Explicit Substitutions for the λΔ-Calculus *

Gilles Barthe[1] Fairouz Kamareddine[2] Alejandro Ríos[2]

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, the Netherlands, email gilles@cwi.nl
[2] University of Glasgow, Department of Computing Science, 17 Lilybank Gardens, Glasgow G12 8QQ, Scotland, UK, email {fairouz,rios}@dcs.gla.ac.uk

Abstract. *The λΔ-calculus is a λ-calculus with a control-like operator whose reduction rules are closely related to normalisation procedures in classical logic. We introduce λΔexp, an explicit substitution calculus for λΔ, and study its properties. In particular, we show that λΔexp preserves strong normalisation, which provides us with the first example –moreover a very natural one indeed– of explicit substitution calculus which is not structure-preserving and has the preservation of strong normalisation property. One particular application of this result is to prove that the simply typed version of λΔexp is strongly normalising.*
In addition, we show that Plotkin's call-by-name continuation-passing style translation may be extended to λΔexp and that the extended translation preserves typing. This seems to be the first study of CPS translations for calculi of explicit substitutions.

1 Introduction

Explicit substitutions were introduced by Abadi, Cardelli, Curien and Lévy in [1] as a bridge between λ-calculus and its implementation. The fundamental idea behind explicit substitutions is simple: in order to provide a full account of the computations involved in computing a λ-term, one must describe a method to compute substitutions. Since the seminal work of Abadi, Cardelli, Curien and Lévy, explicit substitutions have developed into a subject of their own, finding further applications e.g. in proof-search [29], unification [11], representation of incomplete proofs [23, 21] and proof theory [16].

In this paper, we generalise some of the results on explicit substitutions for λ-calculi to classical λ-calculi, i.e. λ-calculi with control-like structures. More precisely, we consider a specific calculus with a control-like operator, called λΔ [28], and define its explicit substitution variant λΔexp. Then we prove that the λΔexp enjoys some important properties:

- λΔexp preserves strong normalisation, i.e. every strongly normalising λΔ-term is strongly normalising with respect to the reduction relation of λΔexp;
- the simply typed λΔexp calculus is strongly normalizing;

* This work is supported by NWO and the British council under UK/Dutch joint scientific research project JRP240 and EPSRC grant GR/K 25014.

- $\lambda\Delta$exp may be translated to λexp –a named explicit λ-calculus– using an extension of the continuation-passing style translation;
- the CPS translation maps simply typable $\lambda\Delta$exp-terms to simply typable λexp-terms and generalises Kolmogorov's double-negation translation.

The motivation for this work is three-fold:

1. control-like operators play a crucial role in functional programming languages, such as LISP [30], SML [2], Scheme [12], etc. We will only be able to claim that explicit substitutions provide a bridge between higher-order rewriting systems and their implementation if the theory of explicit substitutions can be extended –among other– to control-like operators;
2. control-like operators and explicit substitutions both have applications in theorem proving and proof theory. (See e.g. [24] for applications of control-like operators in theorem proving and [5, 14, 24] for applications of control-like operator in proof theory.) The former are used in classical theorem proving and the latter to represent incomplete proofs. By studying explicit substitutions with control operators, we lay the foundations for a classical theorem prover with the ability to handle incomplete proofs and for a classical proof theory based on explicit substitutions.
3. control-like operators fundamentally differ from λ-calculus in that they are not structure-preserving in the sense of [9]. Hence the results of [9] do not apply. Yet we will show that the decency method [7] can be adapted to our setting. This constitutes the first study of explicit substitutions for non-structure-preserving calculi and suggests the possibility of extending the results of [9] to a large and useful class of combinatory rewrite systems.

Organisation of the paper In Section 2, we introduce the $\lambda\Delta$-calculus and state some of its properties. In Section 3, we extend the $\lambda\Delta$-calculus with explicit substitutions. In Section 4, we establish the confluence and preservation of strong normalisation (PSN) of the $\lambda\Delta$exp-calculus. We use the interpretation method [15] to show confluence and the decency method to establish PSN [7]. We also show that the structure preserving method of [9] does not apply to the $\lambda\Delta$exp-calculus. In Section 5 we introduce the simply typed version of $\lambda\Delta$exp and show that it has the desirable properties such as subject reduction and strong normalisation. In Section 6, we present the first study of CPS-translations for calculi of explicit substitutions by providing a CPS-translation for $\lambda\Delta$exp and showing its soundness. In Section 7, we discuss related work. We conclude in Section 8.

Prerequisites and terminology We assume some basic familiarity with λ-calculus [4] and abstract rewriting [19]. We let $\lhd$ denote the subterm relation and $\rightarrow_R$ denote the compatible closure of a relation R –compatibility is defined as usual. The transitive and reflexive-transitive closures of $\rightarrow_R$ are denoted by $\rightarrow_R^+$ and $\twoheadrightarrow_R$ respectively. Finally, we let $\mathrm{SN}(R)$ denote the set of strongly normalising terms w.r.t. $\rightarrow_R$.

2 The $\lambda\Delta$-calculus

Control operators are programming constructs which allow the programmer to have more direct control over the evaluation of a program. In the late 80's, Griffin [14] observed that control operators could be simply typed by the classical axiom of double negation. After Griffin's discovery, there has been a great interest λ-calculi with control-like structures. The $\lambda\Delta$-calculus is such calculus. More precisely, the $\lambda\Delta$-calculus is an extension of the λ-calculus with a binding double negation operator Δ whose computational behavior is closely related to normalisation procedures for classical natural deduction [27] (and of course to reduction rules for control operators).

The following definition is taken from [28].

Definition 1

1. *The set T of (pure) terms is given by the abstract syntax:*

$$T \;=\; V \mid TT \mid \lambda V.T \mid \Delta V.T \qquad \text{with } V = \{x_n : n \in \mathbb{N}\}$$

   where λ and Δ are binding operators.
2. *Meta-substitution $.[./.]$, free and bound variables are defined as usual. We let $\mathsf{FV}(a)$ and $\mathsf{BV}(a)$ denote respectively the sets of free and bound variables of a term a.*
3. *β-reduction $\rightarrow_\beta$ is defined as the compatible closure of*

$$(\lambda x.a)\, b \quad \rightarrow_\beta \quad a[b/x]$$

4. *μ-reduction $\rightarrow_\mu$ is defined as $\rightarrow_{\mu_1} \cup \rightarrow_{\mu_2} \cup \rightarrow_{\mu_3}$ where μ_i-reduction for $1 \leq i \leq 3$ is defined to be the compatible closure of the corresponding i-rule:*

$$\begin{array}{lll} (\Delta x.a)\, b & \rightarrow_{\mu_1} \Delta y.a[\lambda w.y\ (w\ b)/x] & \text{if } y, w \notin \mathsf{FV}(b), y \neq w \\ \Delta x.x\ a & \rightarrow_{\mu_2} a & \text{if } x \notin \mathsf{FV}(a) \\ \Delta x.x\ (\Delta y.x\ a) & \rightarrow_{\mu_3} a & \text{if } x, y \notin \mathsf{FV}(a) \end{array}$$

5. $\rightarrow_{\beta\mu} = \rightarrow_\beta \cup \rightarrow_\mu$.

