AN IMPROVED SYSTEM OF INTERSECTION TYPES FOR EXPLICIT SUBSTITUTIONS

Dan Dougherty
Department of Mathematics and Computer Science, Wesleyan University
Middletown, CT 06459 USA
ddougherty@wesleyan.edu

Stephane Lengrand and Pierre Lescanne
Ecole Normale Superieure de Lyon
46, Alle d'Italie, 69364 Lyon 07, FRANCE
{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 [Abadi et al., 1991].

A fundamental property of classical typed lambda-calculi is strong normalization: no term admits an infinite reduction sequence. Melliès [Melliès, 1995] made the somewhat surprising discovery that strong normalization fails even for simply-typed terms of the calculi of [Abadi et al., 1991] and [Curien et al., 1996].

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 [Ritter, 1999]. 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 [Lescanne, 1994, Bloo and Rose, 1995, Kamareddine and Ríos, 1997, Bloo and Geuvers, 1999, Benaissa et al., 1996] among others. Here we work in the composition-free calculus $\lambda\mathtt{x}$ [Bloo and Rose, 1995] and the calculus $\lambda\mathtt{x}_{gc}$ obtained by adding explicit garbage collection to $\lambda\mathtt{x}$.

In previous work [Dougherty and Lescanne, 2001, Dougherty and Lescanne, pear] 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 [Dougherty and Lescanne, 2001] 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 syntax and reduction rules of the calculus are given in section 2.) The term M_2 is readily seen to be strongly normalizing. But M_2 is not typable in the system $\mathcal{D}$ of [Dougherty and Lescanne, 2001]: 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 [Dougherty and Lescanne, 2001].

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 [Dougherty and Lescanne, 2001] 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.

2. The calculus $\lambda\mathtt{x}$

Definition 2 The set $\Lambda\mathtt{x}$ of terms with explicit substitutions is defined as:

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

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 set of *free* variables $FV(M)$ is the same as in [Dougherty and Lescanne, 2001] and the set of *available* variables $AV(M)$ in a term M is

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

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 <u>and</u> 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 [Barendregt, 1984] 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 [Barendregt, 1984]).

Definition 4 ($\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$) We identify the following reduction rules on $\lambda\mathtt{x}$ terms.

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

The notion of reduction $\lambda\mathtt{x}$ 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.

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 $\lambda\mathtt{x}$, and $\mathcal{SN}_{\mathsf{gc}}$ for the corresponding set under $\lambda\mathtt{x}_{gc}$.

The notion of garbage collection in this paper is more liberal than that originally defined by Bloo and Rose [Bloo and Rose, 1995] and treated in [Dougherty and Lescanne, 2001]: here we define "garbage" in terms of available occurrences rather than free occurrences. Our results will imply that a term is strongly normalizing under $\lambda\mathtt{x}_{gc}$ if and only if it is strongly normalizing under $\lambda\mathtt{x}$ (a result shown directly in [Bloo and Rose, 1995] for their notion of garbage-collection). The following easy observations will be useful later. They apply to each of $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$.

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 [Dougherty and Lescanne, 2001] 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 $\lambda\mathtt{x}$ and $\lambda\mathtt{x}_{gc}$, although we make use of it only in Section 4, where we consider only $\lambda\mathtt{x}$.

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, written $\mathsf{lo}(M)$ is

M	if M is a redex
$\mathsf{lo}(N_1)$	if $M \equiv N_1\ N_2$ where N_1 is not a normal form
$\mathsf{lo}(N_2)$	if $M \equiv N_1\ N_2$ where N_1 is a normal form
$\mathsf{lo}(N)$	if $M \equiv N\langle x = A\rangle$
$\mathsf{lo}(N)$	if $M \equiv \lambda x.N$

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

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

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

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

$$\tau_1 \cap \tau_2 \ \leq \ \tau_1 \qquad \tau_1 \cap \tau_2 \ \leq \ \tau_2 \qquad \text{if } \sigma \leq \tau_1 \text{ and } \sigma \leq \tau_2 \text{ then } \sigma \leq \tau_1 \cap \tau_2$$

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 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.

