive transitive closure given by Definition 8. The third is proved by structural induction on the typing tree, using part 2 for the start case. ///

Adding a universal type to $\mathcal{E}$

The system $\mathcal{E}$ is obtained from the system $\mathcal{D}$ of [17] by adding the rule drop. The system $\mathcal{D}_\omega$ is the extension of $\mathcal{D}$ obtained by adding a universal type ω; in [17] characterizations of the head-normalizing and leftmost-normalizing terms of λx were obtained in terms of typability in $\mathcal{D}_\omega$.

The main result of this paper is that typability in system $\mathcal{E}$ serves to characterize the strongly-normalizing terms of λx, and therefore that the rule drop captures an important aspect of reduction in explicit substitutions calculi. But an important question to raise at this point is whether the addition of rule drop behaves well in the presence of a universal type. In particular we may ask whether the normalization theorems of [17] still hold in the presence of drop.

Definition 14 The type system $\mathcal{E}_\omega$ is obtained from system $\mathcal{E}$ by adding the type constant ω and the rule:

$$\omega\text{-I} \quad \frac{}{\Gamma \vdash M : \omega}$$

Since system $\mathcal{D}_\omega$ is a subsystem of $\mathcal{E}_\omega$ it is clear that the following results follow from the corresponding results for $\mathcal{D}_\omega$.

- if M is head normalizing then M is typable in system $\mathcal{E}_\omega$ with a non-trivial type, and
- if M is normalizing then M is typable in system $\mathcal{E}_\omega$ with a type not involving ω.

The following theorem is sufficient to establish the converses of these results.

Theorem 15 *Suppose $\Gamma \vdash M : \tau$ in system $\mathcal{E}_\omega$. Then $\Gamma \vdash M : \tau$ in system $\mathcal{D}_\omega$ as well.*

Proof. By induction over typing derivations; it suffices to show that an application of rule drop can be simulated in $\mathcal{D}_\omega$. So suppose

$$\mathsf{drop}\quad \frac{\Gamma \vdash M : \tau \qquad \text{A typable}}{\Gamma \vdash M\langle x = A\rangle : \tau}\quad x \notin AV(M)$$

By induction we can derive $\Gamma \vdash M : \tau$ in $\mathcal{D}_\omega$, so certainly $\Gamma, x : \omega \vdash M : \tau$. By ω-I, $\Gamma \vdash A : \omega$ in $\mathcal{D}_\omega$, so we have

$$\mathsf{cut}\quad \frac{\Gamma, x : \omega \vdash M : \tau \qquad \vdash A : \omega}{\Gamma \vdash M\langle x = A\rangle : \tau}$$

in $\mathcal{D}_\omega$, as desired. ///

For completeness we state here the results relating reduction properties of terms with their typing properties in system $\mathcal{E}_\omega$. The results follow from those in [17] together with Theorem 15.

The following definitions are due to Cardone and Coppo [10]: A type is *proper* if it has no positive occurrence of ω. A type is *trivial* if it can be generated by the following rules: (i) ω is trivial, (ii) If σ is trivial and θ is any type, then $\theta \to \sigma$ is trivial, (iii) If σ and τ are trivial, then $\sigma \cap \tau$ is trivial.

Theorem 16 *Let M be a closed term. The following are equivalent.*

1. *M is typable with a non-trivial type in system $\mathcal{E}_\omega$.*
2. *M is head-normalizing in the calculus $\lambda\mathtt{x}_{gc}$.*
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 $MX_1 \cdots X_n = \lambda x.x$.*

Theorem 17 *Let M be a closed term. The following are equivalent.*

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

In Theorem 17 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. The notion of leftmost reduction in $\lambda\mathtt{x}$ is rather subtle. We refer the reader to [17] for details but we may say here that leftmost reduction is a non-deterministic strategy, and to say that a term is leftmost-normalizing above is to say that *any* sequence of leftmost reductions is guaranteed to reach a normal form. Note that leftmost reduction is different from the (deterministic) leftmost-outermost strategy of Definition 7, introduced in this paper for technical reasons only.

4 Typing strongly normalizing terms

In this section "reduction" will always mean "$\lambda\mathtt{x}$ reduction," that is, we do not consider garbage collection. Our goal is to prove, by induction over the leftmost-outermost strategy, that strongly normalizing terms are typable.

Definition 18 When M is not strongly normalizing we set $h(M) := \infty$. Otherwise we define $h(M) := \max\{h(N) + 1 \mid M \longrightarrow N\}$.

This definition makes sense since the reduction $\longrightarrow$ is finitely-branching, so that a term M which is strongly normalizing under $\longrightarrow$ will have only finitely many N such that $M \longrightarrow\!\!\!\!\!\rightarrow N$, and the definition of $h(M)$ involves taking the maximum of a finite set. The height $h(M)$ of a term M is the length of the longest derivation to normal form, and in particular the height of a normal form is 0. Note that $M \longrightarrow N \Rightarrow h(M) > h(N)$ and $M = C[N] \Rightarrow h(M) \geq h(N)$.

Normal forms in $\Lambda\mathtt{x}$ are the same as in classical λ-calculus, and the type system $\mathcal{E}$ is an extension of the standard system of intersection types for classical λ-calculus. The following proposition is thus an immediate consequence of the classical result.