For motivations and explanations of the Δ-operator, we refer the reader to [28]. We shall briefly mention however that the rule μ_1 is what makes the Δ-operator into a control one. Note that μ_1, does not destroy the control nature of the term. After application, a Δ-term remains a Δ-term. μ_2 acts like an η-rule and together with μ_3 allows to define a catch and throw mechanism.

We let $x, y, z, w, \ldots$ range over V and $a, b, c, \ldots$ range over T and $\mathcal{O}$ to range over $\{\lambda, \Delta\}$. For the sake of hygiene, we consider terms modulo α-conversion –generalised over Δ– and assume Barendregt's variable convention [4].

The following proposition is taken from [28].

Proposition 2 *$\rightarrow_{\beta\mu}$ is confluent (CR).*

Finally, we define the norm $\beta\mu$-norm $\beta\mu(a)$ of a pure term a as the maximal number of $\beta\mu$-reduction steps in a reduction starting from a. It is finite if $a \in \mathsf{SN}(\beta\mu)$ and infinite otherwise. The norm of a term will be used in Section 4.

3 The $\lambda\Delta$exp-calculus

The $\lambda\Delta$exp-calculus is a named calculus of explicit substitutions for $\lambda\Delta$.

Definition 3

1. *The set T^e of terms of the $\lambda\Delta$exp-calculus is given by the abstract syntax:*

$$T^e \;=\; V \mid T^eT^e \mid \lambda V.T^e \mid \Delta V.T^e \mid T^e[V := T^e] \quad \textit{with } V = \{x_n : n \in \mathbb{N}.\}$$

   where $\lambda, \Delta, .[. := .]$ are binding operators. Free and bound variables are defined in the obvious way.
2. *$\underline{\beta}$-reduction $\rightarrow_{\underline{\beta}}$ is defined as the compatible closure of*

$$(\lambda x.a)\; b \quad \rightarrow_{\underline{\beta}} \quad a[x := b]$$

3. *$\underline{\mu}$-reduction $\rightarrow_{\underline{\mu}}$ is defined as $\rightarrow_{\underline{\mu_1}} \cup \rightarrow_{\underline{\mu_2}} \cup \rightarrow_{\underline{\mu_3}}$ where $\underline{\mu_i}$-reduction for $1 \leq i \leq 3$ is defined to be the compatible closure of the corresponding i-rule:*

$$\begin{array}{lll}
(\Delta x.a)\; b & \rightarrow_{\underline{\mu_1}} \Delta y.a[x := \lambda w.y\;(w\; b)] & \textit{if } y, w \notin \mathsf{FV}(b), y \neq w \\
\Delta x.x\; a & \rightarrow_{\underline{\mu_2}} a & \textit{if } x \notin \mathsf{FV}(a) \\
\Delta x.x\;(\Delta y.x\; a) & \rightarrow_{\underline{\mu_3}} a & \textit{if } x, y \notin \mathsf{FV}(a)
\end{array}$$

4. *σ-reduction $\rightarrow_\sigma$ is defined as the compatible closure of*

$$\begin{array}{lll}
x[x := b] & \rightarrow_\sigma b & \\
y[x := b] & \rightarrow_\sigma y & \textit{if } x \neq y \\
(a\; a')[x := b] & \rightarrow_\sigma (a[x := b])\;(a'[x := b]) & \\
(\mathcal{O}y.a)[x := b] & \rightarrow_\sigma \mathcal{O}y.(a[x := b]) & \textit{if } y \notin \mathsf{FV}(b)
\end{array}$$

5. *$\rightarrow_{\underline{\beta\mu\sigma}} = \rightarrow_{\underline{\beta}} \cup \rightarrow_{\underline{\mu}} \cup \rightarrow_\sigma$ and $\rightarrow_{\underline{\beta\mu_i}} = \rightarrow_{\underline{\beta}} \cup \rightarrow_{\underline{\mu_i}}$ for $1 \leq i \leq 3$.*

Again we let $a, b, c, \ldots$ range over T^e. The variable convention, α-conversion, meta-substitution, *etc* are generalised in the obvious way. In particular,

$$\mathsf{FV}(a[x := b]) = \mathsf{FV}(b) \cup (\mathsf{FV}(a) \setminus \{x\})$$

Definition 4 *The set $\sigma\mathsf{FV}(a)$ of* substitutable free variables *of a term a is defined inductively as follows:*

$$\begin{array}{rl}
\sigma\mathsf{FV}(x) &= \{x\} \\
\sigma\mathsf{FV}(ab) &= \sigma\mathsf{FV}(a) \cup \sigma\mathsf{FV}(b) \\
\sigma\mathsf{FV}(\mathcal{O}x.a) &= \sigma\mathsf{FV}(a) \setminus \{x\} \\
\sigma\mathsf{FV}(a[x := b]) &= \begin{cases} \sigma\mathsf{FV}(a) & \textit{if } x \notin \mathsf{FV}(a) \\ (\sigma\mathsf{FV}(a) \setminus \{x\}) \cup \sigma\mathsf{FV}(b) & \textit{if } x \in \mathsf{FV}(a) \end{cases}
\end{array}$$

We conclude this section by noting that $\lambda\Delta$exp contains λexp as a subcalculus. The latter is a named explicit λ-calculus, called λx in [8], and obtained from $\lambda\Delta$exp by leaving out Δ.

4 Confluence and preservation of Strong Normalisation

In this section, we show that the $\lambda\Delta$exp-calculus enjoys confluence and preservation of strong normalisation.

4.1 Confluence

Confluence is proved as usual, using the interpretation method of [10, 15].

