Explicit substitutions and intersection types

PIERRE LESCANNE
LIP, Ecole Normale supérieure de Lyon
43 allée d’Italie 69364 Lyon France
e-mail : Pierre.Lescanne@ens-lyon.fr
12 mai 2003

1 Introduction

Calculi of explicit substitutions have been introduced to give an account to the substitution process in lambda calculus. The idea is to introduce a notation for substitutions explicitly in the calculus. In other words one makes substitutions first class citizens whereas the classical lambda calculus leaves them in the meta-theory.

Originally, the expression “explicit substitution” and the concept itself were introduced by Abadi, Cardelli, Curien and Levy [1]. The idea of the authors was to replace categorical combinators by a syntax that extend lambda calculus. Actually the idea of internalizing substitutions into the lambda calculus is older. A first credit should be given to Curien with his $\lambda \rho$ calculus [4] which is the origin of the calculus of [1]. But Curry himself expressed the same wish much earlier. For him combinatory logic was a mean to analyze substitutions. In the introduction of his famous book [5] he noted the advantage of including a calculus of substitutions into the lambda calculus. Perhaps the idea of explicit substitution is not “explicit” there, but at least the program of building a simple system, which analyzes substitutions and which departs less of our intuition than combinatory logic, appears clearly. For Curry this system should not be far from lambda calculus. Nowadays, we would call such a system a “calculus of explicit substitution”. Another ancestor of calculi of explicit substitutions is de Bruijn $\lambda C \xi \varphi$ [7]. As its complicated name might indicate, $\lambda C \xi \varphi$ is not an easy calculus as it mixes up the concrete and the abstract syntax in the same framework. In [2], this calculus has been revisited in modern notations. At the noticeable exception of $\lambda \chi$ [10] which uses de Bruijn levels, most of these calculi use de Bruijn indices [6] but more natural approaches have been proposed where names for variables are made explicit like in classical lambda-calculus. The most popular presentation is due to Roel Bloo and Kristoffer Rose [3], but Lins [11] proposed a calculus with the same features in 1985. The work presented here is the result of a cooperation with Daniel Dougherty and Stéphane Lengrand [9, 8].
2 The calculus $\lambda x$

The calculus of explicit substitution $\lambda x$ extends the syntax of the classical lambda calculus with terms of the form $M \langle x := N \rangle$ called closures. They are terms with a body $M$ and an explicit substitution part.

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

In $\lambda x, \_ \langle x := \_ \rangle$ is a binary operator which is a binder for $x$ in its left subterms. More specifically, in the term $M \langle x := N \rangle$, $x$ is bound in the subterm $M$. The concept of free variable needs also to be extended. To emphasize the difference with usual freeness we call this new concept availability\(^1\). Roughly speaking, a variable is not available in a term if it does not appear in the term or it appears in a substituted part of the term which is associated with a variable not itself available, in other word, a variable is not available if it appears in a term which is going to disappear later on by reduction.

\[
\begin{align*}
AV(x) &= \{x\} \\
AV(\lambda x. M) &= AV(M) \setminus \{x\} \\
AV(MN) &= AV(M) \cup AV(N) \\
AV(M \langle x := N \rangle) &= AV(M) \cup AV(N) \setminus \{x\} \quad \text{if } x \in AV(M) \\
AV(M \langle x := N \rangle) &= AV(M) \setminus \{x\} \quad \text{if } x \notin AV(M)
\end{align*}
\]

The definition of $\lambda x$ makes an extensive use of Barendregt convention on variables, which can be stated as follows in the same context there exists never a variable which is both bound and free. This convention plays a key role in the following rules especially in the rule (Abs).

\[
\begin{align*}
(\lambda x. M) P & \rightarrow M \langle x := P \rangle \quad \text{(B)} \\
(MN) \langle x := P \rangle & \rightarrow M \langle x := P \rangle N \langle x := P \rangle \quad \text{(App)} \\
(\lambda x. M) \langle x := P \rangle & \rightarrow \lambda x. (M \langle x := P \rangle) \quad \text{(Abs)} \\
x \langle x := P \rangle & \rightarrow P \quad \text{(VarI)} \\
y \langle x := P \rangle & \rightarrow y \quad \text{(VarK)}
\end{align*}
\]

3 Types

In order to catch strong normalization we introduce a type system with intersection called $\mathcal{E}$ (see Fig. 1). It is worth to notice that there are two rules for closures. The first called cut, says that if $M$ receives type $\sigma$ in a context $\Gamma$ extended with type $\tau$ for $x$ then $M \langle x := N \rangle$ receives the same type $\sigma$ provided $N$ gets type $\tau$ in the context $\Gamma$. This rule is straightforwardly connected through the Curry-Howard correspondence to the cut rule in natural deduction, hence its name. In addition to cut, we added a new rule, which says that in typing $M \langle x := N \rangle$ one can safely drop the type of $x$ provided that $N$ is typeable and $x$ does not occur (is not available) in $M$. Then $M \langle x := N \rangle$ gets the same type as $M$.

---

\(^1\)Actually there is another extension of freeness to explicit substitutions which is not what we need here.
\[\Gamma \vdash M : \tau_1 \quad \Gamma \vdash M : \tau_2 \quad \Gamma \vdash M : \tau_1 \& \tau_2 \quad i \in \{1, 2\}\]

\[
\text{drop} \quad \frac{\Gamma \vdash M : \sigma \quad \text{Ntypable}}{\Gamma \vdash (\lambda x = N) : \sigma} \quad x \notin AV(M)
\]

\[
\text{cut} \quad \frac{\Gamma, x : \tau \vdash M : \sigma \quad \Delta \vdash N : \tau}{\Gamma \vdash M \lambda x . N : \sigma \& \tau}
\]

\[
\text{start} \quad \frac{\Gamma \vdash x : \sigma}{\Gamma \vdash x : \sigma} \quad x \in \Gamma
\]

**FIG. 1 – The typing system \( \mathcal{E} \)**

4 The result

The main result which is the core of [9] says that a term \( M \) is strongly normalizing if and only if there exists in \( \mathcal{E} \) a context \( \Gamma \) and a type \( \sigma \) such that \( \Gamma \vdash M : \sigma \). Actually \( \mathcal{E} \) is not the only system which has this property. Indeed van Bakkel and Dezani [12] proposed an alternative system with a rule they call K-cut instead of drop

\[
K \land \text{cut} \quad \frac{\Gamma, x : \tau \vdash M : \sigma \quad \Delta \vdash N : \tau}{\Delta, \Gamma \vdash M \lambda x . N : \sigma \land \tau} \quad \text{if} \ x \notin \Gamma
\]

Their system characterizes also strong normalization in \( \lambda x \).

**Références**