Proposition 19 *If M is a normal form then M is typable in system $\mathcal{E}$.*

4.1 From restricted judgments to general judgments

In Section 4.2 we show that if $M \longrightarrow N$ by a leftmost-outermost reduction, we assign a type to M built from the type assigned to N. If the last rule of the typing tree is not an intersection rule, then it is directly determined by the structure of the term. The following technical lemma will allow us, as we treat the various cases for N, to avoid explicitly considering the situation when the last typing rule is $\cap$-I or $\cap$-E.

Definition 20 We write $\Gamma \vDash M : \tau$ if $\Gamma \vdash M : \tau$ is provable with a typing tree whose root (i.e. its last rule) is neither $\cap$-I nor $\cap$-E.

Lemma 21 *Let M,M' be terms and Γ be an environment.*

$$\begin{array}{c} \forall \tau\,(\Gamma \vDash M : \tau \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \tau) \\ \Downarrow \qquad\qquad (21.1) \\ \forall \tau\,(\Gamma \vdash M : \tau \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \tau) \end{array}$$

and

$$\begin{array}{c} \forall \tau\,(\Gamma \vDash M : \tau \Rightarrow \Gamma \vdash M' : \tau) \\ \Downarrow \qquad\qquad (21.2) \\ \forall \tau\,(\Gamma \vdash M : \tau \Rightarrow \Gamma \vdash M' : \tau) \end{array}$$

Proof. The proof is in the appendix. ///

Lemma 22 *1. If $x \in AV(P)$, $\Gamma \vdash P\langle x = A\rangle : \rho \Rightarrow \exists \tau_P \mid \Gamma, (x : \tau_P) \vdash P : \rho$ and $\Gamma \vdash A : \tau_P$.*
2. If $x \notin AV(P)$, $\Gamma \vdash P\langle x = A\rangle : \rho \Rightarrow \Gamma \vdash P : \rho$ and A is typable.
3. If $x \notin AV(P)$, $(\forall \tau)\ \Gamma \vdash P\langle x = A\rangle : \rho \Rightarrow \Gamma, (x : \tau) \vdash P : \rho$.

Proof. The proof is in the appendix. ///

4.2 Subject expansion

It is convenient to identify a general property we will refer to throughout this section as we induct over the height of terms:

$$\boxed{M \in \mathcal{SN} \ \text{ and } \ (\forall P \in \Lambda\mathtt{x})\ h(P) < h(M) \Longrightarrow P \text{ is typable} \qquad \mathbb{P}(M)}$$

We wish to prove that for any term whose leftmost-outermost redex is $s(l)$, if it reduces to a typable term, then the term itself is typable. For such a term, we consider the context $C[\,]$ such that the term is $C[s(l)]$. The proof lies on a structural induction on contexts $C[\,]$.

For this induction to work, we need a somewhat stronger statement.

- A term is assigned the same type as this of the term obtained by contracting its leftmost-outermost redex, except when the rule is B and the term is an abstraction.
- This assignment is made in the same environment, except when the rule is B, in which case the environment is more constrained.

Notice that the initial case of the induction concerns terms whose root can be reduced (so the type is preserved, since such terms are not abstractions) and is treated in the following Lemma.

Lemma 23 (Root reduction) *Given a rule $l \longrightarrow r$ and an instance $s(l)$ of l, assume $\mathbb{P}(s(l))$.*

$$\Gamma \vdash s(r) : \tau \Rightarrow \Gamma \vdash s(l) : \tau \quad \textit{if the rule is not (B)} \quad (23.1)$$

$$\Gamma \vdash s(r) : \tau \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash s(l) : \tau \quad \textit{if the rule is (B)} \quad (23.2)$$

Proof. The proof is in the appendix. ///

Lemma 24 (Leftmost-outermost reduction) *Let $l \longrightarrow r$ be a rule and let $s(l)$ be an instance of l. Let $C[\ ]$ be a context such that $s(l)$ is the leftmost-outermost redex of $C[s(l)]$. Assume $\mathbb{P}(s(l))$ and $\Gamma \vdash C[s(r)] : \tau$.*

If the rule is not (B) : $\Gamma \vdash C[s(l)] : \tau$. *(24.1)*

If the rule is (B) : $\exists\, \Gamma' \leq \Gamma \mid \begin{cases} \exists \tau' \mid \Gamma' \vdash C[s(l)] : \tau' & \text{if } C[\ ] = \lambda x.C'[\ ] \\ \Gamma' \vdash C[s(l)] : \tau & \text{if } C[\ ] \neq \lambda x.C'[\ ] \end{cases}$ *(24.2)*

Proof. The proof is by structural induction on $C[\ ]$.

If $C[\ ] = [\ ]$, this is exactly Lemma 23.

If the rule is not (B), the environment and the type of $C[s(l)]$ are the same as of $C[s(r)]$. The proof is easy using the same typing tree.

The proof is harder when the rule is (B).