Lemma 5 *Let $a, b \in T^e$. The following holds:*

1. *$\rightarrow_\sigma$ is SN and CR. Hence, every term $c \in T^e$ has a unique σ-normal form, denoted $\sigma(c)$.*
2. *$\sigma(ab) = \sigma(a)\sigma(b)$, $\sigma(\lambda x.a) = \lambda x.\sigma(a)$, $\sigma(\Delta x.a) = \Delta x.\sigma(a)$, $\sigma(a[x := b]) = \sigma(a)[\sigma(b)/x]$.*
3. *Projection: If $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} b$ then $\sigma(a) \twoheadrightarrow_{\beta\mu} \sigma(b)$.*
4. *Simulation: for pure terms a, b, if $a \rightarrow_{\beta\mu} b$ then $a \twoheadrightarrow^+_{\underline{\beta\mu\sigma}} b$.*

Proof: Analogous to the proofs of the corresponding results for λexp [8]. We just remark that the function used to prove SN should be here extended with $h(\Delta x.a) = h(a) + 1$. □

Theorem 6 *The $\lambda\Delta$exp-calculus is confluent.*

Proof: If $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} b_1$ and $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} b_2$ then by Lemma 5, $\sigma(a) \twoheadrightarrow_{\beta\mu} \sigma(b_i)$, for $i \in \{1, 2\}$. By CR of $\lambda\Delta$, there exists c such that $\sigma(b_i) \twoheadrightarrow_{\beta\mu} c$, and by Lemma 5 $\sigma(b_i) \twoheadrightarrow_{\underline{\beta\mu\sigma}} c$. Hence, $b_i \twoheadrightarrow_{\underline{\beta\mu\sigma}} c$. □

4.2 Preservation of strong normalisation

Every term is $\underline{\beta\mu\sigma}$-strongly normalising if the σ-normal forms of its subterms are $\beta\mu$-strongly normalising.

Lemma 7 *If $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$ and $b \lhd a$, then $\sigma(b) \in \mathsf{SN}(\beta\mu)$.*

Proof: If $\sigma(b) \notin \mathsf{SN}(\beta\mu)$, then $b \notin \mathsf{SN}(\underline{\beta\mu\sigma})$ as $b \twoheadrightarrow_\sigma \sigma(b)$ and we use Lemma 5.4. Absurd as $b \lhd a$ and $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$. □

Corollary 8 *If a is a pure term such that $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$, then $a \in \mathsf{SN}(\beta\mu)$.*

Proof: If a is pure, $\sigma(a) = a$. □

In other words, $\mathsf{SN}(\underline{\beta\mu\sigma}) \cap T \subseteq \mathsf{SN}(\beta\mu)$. The question arises if the converse holds, i.e. whether $\mathsf{SN}(\beta\mu) \subseteq \mathsf{SN}(\underline{\beta\mu\sigma})$.

Definition 9

1. *A term* $a \in T$ *obeys the* preservation of strong normalisation *(PSN) property if* $a \in \mathsf{SN}(\beta\mu) \Longrightarrow a \in \mathsf{SN}(\underline{\beta\mu\sigma})$.
2. *A term* $a \in T^e$ *obeys the* generalised preservation of strong normalisation *(GPSN) property if* $(\forall b \lhd a.\sigma(b) \in \mathsf{SN}(\beta\mu)) \Longrightarrow a \in \mathsf{SN}(\underline{\beta\mu\sigma})$.

The GPSN property is a mild generalization of the PSN property.[3] In our view, the GPSN property is more fundamental than the PSN property for two reasons:

1. the GPSN property applies to all terms, not only the pure ones;
2. for most typed λ-calculi with explicit substitutions, strong normalisation is an immediate consequence of the GPSN property and of strong normalisation of the standard calculus without explicit substitutions.

We shall prove that the $\lambda\Delta$exp-calculus has the GPSN property using the decency technique of [7] –the technique was introduced to prove that λexp has the PSN property. First, we start with some technical definitions.

Definition 10

1. *A substitution item* $[x := b]$ *is* superfluous *in* a *if* $x \notin \sigma\mathsf{FV}(c)$ *for every* $c[x := b] \lhd a$.
2. *A reduction* $a \to_{\underline{\beta\mu\sigma}} b$ *is* superfluous *if the contracted redex in* a *occurs in a superfluous substitution item* $[x := d]$.

Superfluous reduction plays a role similar to the internal reduction notions of [6, 18] –but the two notions are different from each other. The following is a refinement of Lemma 5.

Lemma 11 *If* $a \to_{\underline{\beta\mu}} b$ *is not superfluous, then* $\sigma(a) \twoheadrightarrow^+_{\beta\mu} \sigma(b)$.

Proof: By induction on the structure of a. □

The following definition of *decent* term is central to the GPSN proof. Note that every $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$ is decent and every decent term is decent of order n.

Definition 12

1. *A term* a *is called* decent *if for every* $[x := b]$ *in* a, $b \in \mathsf{SN}(\underline{\beta\mu\sigma})$.
2. *A term* a *is called* decent of order n *if for every* $[x := b]$ *in* a, $b \in \mathsf{SN}(\underline{\beta\mu\sigma})$ *or* $\beta\mu(\sigma(b)) < n$.

Finally, the following notion of *ancestor* gives a full characterisation of how a substitution item might have been generated. This notion aims to achieve similar conditions to those used in the backtracking lemmas of [6, 18] in the minimal derivation method. Note that we use "$)a$" to denote an application item. For example, in $(\lambda x.a)b$ the application item is $)b$.[4]

Definition 13 *For a reduction* $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} a'$, *we define the notion of* the ancestor *of a substitution item in* a' *as follows:*

[3] It is easy to show that a pure term obeys PSN iff it obeys GPSN.

[4] One can even go further as in [17] by calling λx the λ item but this is not needed here.

1. *If* $a \to_{\underline{\beta\mu\sigma}} a'$ *and* $b = b'$ *or if* $b \to_{\underline{\beta\mu\sigma}} b'$ *and* $a = a'$ *then the substitution item* $[x := b']$ *in* $a'[x := b']$ *has ancestor* $[x := b]$ *in* $a[x := b]$.
2. *In the following reductions, the first underlined item (which may be an application written "$)$".) is ancestor of the second underlined (substitution) item:*

$$\begin{array}{ll}
(bc)\underline{[x := a]} & \to_{\underline{\beta\mu\sigma}} (b\underline{[x := a]})c[x := a] \\
(bc)\underline{[x := a]} & \to_{\underline{\beta\mu\sigma}} (b\underline{[x := a]})c\underline{[x := a]} \\
(\mathcal{O}y.b)\underline{[x := a]} & \to_{\underline{\beta\mu\sigma}} \mathcal{O}y.b\underline{[x := a]} \\
((\lambda x.b)\underline{a}) & \to_{\underline{\beta\mu\sigma}} b\underline{[x := a]} \\
((\Delta x.a)\underline{b}) & \to_{\underline{\beta\mu\sigma}} \Delta y.a\underline{[x := \lambda w.y(wb)]}
\end{array}$$