The innovation in the type system here is the presence of the rule drop. The type system of [Dougherty and Lescanne, 2001] 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

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

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

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

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

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

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

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 11

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 $\Gamma \vdash M : \tau$*

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

The system $\mathcal{E}$ is obtained from the system $\mathcal{D}$ of [Dougherty and Lescanne, 2001] by adding the rule drop. The system $\mathcal{D}_\omega$ is the extension of $\mathcal{D}$ obtained by adding a universal type ω; in [Dougherty and Lescanne, 2001] characterizations of the head-normalizing and leftmost-normalizing terms of $\lambda\mathtt{x}$ 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 $\lambda\mathtt{x}$, 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 [Dougherty and Lescanne, 2001] still hold in the presence of drop.

Definition 12 The type system $\mathcal{E}_\omega$ is obtained from system $\mathcal{E}$ by adding the type constant ω and the judgement $\Gamma \vdash M : \omega$ as an axiom.

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 $\mathcal{D}_\omega$. *1. If M is head normalizing then M is typable in system $\mathcal{E}_\omega$ with a non-trivial type. 2. 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 13 *Suppose $\Gamma \vdash M : \tau$ in system $\mathcal{E}_\omega$. Then $\Gamma \vdash M : \tau$ in system $\mathcal{D}_\omega$ as well.*

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 [Dougherty and Lescanne, 2001] together with Theorem 13.

The following definitions are due to Cardone and Coppo [Cardone and Coppo, 1990]: 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 14 *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 $\lambda\mathtt{x}_{gc}$.

Theorem 15 *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 leftmost-normalizing in the calculus $\lambda\mathtt{x}_{gc}$.

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 16 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 $M \twoheadrightarrow N$, 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 17 *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 N. If in every of those cases we can assign the same type to M, then we can do it for every type, whether or not the last rule is $\cap$-I or $\cap$-E. This will allow us, as we treat the various cases for N, to avoid explicitly considering the situation when the last typing rule is an intersection rule.

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:

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

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. The proof is omitted for lack of space.

Lemma 18 (Root reduction) *Given a rule $l \longrightarrow r$ and an instance $s(l)$ of l, assume $\mathbb{P}(s(l))$.*

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

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

Lemma 19 (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) : $\Gamma \vdash C[s(l)] : \tau$.

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

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 20 *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 19, 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 fact that garbage collection does not change the set of strongly normalizing terms. See Theorem 26.

We only slightly modify the proof made in [Dougherty and Lescanne, 2001], in changing FV to AV when required namely in definition sat-gc.

Definition 21 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.

Lemma 22 *$\mathcal{SN}_{\mathsf{gc}}$ is $\mathcal{SN}_{\mathsf{gc}}$-saturated.*

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

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

$$\text{sat-App}\ \frac{(U\langle x=A\rangle)(V\langle x=A\rangle)\langle \boldsymbol{z}=\boldsymbol{S}\rangle\boldsymbol{T}}{(UV)\langle x=A\rangle\langle \boldsymbol{z}=\boldsymbol{S}\rangle\boldsymbol{T}}$$

$$\text{sat-comp}\ \frac{M\langle y=Q\rangle\langle x=P\langle y=Q\rangle\rangle\langle \boldsymbol{z}=\boldsymbol{S}\rangle\boldsymbol{T}}{M\langle x=P\rangle\langle y=Q\rangle\langle \boldsymbol{z}=\boldsymbol{S}\rangle\boldsymbol{T}}$$

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

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

Proof. The proof relies of Corollary 3.6 of [Dougherty and Lescanne, 2001], 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 23 We define $\mathcal{S}_\tau$for each type τ as

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

Lemma 24 *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 [Dougherty and Lescanne, 2001] 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 25 (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 [Dougherty and Lescanne, 2001] 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 24, 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 26 *The following are equivalent.*

1 M *is typable in system* $\mathcal{E}$.

2 $M \in \mathcal{SN}_{\mathsf{gc}}$.