$\boxed{C[\ ] = \lambda x.C'[\ ]}$ According to Definition 7, $s(l)$ is also the leftmost-outermost redex of $C'[s(l)]$. We can apply the induction hypothesis to $C'[\ ]$, namely

$$\Gamma, (x : \tau_1) \vdash C'[s(r)] : \tau_2 \implies \exists\, \Gamma', \tau_1', \tau_2' \mid \Gamma' \leq \Gamma \text{ and } (\Gamma', (x : \tau_1')) \vdash C'[s(l)] : \tau_2'.$$

On the other hand (Definition 20)

$$\Gamma \vDash \lambda x.C'[s(r)] : \tau \implies \exists \tau_1, \tau_2 \mid (\Gamma, (x : \tau_1)) \vdash C'[s(r)] : \tau_2 \text{ and } \tau = \tau_1 \to \tau_2.$$

hence combining the above statements

$$\Gamma \vDash \lambda x.C'[s(r)] : \tau \implies \exists\, \Gamma', \tau_1', \tau_2' \mid \Gamma' \leq \Gamma \text{ and } \Gamma' \vdash \lambda x.C'[s(l)] : \tau_1' \to \tau_2'.$$

Clearly $\Gamma \vdash C[s(r)] : \tau \Rightarrow \exists \tau'' \mid \Gamma \vDash C[s(r)] : \tau''$, therefore combining everything we get

$$\Gamma \vdash C[s(r)] : \tau \implies \exists\, \Gamma', \tau' \mid \Gamma' \leq \Gamma \text{ and } \Gamma' \vdash C[s(l)] : \tau'.$$

$\boxed{C[\ ] = C'[\ ]\, N}$ $C'[\ ]$ cannot be an abstraction, otherwise $C'[s(l)]\ N$ would be a redex and $s(l)$ would not be the leftmost-outermost redex of $C[s(l)]$. Definition 7 says that $s(l)$ is the leftmost-outermost of $C'[s(l)]$. By Definition 20

$$\Gamma \vDash C'[s(r)]\ N : \tau_2 \Rightarrow \exists \tau_1 \mid \Gamma \vdash C'[s(r)] : \tau_1 \to \tau_2 \text{ and } \Gamma \vdash N : \tau_1.$$

By induction hypothesis on $C'[\ ]$:

$$\Gamma \vdash C'[s(r)] : \tau_1 \to \tau_2 \implies \exists\, \Gamma' \mid \Gamma' \leq \Gamma \text{ and } \Gamma' \vdash C'[s(l)] : \tau_1 \to \tau_2$$

and by $\Gamma' \leq \Gamma$ and $\Gamma \vdash N : \tau_1 \implies \Gamma' \vdash N : \tau_1$ we get

$$\Gamma \vDash C'[s(r)]\ N : \tau_2 \implies \exists \tau_1, \exists\, \Gamma' \mid \Gamma' \leq \Gamma \text{ and } \Gamma' \vdash C'[s(l)] : \tau_1 \to \tau_2 \text{ and } \Gamma' \vdash N : \tau_1$$

therefore $\Gamma \vDash C'[s(r)]\ N : \tau_2 \implies \exists\, \Gamma' \mid \Gamma' \leq \Gamma$ and $\Gamma' \vdash C'[s(l)]\ N : \tau_2$.
And by Lemma 21.1 $\Gamma \vdash C[s(r)] : \tau_2 \Rightarrow \exists\, \Gamma' \mid \Gamma' \leq \Gamma$ and $\Gamma' \vdash C[s(l)] : \tau_2$.

$\boxed{C[\ ] = N\, C'[\ ]}$ N is not an abstraction, otherwise $N\ C'[s(l)]$ would be a redex and $s(l)$ would not be the leftmost-outermost redex of $C[s(l)]$. From Definition 7, we know that N is a normal form. Hence N is a λ-free normal form and $s(l)$ is the leftmost-outermost redex of $C'[s(l)]$. The induction hypothesis on $C'[\ ]$ yields :

$$\Gamma \vdash C'[s(r)] : \tau_1 \implies \exists\, \Gamma', \tau_1' \mid \Gamma' \leq \Gamma \text{ and } \Gamma' \vdash C'[s(l)] : \tau_1'.$$

Assume $\Gamma \vdash N\ C'[s(r)] : \tau_2$ and apply Lemma 21.1 with type $\tau_1' \to \tau_2$ for N, then there exists Γ'' such that $\Gamma'' \vdash N : \tau_1' \to \tau_2$, hence

$$\Gamma' \sqcap \Gamma'' \vdash N : \tau_1' \to \tau_2 \text{ and } \Gamma' \sqcap \Gamma'') \vdash C'[s(r] : \tau_1'$$

Finally, using transitivity and naming Γ''' as $\Gamma' \sqcap \Gamma''$

$$\Gamma \vdash N\ C'[s(r)] : \tau_2 \implies \exists\, \Gamma''' \mid \Gamma''' \vdash N\ C'[s(l)] : \tau_2.$$

$\boxed{C[\ ] = C'[\ ]\langle x = A\rangle}$ can never happen, since the left-outermost cannot both belong to $C'[s(l)]$ and be a (B) redex.

///

In the above proof it is important to notice that the reduction *is* the leftmost-outermost one. Indeed this way we escape difficult cases, namely when the term to reduce is a closure or when $C[\,] = C'[\,]\,N$ and we have a B-redex.