3. *The ancestor relation behaves as expected in the confrontation with* σ*-reductions; i.e., if* $\xi[x := a]$ *is a context in which* $[x := a]$ *appears, then:*

$$\begin{array}{ll}
(\lambda y.b)\xi\underline{[x := a]} & \to_{\underline{\beta\mu\sigma}} b[y := \xi\underline{[x := a]}] \\
(\Delta y.b)\xi\underline{[x := a]} & \to_{\underline{\beta\mu\sigma}} \Delta z.b[y := \lambda w.z(w\xi\underline{[x := a]})] \\
(\lambda y.\xi\underline{[x := a]})b & \to_{\underline{\beta\mu\sigma}} \xi\underline{[x := a]}[y := b] \\
(\Delta y.\xi\underline{[x := a]})b & \to_{\underline{\beta\mu\sigma}} \Delta z.\xi\underline{[x := a]}[y := \lambda w.z(wb)] \\
(\mathcal{O}y.\xi\underline{[x := a]})[z := b] & \to_{\underline{\beta\mu\sigma}} \mathcal{O}y.\xi\underline{[x := a]}[z := b] \\
(\mathcal{O}y.b)[z := \xi\underline{[x := a]}] & \to_{\underline{\beta\mu\sigma}} \mathcal{O}y.b[z := \xi\underline{[x := a]}] \\
(bc)[z := \xi\underline{[x := a]}] & \to_{\underline{\beta\mu\sigma}} b[z := \xi\underline{[x := a]}]c[z := \xi\underline{[x := a]}] \\
(b\xi\underline{[x := a]})[y := c] & \to_{\underline{\beta\mu\sigma}} b[y := c]\xi\underline{[x := a]}[y := c] \\
(\xi\underline{[x := a]}b)[y := c] & \to_{\underline{\beta\mu\sigma}} \xi\underline{[x := a]}[y := c]b[y := c]
\end{array}$$

4. *The ancestor relation is compatible; e.g.: if* $a \to_{\underline{\beta\mu\sigma}} a'$ *where* $[x := b']$ *in* a' *has ancestor* $[x := b]$ *resp.,* $)b$ *in* a*, and if* $c \to_{\underline{\beta\mu\sigma}} c'$ *then* $[x := b']$ *in* $a'c'$ *has ancestor* $[x := b]$ *resp.,* $)b$ *in* ac.

The following lemma is similar to bactracking in the minimal derivation method of [6, 18].

Lemma 14 *If* $a \to_{\underline{\beta\mu\sigma}} a'$ *and* $[x := b']$ *is in* a'*, then one of the following holds:*

1. *Exactly one* $[x := b]$ *in* a *is an ancestor of* $[x := b']$ *in* a' *and* $b \to\!\!\!\!\to_{\underline{\beta\mu\sigma}} b'$.
2. $[x := b']$ *has an application item* $)b$ *as ancestor with* $b = b'$ *or* $b' = \lambda w.y(wb)$ *for some* $y, w \notin \mathsf{FV}(b)$ *and* $y \neq w$.

Proof: By induction on the structure of a. □

The following technical lemma is informative about the subterms b of a term a that are not part of substitution items $[y := d]$ in a. It says that for any such b, performing some meta-substitutions on $\sigma(b)$ results in a subterm of $\sigma(a)$.

Lemma 15

1. *If* $b \lhd a$ *and* b *is not a part of* d *for some* $[y := d]$ *in* a*, then* $\exists m, x_1, \ldots x_m, c_1, \ldots c_m$ *such that* $\sigma(b)[c_1/x_1][c_2/x_2]\ldots[c_m/x_m]$ *is a subterm of* $\sigma(a)$.
2. *If* $(\mathcal{O}x.b)c \lhd a$ *which is not part of* d *for any* $[y := d]$ *in* a*, and if* $\sigma(a) \in \mathsf{SN}(\beta\mu)$ *then* $\beta\mu(\sigma(c)) < \beta\mu(\sigma(a))$.

Proof: 1: By induction on the structure of a. 2: By 1 and Lemma 5, there exists $c_i, x_i, 1 \leq i \leq m$ such that $(\mathcal{O}x.\sigma(b))\sigma(c)[c_1/x_1]\ldots[c_m/x_m] \lhd \sigma(a)$. Hence $\beta\mu(((\mathcal{O}x.\sigma(b))\sigma(c))) \leq \beta\mu(\sigma(a))$. It follows that $\beta\mu(\sigma(c)) < \beta\mu(\sigma(a))$. □

The following lemma is the key to proving GPSN. It says that any $\underline{\beta\mu\sigma}$-reduct a' of a decent term a whose σ-normal form has no infinite $\beta\mu$-derivations, is itself decent and its σ-normal form has no infinite $\beta\mu$-derivations.

Lemma 16 *If a is a decent term s.t. $\sigma(a) \in \mathsf{SN}(\beta\mu)$ and $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} a'$, then a' is decent of order $\beta\mu(\sigma(a))$.*

Proof: By induction on the number of reduction steps in $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} a'$.

- For the base case, as a is decent, a is decent of order $\beta\mu(\sigma(a))$.
- For the induction step, assume $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} a'' \rightarrow_{\underline{\beta\mu\sigma}} a'$. By IH, a'' is decent of order $\beta\mu(\sigma(a))$. Let $[x := b]$ in a'. We must show that $b \in \mathsf{SN}(\underline{\beta\mu\sigma})$ or $\beta\mu(\sigma(b)) < \beta\mu(\sigma(a))$.
  The ancestor of $[x := b]$ in a'' is either:
  1. $[x := b']$ in a'' where $b' \twoheadrightarrow_{\underline{\beta\mu\sigma}} b$
  2. $)b$ in a'' and $(\lambda x.c)b \rightarrow_{\underline{\beta\mu\sigma}} c[x := b]$ is the contracted redex in $a'' \rightarrow_{\underline{\beta\mu\sigma}} a'$.
  3. $)b'$ in a'' where $(\Delta x.c)\overline{b'} \rightarrow_{\underline{\beta\mu\sigma}} c[x := \lambda w.y(wb')]$ is the contracted redex in $a'' \rightarrow_{\underline{\beta\mu\sigma}} a'$ and $b = \lambda w.y(wb')$.