3 $M \in \mathcal{SN}$.

Proof. Part 1 implies part 2 by Theorem 25 together with Lemma 24. Clearly 2 implies 3. Theorem 20 yields 3 implies 1. ///

The fact that garbage-collection does not change the set of strongly normalizing terms was originally established by Rose [Rose, 1996] 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 [van Bakel and Dezani-Ciancaglini, 2002] 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).

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

They prove that typability in their system characterizes the strongly normalizing terms, so clearly their system types the same terms as ours. In fact 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.

Theorem 27 $\Gamma \vdash M : \tau$ *if and only if* $\Gamma \vdash_{DvB} M : \tau$.

7. Conclusions and future work

We have defined a new intersection-types system $\mathcal{E}$ for terms of the explicit substitutions calculus $\lambda\mathtt{x}$ 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 [Dougherty and Lescanne, 2001] we can show 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 can also give a direct proof of equivalence with a different system, found independently by Dezani and van Bakel, which also characterizes strong normalization in $\lambda\mathtt{x}$.

Future work. Intersection types have long been known to be a robust tool for exploring properties of classical λ-terms: Krivine's book [Krivine, 1993] has many examples of this; recent work includes [Bucciarelli et al., 1999, Kfoury and Wells, 1999, Dezani-Ciancaglini et al., 2000, Davies and Pfenning, 2000, Ghilezan, 2001]. 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. For lack of space we removed the bibliography which can found in the full version at `http://www.ens-lyon.fr/~plescann/PUBLICATIONS/avail.ps`.

References

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

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

Benaissa, Z., Briaud, D., Lescanne, P., and Rouyer-Degli, J. (1996). $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722.

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

Bloo, R. and Rose, K. H. (1995). Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95— Computing Science in the Netherlands*, pages 62–72, Koninklijke Jaarbeurs, Utrecht.

Bucciarelli, A., Lorenzis, S. D., Piperno, A., and Salvo, I. (1999). Some computational properties of intersection types. In *14th Symposium on Logic in Computer Science (LICS'99)*, pages 109–118, Washington - Brussels - Tokyo. IEEE.

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

Curien, P.-L., Hardin, T., and Lévy, J.-J. (1996). Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397.

Davies, R. and Pfenning, F. (2000). 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. ACM Press.

Dezani-Ciancaglini, M., Honsell, F., and Motohama, Y. (2000). Compositional characterization of lambda -terms using intersection types. In *Mathematical Foundation of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 304–314. Springer-Verlag. extended abstract.

Dougherty, D. and Lescanne, P. (2001). Reductions, intersection types, and explicit substitutions (extended abstract). In Abramsky, S., 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. Springer-Verlag.

Dougherty, D. and Lescanne, P. (to appear). Reductions, intersection types, and explicit substitutions. *Mathematical Structures in Computer Science*.

Ghilezan, S. (2001). Full intersection types and topologies in lambda calculus. *JCSS: Journal of Computer and System Sciences*, 62(1):1–14.

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

Kfoury, A. J. and Wells, J. B. (1999). 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. ACM Press.

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

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

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

Ritter, E. (1999). Characterising explicit substitutions which preserve termination. In *TLCA'99, Int. Conf. on Typed Lambda Calculus and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag.

Rose, K. (1996). *Operational Reduction Models for Functional Programming Languages*. PhD thesis, DIKU, Universitetsparken 1, DK-2100 København Ø. DIKU report 96/1.

van Bakel, S. and Dezani-Ciancaglini, M. (2002). Characterizing strong normalization for explicit substitutions. In *Latin American Theoretical INformatics - LATIN*. to appear.