Theorem 25 *If $M \in \mathcal{SN}$ then M is typable.*

Proof. By induction on the height of terms: if M is a normal form, we are done. If not, then it can be reduced to a term N by a leftmost-outermost reduction. By induction every P such $h(P) < h(M)$ is typable, thus $\mathbb{P}(M)$ is fulfilled. Especially $h(N) < h(M)$, hence N is typable. Then, using Lemma 24, we get M is typable. ///

5 Characterization of strongly normalizing terms

In this section "reduction" will always mean "$\lambda \mathtt{x}_{gc}$ reduction," that is, we allow garbage collection. Our goal is to prove that typable terms are strongly normalizing (we use $\mathcal{SN}_{\mathsf{gc}}$ to refer to the set of such terms. As described in the introduction, a consequence of this result and the result of the previous section is the fac that garbage collection does not change the set of strongly normalizing terms. See Theorem 31.

5.1 Saturated Sets and the Soundness theorem

We only slightly modify the proof made in [17], in changing FV to AV when required namely in definition sat-gc.

Definition 26 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 2.

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

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

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

Table 2. $\mathcal{X}$-saturated sets

Lemma 27 *$\mathcal{SN}_{gc}$ is $\mathcal{SN}_{gc}$-saturated.*

Proof. The proof relies of Corollary 3.6 of [17], which itself relies on Lemma 3.5 there. But in this lemma, FV can be changed safely into AV, since the statement $x_i \notin FV(M_i)$ is used for insuring that rule gc can be applied, but in our formulation of gc, FV has been precisely changed into AV. ///

Definition 28 We define $\mathcal{S}_\tau$ for each type τ as

- $\mathcal{S}_t := \mathcal{SN}_{\mathsf{gc}}$ for each type variable t.
- $\mathcal{S}_{\tau\cap\sigma} := \mathcal{S}_\tau \bigcap \mathcal{S}_\sigma$.
- $\mathcal{S}_{\sigma\to\tau} := \{F \in \Lambda\mathsf{x} \mid (\forall A \in \mathcal{S}_\sigma)\ (F\ A) \in \mathcal{S}_\tau\}$.

Lemma 29 *Then for any type τ, $\mathcal{S}_\tau \subset \mathcal{SN}_{gc}$, and $\mathcal{S}_\tau$ is $\mathcal{SN}_{gc}$-saturated.*

Proof. The proof is entirely done in [17] by structural induction on types. Although the first statement is completely independent, the initial case of the second one relies on the former lemma. ///

Theorem 30 (Soundness theorem for $\mathcal{SN}_{\mathsf{gc}}$) *For any terms $M, A_1, \ldots, A_n$, suppose*

- $(x_1 : \sigma_1), \ldots, (x_n : \sigma_n) \vdash M : \tau$
- $\forall i \in [1, n], A_i \in \mathcal{S}_{\sigma_i}$
- $\forall i \in [1, n], \forall j \geq 0, x_{i+j} \notin AV(A_i)$

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

Proof. The proof in [17] consists in a structural induction on the typing tree of M. Most of it need not to be modified (the weakening of the hypothesis -AV instead of FV- is balanced by the change in the definition of saturated sets), we only have to proceed with a new case due to the addition of the drop rule.

Let $\Gamma := (x_1 : \sigma_1), \ldots, (x_n : \sigma_n)$ and assume $\Gamma \vdash M\langle x = A\rangle : \tau$ comes by the drop rule from $\Gamma \vdash M : \tau$ and $\Gamma' \vdash A : \sigma$ for some Γ' and σ.

Applying the induction hypothesis to $\Gamma \vdash M : \tau$ we get $M\langle x_1 = A_1\rangle \ldots \langle x_n = A_n\rangle \in \mathcal{S}_\tau$. Applying the induction hypothesis to $\Gamma' \vdash A : \sigma$ and using Lemma 29, we get $A \in \mathcal{SN}_{\mathsf{gc}}$. Since $\mathcal{S}_\tau$ is $\mathcal{SN}_{\mathsf{gc}}$-saturated, we can apply rule sat-gc which yields $M\langle x = A\rangle\langle x_1 = A_1\rangle \ldots \langle x_n = A_n\rangle \in \mathcal{S}_\tau$. ///

Theorem 31 *The following are equivalent.*

1. *M is typable in system $\mathcal{E}$.*
2. *$M \in \mathcal{SN}_{gc}$.*
3. *$M \in \mathcal{SN}$.*

Proof. Part 1 implies part 2 by Theorem 30 together with Lemma 29. Clearly 2 implies 3. Theorem 25 yields 3 implies 1. ///

The fact that garbage-collection does not change the set of strongly normalizing terms was originally established by Rose [30] for the slightly more restricted original notion of garbage-collection.

6 The type system of Dezani and van Bakel

As mentioned in the introduction, Dezani and van Bakel [32] have independently found a typing characterization of the strongly normalizing terms. The innovation in their typing system also involves a new rule for typing closures $M\langle x = N\rangle$, but rather than attending directly to the way x occurs in M, as we do, they focus on whether M can be typed in an environment not binding x. Specifically, they use the following rule in place of our drop (let us write $\vdash_{DvB}$ for typability in the system of Dezani and van Bakel).

$$\mathsf{K\text{-}cut}\quad \frac{\Gamma \vdash_{DvB} M : \tau \qquad \Delta \vdash_{DvB} N : \sigma}{\Gamma \vdash_{DvB} M\langle x = N\rangle : \tau}\quad x \text{ not in } \mathrm{dom}(\Gamma)$$

They prove that typability in their system characterizes the strongly normalizing terms, so clearly their system types the same terms as ours. In this section we give a direct proof that the system of Dezani and van Bakel is equivalent to ours, in the strong sense that it types the same terms as ours, with the same types.