Case 1 As a'' is decent of order $\beta\mu(\sigma(a))$, then either $b' \in \mathsf{SN}(\underline{\beta\mu\sigma})$ or $\beta\mu(\sigma(b')) < \beta\mu(\sigma(a))$. Hence, $b \in \mathsf{SN}(\underline{\beta\mu\sigma})$ or $\beta\mu(\sigma(b)) \leq \beta\mu(\sigma(b')) < \beta\mu(\sigma(a))$ using Lemma 5.

Case 2 If $)b$ is not part of d for some $[y := d]$ in a'', then by Lemma 15, as $\beta\mu(\sigma(a'')) < \infty$, $\beta\mu(\sigma(b)) < \beta\mu(\sigma(a'')) \leq \beta\mu(\sigma(a))$ by Lemma 5. If $)b$ is part of d for some $[y := d]$ in a'', then we may assume that there is no $[z := e]$ such that $)b$ is part of e and $[z := e]$ is part of d. Then as a'' is decent, either $d \in \mathsf{SN}(\underline{\beta\mu\sigma})$ or $\beta\mu(\sigma(d)) < \beta\mu(\sigma(a''))$. If $d \in \mathsf{SN}(\underline{\beta\mu\sigma})$ then $b \in \mathsf{SN}(\underline{\beta\mu\sigma})$. If $\beta\mu(\sigma(d)) < \beta\mu(\sigma(a'')) \leq \beta\mu(\sigma(a))$ then as $(\lambda x.c)b$ is not part of some $[z := e]$ in d, we get by Lemma 15 that $\beta\mu(\mu\sigma(b)) < \beta\mu(\mu\sigma(d))$. Hence, $\beta\mu(\mu\sigma(b)) < \beta\mu(\mu\sigma(a))$.

Case 3 Similar to the second but note that $\beta\mu(\sigma(\lambda w.y(wb'))) = \beta\mu(\mu\sigma(b'))$ by Lemma 5, and $b' \in \mathsf{SN}(\underline{\beta\mu\sigma})$ iff $\lambda w.y(wb') \in \mathsf{SN}(\underline{\beta\mu\sigma})$. □

Finally, we show that every decent term whose σ-normal form is $\beta\mu$-strongly normalising is itself $\underline{\beta\mu\sigma}$-strongly normalising:

Theorem 17 *If a is a decent term and $\sigma(a) \in \mathsf{SN}(\beta\mu)$, then $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$.*

Proof: By strong induction on $\beta\mu(\sigma(a)) < \infty$ (note that $\sigma(a) \in \mathsf{SN}(\beta\mu)$). By Lemma 16, $\forall a'$, if $a \twoheadrightarrow_{\underline{\beta\mu\sigma}} a'$, then a' is decent of order $\beta\mu(\sigma(a))$.

Assume a has an infinite derivation. We shall derive a contradiction. As σ is SN (Lemma 5), this derivation can be written as

$$a \twoheadrightarrow_{\sigma} b_1 \rightarrow_{\underline{\beta\mu}} c_1 \twoheadrightarrow_{\sigma} b_2 \rightarrow_{\underline{\beta\mu}} c_2 \ldots$$

Again by Lemma 5, $\sigma(a) = \sigma(b_1) \twoheadrightarrow_{\beta\mu} \sigma(c_1) \twoheadrightarrow_{\beta\mu} \sigma(c_2) \twoheadrightarrow_{\beta\mu} \ldots$.
By Lemma 11 and the fact that $\beta\mu(\sigma(a)) < \infty$, only finitely many of the reductions $b_m \to_{\underline{\beta\mu}} c_m$ are not superfluous –otherwise, we will have an infinite $\beta\mu$-derivation starting at $\sigma(a)$ which is impossible since $\beta\mu(\sigma(a)) < \infty$. So let $b_M \to_{\underline{\beta\mu}} c_M$ be the last non-superfluous $\to_{\underline{\beta\mu}}$-reduction and define h_2 as follows:

$$h_2(x) = 1 \qquad h_2(ab) = h_2(a) + h_2(b) + 1$$

$$h_2(\mathcal{O}x.b) = h_2(b) + 1 \qquad h_2(a[x := b]) = \begin{cases} h_2(a).(h_2(b) + 2) & \text{if } x \in \sigma\mathsf{FV}(b) \\ 2h_2(a) & \text{otherwise} \end{cases}$$

It is easy to prove by induction on the structure of terms that:

- If $a \to_{\underline{\beta\mu\sigma}} b$ is superfluous then $h_2(a) = h_2(b)$;
- If $a \to_{\underline{\mu_2\mu_3}} b$ is not superfluous then $h_2(a) > h_2(b)$;
- If $a \to_{\sigma} b$ is not superfluous then $h_2(a) > h_2(b)$.

Now, $\exists N > M$ such that $\forall n \geq N$, $h_2(c_n) = h_2(c_N)$, as $\forall n > M$, $b_n \to_{\underline{\beta\mu_1}} c_n$ is superfluous. Hence, $h_2(b_n) = h_2(c_n)$. Moreover, $h_2(d) < \infty$ for any term d.

Next, look at the part of the derivation: $c_N \twoheadrightarrow_{\sigma} b_{N+1} \to_{\underline{\beta\mu}} c_{N+1} \twoheadrightarrow_{\sigma} \ldots$.
We know that in this derivation, all $\underline{\beta\mu}$-reduction steps are superfluous. As $\forall n \geq N$, $h_2(c_n) = h_2(c_N) = h_2(b_n) = h_2(b_{n+1})$, it must be also the case that $c_n \twoheadrightarrow_{\sigma} b_{n+1}$ is superfluous for all $n \geq N$, otherwise, $h_2(c_n) > h_2(b_{n+1})$, contradiction.

Hence, one $[x := d]$ in c_N has an infinite $\underline{\beta\mu\sigma}$-derivation. Otherwise, there wouldn't be an infinite $\underline{\beta\mu\sigma}$-derivation starting at c_N, contradicting infinity of $c_N \twoheadrightarrow_{\sigma} b_{N+1} \to_{\underline{\beta\mu}} c_{N+1} \ldots$.
Now, take one innermost $[x := d]$ in c_N which has an infinite $\underline{\beta\mu\sigma}$-derivation. Then d is decent. As c_N is a $\underline{\beta\mu\sigma}$-reduct of a, then c_N is decent of order $\beta\mu(\sigma(a))$ by Lemma 16. Moreover, $\beta\mu(\sigma(d)) < \beta\mu(\sigma(a))$.