Lemma 32 *If $\Gamma \vdash P : \tau$ then $\Gamma' \vdash P : \tau$ where Γ' is Γ restricted to the variables of $AV(P)$. Furthermore the derivation from Γ' is no longer than the given one from Γ.*

Proof. An easy induction over typing derivations. ///

Proposition 33 *If $\Gamma \vdash P : \tau$ then $\Gamma' \vdash_{DvB} P : \tau$. where Γ' is Γ restricted to the available variables of P.*

Proof. Induct over derivations. The non-trivial case is drop in our system

$$\mathsf{drop}\quad \frac{\Gamma \vdash M : \tau \qquad \Delta \vdash N : \sigma}{\Gamma \vdash M\langle x = N\rangle : \tau}\quad x \text{ not available in } M$$

By Lemma 32 we can replace $\Gamma \vdash M : \tau$ by $\Gamma' \vdash M : \tau$ where Γ' does not contain x. Then apply the induction hypothesis to the two premises, to obtain $\Gamma' \vdash_{DvB} M\langle x = N\rangle : \tau$ by a K-cut.

///

Lemma 34 *If $\Gamma \vdash_{DvB} P : \tau$ and x is available in P then x is in $dom(\Gamma)$.*

Proof. Induct over derivations; consider cases according to the form of P. The interesting case is when P is $M\langle y = N\rangle$ and the derivation ends with I-cut or K-cut.

So suppose $\Gamma \vdash_{DvB} M\langle y = N\rangle : \tau$ with x available in $M\langle y = N\rangle$. Note that since available variables are always free and since y is bound in the term $M\langle y = N\rangle$, we may assume without loss of generality that x is not y.

We identify two cases: either y is available in M or not. If not, then x must be available in M itself. Then no matter which rule (I-cut or K-cut) was used, the induction hypothesis applies to the premise typing M, and (using the fact that x is not y in the I-cut case) we conclude that x is in $\mathrm{dom}(\Gamma)$.

On the other hand, if y is available in M then x could be available in M or available in N. But the typing rule used must have been I-cut. Since the induction hypothesis applies to each premise, we conclude (again perhaps noting that x is not y) that x is in $\mathrm{dom}(\Gamma)$. ///

Proposition 35 *If $\Gamma \vdash_{DvB} P : \tau$ then $\Gamma \vdash P : \tau$.*

Proof. Induct over derivations. The non-trivial case is K-cut. Referring to their rule as given at the beginning of this section, suppose that $\Gamma \vdash_{DvB} M\langle x = N\rangle : \tau$ with N typable and x not in $\mathrm{dom}(\Gamma)$. By induction hypothesis $\Gamma \vdash M\langle x = N\rangle : \tau$ and N is typable in our system. By Lemma 34 x is not available in M, so our drop rule applies to give $\Gamma \vdash M\langle x = N\rangle : \tau$ as desired. ///

Theorem 36 *$\Gamma \vdash M : \tau$ if and only if $\Gamma \vdash_{DvB} M : \tau$.*

Proof. One direction is immediate from Proposition 35; the other follows from Proposition 33 and the "weakening" property of the system of Dezani and van Bakel (cf. [32]). ///

7 Conclusions and future work

We have defined a new intersection-types system $\mathcal{E}$ for terms of the explicit substitutions calculus λx and shown that typability in $\mathcal{E}$ characterizes strong normalization. We defined a new notion of garbage-collection and proved that a term is strongly normalizing in the core calculus if and only if it is strongly normalizing in the presence of garbage collection.

Using results from [17] we have shown that the system $\mathcal{E}_\omega$ obtained by adding a universal type smoothly characterizes the weakly normalizing terms and the head-normalizing, or solvable terms.

We have also given a direct proof of equivalence with a different system, found independently by Dezani and van Bakel, which also characterizes strong normalization in λx.

Future work. Intersection types have long been known to be a robust tool for exploring properties of classical λ-terms: Krivine's book [23] has many examples of this; recent work includes [9, 22, 15, 14, 20]. There is much more work to be done in applying intersection types to calculi of explicit substitutions. One intriguing idea is to attempt to better understand the reduction properties of calculi with substitution-composition with the help of these type systems. Another, largely unexplored, area of investigation is semantics for explicit substitutions calculi: of course intersection types have proven to be a very fruitful tool for studying semantics of the classical λ-calculus.

Acknowledgements

The authors are grateful to Steffen van Bakel, Norman Danner, Mariangiola Dezani, Simona Ronchi della Rocca, and Silvia Ghilezan for encouragement and instructive conversations.