Hence, by IH, we get that $d \in \mathsf{SN}(\underline{\beta\mu\sigma})$. Absurd. □

Now, the proof of GPSN is immediate:

Theorem 18 (Generalised Preservation of Strong Normalisation)
Let $a \in \mathcal{T}^e$, if every subterm b of a satisfies $\sigma(b) \in \mathsf{SN}(\beta\mu)$, then $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$.

Proof: By induction on the structure of a. As a is a subterm of a, then $\sigma(a) \in \mathsf{SN}(\beta\mu)$. If $[x := b]$ is a substitution item in a, then the IH holds for b and $b \in \mathsf{SN}(\underline{\beta\mu\sigma})$ and hence a is decent. So by Theorem 17, $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$. □

5 A type-assignment for $\lambda\Delta$exp

In [28], a classical type-assignment system for $\lambda\Delta$ is presented. The type-assignment system is simply typed, with a specific type $\perp$ standing for absurdity. Δ is typed with double negation.

Definition 19

1. *The set of types is given by the abstract syntax: $\mathcal{T} = \perp \mid \mathcal{T} \to \mathcal{T}$*

2. *A variable declaration is a pair $x : A$ where $x \in V$ and $A \in \mathcal{T}$.*
3. *A context is a finite list of declarations $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ such that $i \neq j \Rightarrow x_i \neq x_j$. If $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ is a context, $B \in \mathcal{T}$ and x does not occur in Γ, then $\Gamma, x : B$ is used to denote the context $x_1 : A_1, \ldots, x_n : A_n, x : B$.*
4. *The set of contexts is denoted by $\mathcal{C}$.*
5. *The derivability relation $\vdash_{\beta\mu} \subseteq \mathcal{C} \times \mathcal{T} \times \mathcal{T}$ is defined as follows (using the standard notation):*

$$(var)\quad \frac{}{\Gamma \vdash_{\beta\mu} x : A} \text{ if } (x : A) \in \Gamma \qquad (\lambda)\quad \frac{\Gamma, x : A \vdash_{\beta\mu} a : B}{\Gamma \vdash_{\beta\mu} \lambda x.a : A \to B}$$

$$(ap)\quad \frac{\Gamma \vdash_{\beta\mu} a : A \to B \quad \Gamma \vdash_{\beta\mu} b : A}{\Gamma \vdash_{\beta\mu} a\ b : B} \qquad (\Delta)\quad \frac{\Gamma, x : A \to \bot \vdash_{\beta\mu} a : \bot}{\Gamma \vdash_{\beta\mu} \Delta x.a : A}$$

6. *The derivability relation $\vdash_{\underline{\beta\mu\sigma}} \subseteq \mathcal{C} \times \mathcal{T}^e \times \mathcal{T}$ is defined by the above rules and the new rule:*

$$(subst)\quad \frac{\Gamma, x : A \vdash_{\underline{\beta\mu\sigma}} a : B \quad \Gamma \vdash_{\underline{\beta\mu\sigma}} b : A}{\Gamma \vdash_{\underline{\beta\mu\sigma}} a[x := b] : B}$$

The following lemma establishes three basic properties of the type system:

Lemma 20

1. *Subject Reduction: if $\Gamma \vdash_{\underline{\beta\mu\sigma}} a : A$ and $a \to_{\underline{\beta\mu\sigma}} b$, then $\Gamma \vdash_{\underline{\beta\mu\sigma}} b : A$.*
2. *Conservativity: if $\Gamma \vdash_{\underline{\beta\mu\sigma}} a : A$ then $\Gamma \vdash_{\beta\mu} \sigma(a) : A$.*
3. *Closure under subterms: every subterm of a well-typed term is well-typed.*

Proof: By an easy induction on the derivation of $\Gamma \vdash_{\underline{\beta\mu\sigma}} a : A$. □

The following proposition establishes that the simply typed version of $\lambda\Delta$exp is SN. Its proof is simple thanks to the generalised PSN.

Proposition 21

1. *If $\Gamma \vdash_{\beta\mu} a : A$, then $a \in \mathsf{SN}(\beta\mu)$.*
2. *If $\Gamma \vdash_{\underline{\beta\mu\sigma}} a : A$, then $a \in \mathsf{SN}(\underline{\beta\mu\sigma})$.*

Proof: 1: proved in [28]. 2: assume a is a term of minimal length such that $\Gamma \vdash_{\underline{\beta\mu\sigma}} a : A$ and $a \notin \mathsf{SN}(\underline{\beta\mu\sigma})$. By Lemma 20.2 and 1 above, $\sigma(a) \in \mathsf{SN}(\beta\mu)$. By GPSN (Theorem 18), a must therefore contain a strict subterm b such that $\sigma(b) \notin \mathsf{SN}(\beta\mu)$. By Lemma 4, $\to_{\beta\mu} \subseteq \twoheadrightarrow_{\underline{\beta\mu\sigma}}$, hence it follows that $\sigma(b) \notin \mathsf{SN}(\underline{\beta\mu\sigma})$ and so $b \notin \mathsf{SN}(\underline{\beta\mu\sigma})$. By Lemma 20.3, b is a well-typed term. This contradicts the minimality of a. □

6 CPS translation

Continuation-passing style (CPS) translation is a standard compilation technique. Its properties have been thoroughly studied in the context of pure and typed λ-calculus, see for example [26, 22]. In this section, we extend these results to the $\lambda\Delta$exp-calculus. To our knowledge, it is the first study of CPS translations for calculi of explicit substitutions.

Definition 22 *The CPS translation* $\underline{\ }$ *takes as input a* $\lambda\Delta$exp*-term and returns as output a* λexp*-term. It is defined as follows:*

1. *CPS translation on terms:*

$$\begin{aligned}
\underline{x} &= \lambda k.\, x\ k \\
\underline{\lambda x.\, M} &= \lambda k.\, k\ (\lambda x.\, \underline{M}) \\
\underline{M_1\ M_2} &= \lambda k.\, \underline{M_1}\ (\lambda y.\, y\ \underline{M_2}\ k) \\
\underline{\Delta x.\, M} &= \lambda k.\, \underline{M}[x := \lambda h.\, h\ \lambda j.\, \lambda i.\, i\ (j\ k)]\lambda z.\, z \\
\underline{M[x := N]} &= \underline{M}[x := \underline{N}]
\end{aligned}$$

2. *CPS translations on types:*

$$\begin{aligned}
(\!|\alpha|\!) &= \neg\neg\alpha \\
(\!|A \to B|\!) &= \neg\neg((\!|A|\!) \to (\!|B|\!))
\end{aligned}$$

where $\neg A \equiv A \to \bot$ *for some fixed type* $\bot$.

The translation is an extension of Plotkin's call-by-name translation for the untyped λ-calculus. When considered as a translation on typed terms, the translation corresponds to Kolmogorov's double-negation translation. Also note that the explicit CPS translation yields a CPS translation $\underline{\underline{\ }}$ from pure $\lambda\Delta$-terms to pure λ-terms in the obvious way; this translation is proved correct in [5].

Theorem 23 (Correctness of CPS translation)

1. *For every two terms* M, N,

$$M =_{\beta\mu\sigma} N \quad \Rightarrow \quad \underline{M} =_{\underline{\beta\sigma}} \underline{N}$$

2. *For every judgement* (Γ, M, A),

$$\Gamma \vdash_{\lambda\mu\text{exp}} M : A \quad \Rightarrow \quad (\!|\Gamma|\!) \vdash_{\lambda\text{exp}} \underline{M} : (\!|A|\!)$$

Proof. The first item is proved in three steps:

1. prove by induction on the structure of the terms that for every term a,

$$\underline{\sigma(b)}[\underline{\sigma(c)}/x] \quad \twoheadrightarrow_{\underline{\beta\sigma}} \quad \underline{\sigma(b)[\sigma(c)/x]}$$

2. prove that for every term a, we have $\underline{a} \twoheadrightarrow_{\underline{\beta\sigma}} \underline{\sigma(a)}$. We treat the case where $a \equiv b[x := c]$. We have

$$\begin{aligned}
\underline{a} &\twoheadrightarrow_{\underline{\beta\sigma}} \underline{\sigma(b)}[x := \underline{\sigma(c)}] && \text{by I.H.} \\
&\twoheadrightarrow_{\underline{\beta\sigma}} \underline{\sigma(b)}[\underline{\sigma(c)}/x] \\
&\twoheadrightarrow_{\underline{\beta\sigma}} \underline{\sigma(b)[\sigma(c)/x]} \\
&\equiv \underline{\sigma(b[x := c])}
\end{aligned}$$

3. use the interpretation method, the correctness of $\underline{\ }$ and the fact that $\underline{a} \twoheadrightarrow_{\underline{\beta\sigma}} \underline{a}$ to conclude.

$$\begin{aligned} M =_{\underline{\beta\mu\sigma}} N &\Rightarrow \sigma(M) =_{\beta\mu} \sigma(N) \\ &\Rightarrow \underline{\underline{\sigma(M)}} =_{\beta} \underline{\underline{\sigma(N)}} \\ &\Rightarrow \underline{\sigma(M)} =_{\underline{\beta\sigma}} \underline{\sigma(N)} \\ &\Rightarrow \underline{M} =_{\underline{\beta\sigma}} \underline{N} \end{aligned}$$

For the second item, proceed by induction on the structure of derivations.

The above theorem proves that the CPS translation preserves equalities. One may consider whether the CPS translation preserves reductions. Unfortunately, $\underline{\ }$ does not.

Lemma 24 *Let a and b be $\lambda\Delta$exp-terms.*

1. $a \rightarrow_{\beta\sigma} b \quad \Rightarrow \quad \underline{a} \twoheadrightarrow^{+}_{\underline{\beta\sigma}} \underline{b}$
2. $a \rightarrow_{\underline{\mu}} b \quad \Rightarrow \quad \underline{a} =_{\underline{\beta\sigma}} \underline{b}$

Proof. Show that for every term a, we have $\lambda k.\underline{a}\ k \twoheadrightarrow_{\lambda\mathrm{exp}} \underline{a}$. Then proceed by induction on the structure of the terms.

In the $\lambda\Delta$-calculus, it is possible to obtain a reduction-preserving translation by defining an optimized CPS-translation which performs some so-called administrative reductions. This reduction correspondence may be used for example to deduce strong normalisation of the $\lambda\Delta$-calculus from strong normalisation of the simply typed λ-calculus [5].

The question arises whether such an optimized CPS translation may be used to prove PSN for $\lambda\Delta$exp. In calculi on explicit substitutions, it is however not possible to obtain such a reduction-preserving translation unless some form of composition of substitutions is assumed:

$$a[x := b][y := c] \quad \rightarrow \quad a[x := b[y := c]] \quad \text{if } y \notin FV(a) \qquad (*)$$

The above rule is needed in order to obtain an optimized CPS translation which is not too optimizing. Indeed, assume that we want to find optimizations c_1 and c_2 s.t.

$$\begin{aligned} \underline{(\Delta x.\, a)\ b} &\twoheadrightarrow_{\lambda\mathrm{exp}} c_1 \\ \underline{\Delta y.\, a[x := \lambda w.y\ (w\ b)]} &\twoheadrightarrow_{\lambda\mathrm{exp}} c_2 \\ c_1 &\twoheadrightarrow_{\lambda\mathrm{exp}} c_2 \end{aligned}$$

In the current calculus, we have to perform too many steps to find such a c_1. We have:

$$\begin{aligned} \underline{(\Delta x.\, a)\ b} &\equiv \lambda k.(\lambda k'.\underline{a}[x := \lambda h.h\ \lambda j.\lambda i.i\ (j\ k')]\ \lambda z.z)\ \lambda j.j\ \underline{b}\ k \\ &\twoheadrightarrow \lambda k.\underline{a}[x := \lambda h.h\ \lambda j.\lambda i.i\ (j\ k')][k' := \lambda j.j\ \underline{b}\ k]\ \lambda z.z \end{aligned}$$

If we want to proceed further without reducing the substitution items, then some form of composition of substitutions, as indicated above, is necessary. Unfortunately, the rule $(*)$ breaks PSN, as shown in [8]. It remains open whether one can find a

restriction of (*) which does not break PSN and which allows to obtain a reduction correspondence for CPS.

Remark: it may be possible to obtain a reduction-preserving translation by using meta-substitution instead of explicit substitution in the definition of the CPS translation for Δ-abstractions. However, we consider that a CPS translation between calculi of explicit substitutions should use explicit substitution rather than meta-substitution.

7 Related work

7.1 On preservation of strong normalisation

In a recent paper [9], Bloo and Rose describe how to construct an explicit substitution CRS from an arbitrary CRS.[5] Moreover they show that PSN holds for a restricted class of CRSs, which they call structure-preserving. Unfortunately, PSN for the $\lambda\Delta$exp-calculus cannot be derived from [9]. Indeed, the first μ-rewrite rule is written in the CRS framework as $(\mu x.X(x))\ Y \rightarrow \mu y.X(\lambda w.y\ (w\ Y))$. The condition of structure-preserving requires the argument $\lambda w.y\ (w\ Y)$ of the meta-application in the right-hand side to be a subterm of the left-hand side. Obviously this is not the case.