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. R. Bloo. *Preservation of Termination for Explicit Substitution.* PhD thesis, Technische Universiteit Eindhoven, 1997. IPA Dissertation Series 1997-05.
7. R. Bloo and J. H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211:375 – 395, 1999.
8. R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95—Computing Science in the Netherlands*, pages 62–72, Koninklijke Jaarbeurs, Utrecht, November 1995.
9. A. Bucciarelli, S. De Lorenzis, A. Piperno, and I. Salvo. Some computational properties of intersection types. In *14th Symposium on Logic in Computer Science (LICS'99)*, pages 109–118, Washington - Brussels - Tokyo, July 1999. IEEE.
10. F. Cardone and M. Coppo. Two extension of Curry's type inference system. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Series*, pages 19–75. Academic Press, New York, NY, 1990.
11. M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19:139–156, 1978.
12. 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.
13. H. B Curry and R. Feys. *Combinatory Logic I.* North-Holland, Amsterdam, 1958.
14. R. Davies and F. Pfenning. Intersection types and computational effects. In *Proceedings of the ACM Sigplan International Conference on Functional Programming (ICFP-00)*, volume 35.9 of *ACM Sigplan Notices*, pages 198–208, N.Y., September 18–21 2000. ACM Press.
15. M. Dezani-Ciancaglini, F. Honsell, and Y. Motohama. Compositional characterizations of lambda-terms using intersection types (extended abstract). In *MFCS: Symposium on Mathematical Foundations of Computer Science*, 2000.
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. D. Dougherty and P. Lescanne. Reductions, intersection types, and explicit substitutions (extended abstract). In S. Abramsky, editor, *TLCA 2001—5th Int. Conf. on Typed Lambda Calculus and Applications*, volume 2044 of *Lecture Notes in Computer Science*, pages 121–135, Krakow, Poland, May 2001. Springer-Verlag.
18. D. Dougherty and P. Lescanne. Reductions, intersection types, and explicit substitutions. *Mathematical Structures in Computer Science*, to appear.

19. S. Ghilezan. Strong normalization and typability with intersection types. *Notre Dame J. Formal Loigc*, 37(1):44–52, 1996.
20. S. Ghilezan. Full intersection types and topologies in lambda calculus. *JCSS: Journal of Computer and System Sciences*, 62, 2001.
21. 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.
22. A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 161–174, New York, NY, USA, 1999. ACM Press.
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 $\lambda\upsilon$: a journey through calculi of explicit substitutions. In Hans-J. Boehm, editor, *POPL '94—21st Annual ACM Symposium on Principles of Programming Languages*, pages 60–69, Portland, Oregon, January 1994. ACM.
26. P. Lescanne and J. Rouyer-Degli. The calculus of explicit substitutions $\lambda\upsilon$. 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. 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.
29. E. Ritter. Characterising explicit substitutions which preserve termination. In *TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.
30. 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.
31. S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.
32. S. van Bakel and M. Dezani-Ciancaglini. Characterizing strong normalization for explicit substitutions. In *Latin American Theoretical INformatics - LATIN*, 2002. to appear.

A Appendix

This appendix is addressed to the referees. We supply some proofs which could not be given in the main body due to space constraints.

Lemma 21 *Let M,M' be terms and Γ be an environment.*

$$\begin{array}{c} \forall \tau\,(\Gamma \vDash M : \tau \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \tau) \\ \Downarrow \\ \forall \tau\,(\Gamma \vdash M : \tau \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \tau) \end{array} \qquad (21.1)$$

and

$$\begin{array}{c} \forall \tau\,(\Gamma \vDash M : \tau \Rightarrow \Gamma \vdash M' : \tau) \\ \Downarrow \\ \forall \tau\,(\Gamma \vdash M : \tau \Rightarrow \Gamma \vdash M' : \tau) \end{array} \qquad (21.2)$$

Proof. Assume $\forall \rho\,(\Gamma \vDash M : \rho \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \rho)$. Fix τ and assume $\Gamma \vdash M : \tau$. The proof of 21.1 is by structural induction on the typing tree of $\Gamma \vdash M : \tau$.

- If the last rule is neither $\cap$-I nor $\cap$-E then $\Gamma \vDash M : \tau$ and the conclusion comes from $\forall \rho\,(\Gamma \vDash M : \rho \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \rho)$.
- if the rule is $\cap$-I : $\dfrac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash M : \tau_2}{\Gamma \vdash M : \tau_1 \cap \tau_2}$ then the induction hypothesis yields: $\exists\, \Gamma_1 \leq \Gamma \mid \Gamma_1 \vdash M' : \tau_1$ and $\exists\, \Gamma_2 \leq \Gamma \mid \Gamma_2 \vdash M' : \tau_2$. If we set $\Gamma' := \Gamma_1 \sqcap \Gamma_2$, then $\Gamma' \vdash M' : \tau_1 \cap \tau_2$.
- if the rule is $\cap$-E : $\dfrac{\Gamma \vdash M : \tau_1 \cap \tau_2}{\Gamma \vdash M : \tau_i}$ (with $i \in \{1,2\}$), then the induction hypothesis yields: $\exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash M' : \tau_1 \cap \tau_2$ then we have $\Gamma' \vdash M' : \tau_i$.

The proof of 21.2 (i.e., when $\Gamma' = \Gamma$) is exactly the same. ///

Lemma 22 *1. If $x \in AV(P)$, $\Gamma \vdash P\langle x = A\rangle : \rho \Rightarrow \exists \tau_P \mid \Gamma, (x : \tau_P) \vdash P : \rho$ and $\Gamma \vdash A : \tau_P$.*
2. If $x \notin AV(P)$, $\Gamma \vdash P\langle x = A\rangle : \rho \Rightarrow \Gamma \vdash P : \rho$ and A is typable.
3. If $x \notin AV(P)$, $(\forall \tau)\ \Gamma \vdash P\langle x = A\rangle : \rho \Rightarrow \Gamma, (x : \tau) \vdash P : \rho$.