Independently of [9], Bloo and Geuvers have developed a technique based on recursive path ordering (RPO) to prove PSN for various calculi of explicit substitutions. As was pointed to us by Roel Bloo, the RPO technique may be used to prove PSN for $\lambda\Delta$exp. Finally, the minimal derivation technique of [6, 18] may be used to prove PSN of $\lambda\Delta$exp.

7.2 On explicit substitutions for control-like operators

Audebaud and Pym, Ritter and Wallen have studied calculi of explicit substitutions for another classical λ-calculus, namely Parigot's $\lambda\mu$-calculus [25]. Audebaud's calculus [3] of explicit substitutions is an explicit substitution calculus with de Bruijn indices and composition of substitutions –in the spirit of $\lambda\sigma$– whereas Pym, Ritter and Wallen's $\lambda\mu\epsilon$ [29] calculus is a named explicit substitution calculus without composition of substitutions –in the spirit of λexp.

In [3], the system presented is shown to be confluent on open terms. Confluence on open terms is not however a question that is usually studied in calculi written with named variables (such as the $\lambda\Delta$exp).

In [29], it is shown by a computability predicate argument that simply typable $\lambda\mu\epsilon$-terms are strongly normalising. Their result and ours do not imply each other in neither way. Yet we are confident that the GPSN proof of this paper may be adapted to $\lambda\mu\epsilon$. The advantage of GPSN is that it implies strong normalisation of the simply-typed, polymorphic, higher-order $\lambda\mu\epsilon$-calculus.

[5] The theory of Combinatory Reduction Systems was developed by J.W. Klop [20].

8 Conclusion

We have introduced a calculus of explicit substitutions $\lambda\Delta$exp for the calculus $\lambda\Delta$ and proved various properties of the calculus.

On the one hand, we showed that $\lambda\Delta$exp has the GPSN property. To our knowledge, $\lambda\Delta$exp is the first calculus of explicit substitutions which has the PSN property and is not structure-preserving. Its study suggests that one may be able to prove PSN for a class of CRSs substantially bigger than the class of structure-preserving CRSs.

On the other hand, we showed that Plotkin's call-by-name CPS translation can be extended to the $\lambda\Delta$exp in such a way that typing is preserved. Studying CPS translations for calculi of explicit substitutions seems to be an interesting subject, which we plan to investigate in greater depth.

Our choice of the $\lambda\Delta$-calculus rather than other calculi of control-like operators is based on the fact that the $\lambda\Delta$-calculus is the simplest (and most restrictive) control calculus. It is an open question to study explicit substitutions for non-local control operators such as Felleisen's $\mathcal{C}$ [13]. Interestingly, expliciting such calculi will require an explicit handling of contexts. This subject is left for future work.

Finally, it remains to exploit the results of this paper in classical theorem-proving and proof theory and other applications mentioned in the introduction. An implementation of a proof/type checker based on $\lambda\Delta$exp is currently being developed at Glasgow.

Acknowledgements We are grateful for Roel Bloo for his observation that the RPO method of [8] does apply to $\lambda\Delta$exp. The first author would also like to thank John Hatcliff and Morten Heine Sørensen for discussions on classical λ-calculi.

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. A. W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.
3. P. Audebaud. Explicit substitutions for the $\lambda\mu$-calculus. Technical Report RR94-26, Ecole Normal Superieure de Lyon, 1994.
4. H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*. North Holland, 1984.
5. G. Barthe, J. Hatcliff, and M.H. Sørensen. A notion of classical pure type system. In *Proceedings of MFPS'97*, volume 6 of *Electronic Notes in Theoretical Computer Science*, 1997. To appear.
6. 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), 1996.
7. R. Bloo. Preservation of Strong Normalisation for Explicit Substitution. Technical Report CS-95-08, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1995.
8. R. Bloo and H. Geuvers. Explicit substitution: On the edge of strong normalisation. Technical Report CS-96-10, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1996.

9. R. Bloo and K. Rose. Combinatory reduction systems with explicit substitutions that preserve strong normalisation. In H. Ganzinger, editor, *RTA '96*, volume 1103 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
10. 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.
11. G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 366–374. IEEE Computer Society Press, 1995.
12. R.K. Dybvig. *The Scheme Programming Language*. Prentice-Hall, 1987.
13. M. Felleisen, D.P. Friedman, E. Kohlbecker, and B. F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.
14. T.G. Griffin. A formulae-as-types notion of control. In *Principles of Programming Languages*, pages 47–58. ACM Press, 1990.
15. T. Hardin. Confluence Results for the Pure Strong Categorical Logic CCL : λ-calculi as Subsystems of CCL. *Theoretical Computer Science*, 65(2):291–342, 1989.
16. H. Herbelin. *Elimination des coupures dans les sequents qu'on calcule*. PhD thesis, Université de Paris 7, 1994.
17. F. Kamareddine and R. P. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.
18. F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. *Lecture Notes in Computer Science*, 982:45–62, 1995.
19. J.-W. Klop. Term rewriting systems. *Handbook of Logic in Computer Science*, II, 1992.
20. J.-W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.
21. L. Magnusson. *The implementation of ALF: a proof editor based on Martin-Löf's monomorphic type theory with explicit substitution*. PhD thesis, Department of Computer Science, Chalmers University, 1994.
22. A.R. Meyer and M. Wand. Continuation semantics in typed lambda-calculi (summary). In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer-Verlag, 1985.
23. C. Muñoz. Proof representation in type theory: State of the art. XXII Latinamerican Conference of Informatics CLEI Panel 96, June 3–7, 1996, Santafé de Bogotá, Colombia, April 1996.
24. C. Murthy. *Extracting Constructive Contents from Classical Proofs*. PhD thesis, Cornell University, 1990.
25. M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.
26. G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, December 1975.
27. D. Prawitz. *Natural Deduction: A proof theoretical study*. Almquist & Wiksell, 1965.
28. N.J. Rehof and M.H. Sørensen. The λ_Δ calculus. In M. Hagiya and J. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 516–542. Springer-Verlag, 1994.
29. E. Ritter, D. Pym, and L. A. Wallen. On the intuitionistic force of classical search. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Procedings of TABLEAU'96*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 295–311. Springer Verlag, 1996.
30. G. L. Steele. *Common Lisp: The Language*. Digital Press, Bedford, MA, 1984.