Proof. We prove the first two parts simultaneously, by induction on the typing tree.

- If $\Gamma \vDash P\langle x = A\rangle : \rho$, we know that the last rule is either cut or drop, then their definitions induce the expected result.
- The rule is $\cap$-I : $\dfrac{\Gamma \vdash P\langle x = A\rangle : \rho_1 \quad \Gamma \vdash P\langle x = A\rangle : \rho_2}{\Gamma \vdash P\langle x = A\rangle : \rho_1 \cap \rho_2}$, therefore
  - if $x \in AV(P)$, the induction hypothesis yields
    - $\exists \tau_P \mid \Gamma, (x : \tau_P) \vdash P : \rho_1$ and $\Gamma \vdash A : \tau_P$
    - $\exists \tau'_P \mid \Gamma, (x : \tau'_P) \vdash P : \rho_2$ and $\Gamma \vdash A : \tau'_P$

    then $\Gamma, (x : \tau_P \cap \tau'_P) \vdash P : \rho_1 \cap \rho_2$ and $\Gamma \vdash A : \tau_P \cap \tau'_P$
  - if $x \notin AV(P)$, the induction hypothesis yields :
    - $\Gamma \vdash P : \rho_1$ and A is typable
    - $\Gamma \vdash P : \rho_2$ and A is typable

    then $\Gamma \vdash P : \rho_1 \cap \rho_2$ and A is typable.
- The rule is $\cap$-E : $\dfrac{\Gamma \vdash M : \rho_1 \cap \rho_2}{\Gamma \vdash M : \rho_i}$ (with $i \in \{1,2\}$), therefore
  - if $x \in AV(P)$, the induction hypothesis yields :
    - $\exists \tau_P \mid \Gamma, (x : \tau_P) \vdash P : \rho_1 \cap \rho_2$ and $\Gamma \vdash A : \tau_P$

    then $\Gamma, (x : \tau_P) \vdash P : \rho_i$ and $\Gamma \vdash A : \tau_P$
  - if $x \notin AV(P)$, the induction hypothesis yields :
    - $\Gamma \vdash P : \rho_1 \cap \rho_2$ and A is typable

    then $\Gamma \vdash P : \rho_i$ and A is typable.

Part 3 follows from part 2, using part 1 of Lemma 13. ///

Lemma 23 (Root reduction) *Given a rule $l \longrightarrow r$ and an instance $s(l)$ of l, assume $\mathbb{P}(s(l))$.*

$$\Gamma \vdash s(r) : \tau \Rightarrow \Gamma \vdash s(l) : \tau \quad \textit{if the rule is not (B)} \quad (23.1)$$

$$\Gamma \vdash s(r) : \tau \Rightarrow \exists\, \Gamma' \leq \Gamma \mid \Gamma' \vdash s(l) : \tau \quad \textit{if the rule is (B)} \quad (23.2)$$

Proof. Let us reason by case on the rule used.

- $(\lambda x.M)\ A \longrightarrow M\langle x = A\rangle$
  We wish to prove $\Gamma \vdash M\langle x = A\rangle : \tau \Rightarrow \exists\, \Gamma'' \leq \Gamma \mid \Gamma'' \vdash (\lambda x.M)\ A : \tau$. Using Lemma 21.1, we only have to prove : $(\forall \tau)\, \Gamma \vDash M\langle x = A\rangle : \tau \Rightarrow \exists\, \Gamma'' \leq \Gamma \mid \Gamma'' \vdash (\lambda x.M)\ A : \tau$. Therefore assume $\Gamma \vDash M\langle x = A\rangle : \tau$.
  - If $x \notin AV(M)$, we have $\Gamma \vdash M : \tau$ and $\exists\, \Gamma', \tau' \mid \Gamma' \vdash A : \tau'$. From Lemma 13.1, $\Gamma, (x : \tau') \vdash M : \tau$ which yields $\Gamma \vdash \lambda x.M : \tau' \to \tau$. Hence $\exists\, \Gamma', \tau' \mid \Gamma \vdash \lambda x.M : \tau' \to \tau$ and $\Gamma' \vdash A : \tau'$
    If we set Γ'' to be $\Gamma \sqcap \Gamma' \leq \Gamma$ we get $\Gamma'' \vdash \lambda x.M : \tau' \to \tau$ and $\Gamma'' \vdash A : \tau'$ which entails $\Gamma'' \vdash (\lambda x.M)\ A : \tau$
  - If $x \in AV(M)$ we have $\exists\ \tau' \mid \Gamma, (x : \tau') \vdash M : \tau$ and $\Gamma \vdash A : \tau'$, so $\exists\, \tau' \mid \Gamma \vdash \lambda x.M : \tau' \to \tau$ and $\Gamma \vdash A : \tau'$ which entails $\Gamma \vdash (\lambda x.M)\ A : \tau$.
- *If the top rules are (App) and (Abs),* we have to prove : $\Gamma \vdash s(r) : \tau \Rightarrow \Gamma \vdash s(l) : \tau$. Using Lemma 21.2, we only have to prove : $\forall \tau\, \Gamma \vDash s(r) : \tau \Rightarrow \Gamma \vdash s(l) : \tau$. Therefore assume $\Gamma \vDash s(r) : \tau$ and let look at the details for each rule.
- $(M\ N)\langle x = A\rangle \longrightarrow M\langle x = A\rangle\ N\langle x = A\rangle$. In this case, the assumption we have made is $\Gamma \vDash M\langle x = A\rangle\ N\langle x = A\rangle : \tau$ which induces

  $$\exists\, \tau' \ \mid\ \Gamma \vdash M\langle x = A\rangle : \tau' \to \tau \text{ and } \Gamma \vdash N\langle x = A\rangle : \tau'.$$

  If $x \notin AV(M)$ and $x \notin AV(N)$, we apply Lemma 22.2 with $P := M$ and $P := N$, which induces $\Gamma \vdash M : \tau' \to \tau$ and $\Gamma \vdash N : \tau'$, as well as A *is typable.* Consequently, $\Gamma \vdash M\ N : \tau$ and finally $\Gamma \vdash (M\ N)\langle x = A\rangle : \tau$.
  If $x \in AV(M)$ or $x \in AV(N)$, we have to prove

  $$\exists \tau'' \ \mid\ \Gamma \vdash A : \tau'' \ \text{ and } \ \Gamma, (x : \tau'') \vdash M : \tau' \to \tau \ \text{ and } \ \Gamma, (x : \tau'') \vdash N : \tau'$$

  (which induces $\Gamma \vdash (M\ N)\langle x = A\rangle : \tau$). Let us look at each case.
  - If $x \in AV(M)$ and $x \notin AV(N)$, applying Lemma 22.1 with $P := M$ and $\rho := \tau' \to \tau$, we get τ_M. Taking τ'' to be τ_M and using Lemma 22.3 with $P := N$ and $\rho := \tau'$ we get the result.
  - If $x \notin AV(M)$ and $x \in AV(N)$, applying Lemma 22.1 with $P := N$ and $\rho := \tau'$ we get τ_N. As above, taking τ'' to be τ_N and using Lemma 22.3 with $P := M$ and $\rho := \tau' \to \tau$ we get the result.
  - If $x \in AV(M)$ and $x \in AV(N)$, applying Lemma 22.1 with $P := N$ and $\rho := \tau'$, we get τ_N, and applying it again with $P := M$ and $\rho := \tau' \to \tau$, we get τ_M. Then, we get four judgments
    * $\Gamma \vdash A : \tau_M$,
    * $\Gamma \vdash A : \tau_N$,
    * $\Gamma, (x : \tau_M) \vdash M : \tau' \to \tau$ and
    * $\Gamma, (x : \tau_N) \vdash N : \tau'$.

    If we set τ'' to $\tau_M \cap \tau_N$ we get the result.
- $(\lambda y.M)\langle x = A\rangle \longrightarrow \lambda y.(M\langle x = A\rangle)$ in which we suppose $y \notin AV(A)$ and $x \neq y$ then $x \notin AV(M) \Leftrightarrow x \notin AV(\lambda y.M))$. In this case, the assumption we have made is $\Gamma \vDash \lambda y.(M\langle x = A\rangle) : \tau$ which induces $\exists \tau_1, \tau_2 \mid \Gamma, (y : \tau_1) \vdash M\langle x = A\rangle : \tau_2$ and $\tau = \tau_1 \to \tau_2$.
  If $x \in AV(M)$ we have to prove $\exists \tau'' \mid \Gamma \vdash A : \tau''$ and $\Gamma, (x : \tau''), (y : \tau_1) \vdash M : \tau_2$. Applying Lemma 22.1 with $P := M$ and $\rho := \tau_2$ and with the environment $\Gamma, (y : \tau_1)$ instead of Γ we get $\Gamma, (y : \tau_1), (x : \tau_M) \vdash M : \tau_2$ and $\Gamma, (y : \tau_1) \vdash A : \tau_M$. But as we assumed $y \notin AV(A)$, we can apply Lemma 13.1 and with $\tau'' = \tau_M$, we get $\Gamma, (x : \tau''), (y : \tau_1) \vdash M : \tau_2$ and $\Gamma \vdash A : \tau''$
  If $x \notin AV(M)$ we have to prove A *typable* and $\Gamma, (y : \tau_1) \vdash M : \tau_2$. Applying Lemma 22.2 with $P := M$ and $\rho := \tau_2$ and with the environment $\Gamma, (y : \tau_1)$ we get the result.

- $x\langle x = A\rangle \longrightarrow A$. Assume $\Gamma \vdash A : \tau$. Clearly $\Gamma, (x : \tau) \vdash x : \tau$ and $\Gamma \vdash x\langle x = A\rangle : \tau$.
- $M\langle x = A\rangle \longrightarrow M$, if $x \notin AV(M)$. Assume $\Gamma \vdash M : \tau$. The redex to be reduced lies at the root, so we have $h(M\langle x = A\rangle) > h(A)$. By $\mathbb{P}(M\langle x = A\rangle)$, we conclude that A is typable. Therefore $\Gamma \vdash M\langle x = A\rangle : \tau$.

///