Solving equations in a language with control operators

Stephane Le Roux, Pierre Lescanne
LIP, École Normale Supérieure de Lyon, Lyon, France

1 Introduction

Thirty years ago, Huet designed an algorithm for higher-order unification in simply typed \(\lambda\)-calculus [3, 4]. His algorithm was revisited at several occasions for instance by Snyder [7] and Prehofer [6]. In the early 90’s, Parigot [5] designed the \(\lambda\mu\)-calculus, an extension of the \(\lambda\)-calculus with continuations which offers a Curry-Howard correspondence for the classical logic. Then another calculus with the same “logical” features was introduced by Curien and Herbelin and called \(\lambda\mu\). Roughly speaking \(\lambda\mu\) and \(\lambda\mu\) are \(\lambda\)-calculi with control operators and continuations. Here we address the question of solving equations in those languages since it is as important in program manipulation as is unification in \(\lambda\)-calculus. As a starting point we use the \(\lambda\mu\) with the terminology introduced by Ghilezan and Lescanne [2]. At first glance, the non-determinism of that calculus hinders us from defining a simple notion of equality, therefore we decided to restrict our study to a confluent subcalculus which is associated with call by name. Its grammar, reduction and typing rules are simpler and, above all, it is confluent. This leads to a natural notion of equality by normalization. In this calculus, there is a natural notion of extensional equivalence given by two reduction rules. The new calculus is terminating and confluent.

In the second section, we present the calculus enriched with unknowns and constants. We then define substitutions and equations. In a last section we propose a set of rules for unification via transformations. First we give a simple general equation transformation system GET which is easily proved correct, and then we give a bound equation transformation system BET which is obtained from GET by restriction (conditional use of the transformation rules). BET should terminate (the proof is under way) leading to completeness. This work is really preliminary and therefore incomplete. Moreover it requires knowledge about \(\lambda\mu\), \(\lambda\mu\) and others, therefore it can look a bit technical at some place for someone not used to these calculi. We apologize to the reader for that, but we hope to open an exciting field to unification.

2 Calculi

The \(\lambda\mu\)-calculus was designed by Curien and Herbelin in [1]. As well known there is no notion of unification without a good notion of equality. For this we focus on a restricted version of \(\lambda\mu\), namely the version without \(\mu\) which we call \(\lambda\). \(\lambda\) is still
large enough to simulate \( \lambda \)-calculus and to interpret à la Curry-Howard classical logic through sequent calculus. Moreover

**Lemma 1** \( \lambda \mu \) terminates and is confluent.

In the \( \lambda \)-calculus, beside the \( \beta \)-reduction rule, there is the \( \eta \)-contraction.

\[
\lambda x.Mx \rightarrow M \quad \text{if} \ x \notin \text{FV}(M)
\]

It captures the extensional equality. Indeed in the simply typed \( \lambda \)-calculus, \( M \) and \( \lambda x.Mx \) have the same type and in an appropriate context that provides all the “expected” arguments \( \lambda x_1x_2\ldots x_n.M(x_1x_2\ldots x_n) \) and \( M \) reduce to the same term:

\[
(\lambda x_1x_2\ldots x_n.M(x_1x_2\ldots x_n)) \left[ r_1\ldots r_n \right] \rightarrow_\beta M \left[ r_1\ldots r_n \right]
\]

In \( \lambda \mu \) we keep the extensional reduction \( \eta \mu \) introduced by Curien and Herbelin. Those who know \( \lambda \mu \) may notice that there is less contexts to distinguish in \( \lambda \mu \) and therefore the extensional equality is different. Two terms could now be considered as equivalent in \( \lambda \mu \) whereas they are not in \( \lambda \mu \). That leads to introduce new contraction rules. In order to define equational problems, we add constants and unknowns. The grammar for \( \lambda \mu \) is as follows:

**Definition 1 (Terms)**

\[
\begin{align*}
    r &::= x \mid r_c \mid R \mid \lambda x.r \mid \mu \alpha.c \\
    e &::= \alpha \mid e_c \mid E \mid r \cdot e \\
    c &::= (r|e)
\end{align*}
\]

\( \Lambda \mu \) is the language associated with the above grammar. \( R \) and \( E \) are unknowns, \( r_c \) and \( e_c \) are constants. Terms of form \( r \) are called \( \text{CalleR} \), terms of the form \( e \) are called \( \text{CalleE} \) and terms of the form \( (r|e) \) are called Capsules. \( \Lambda \mu \) contains two kinds of bound variables, \( x \)-variables and \( \alpha \)-variables. Notice the distinction we make between unknowns and variables. Unknowns are there to be replaced by terms in substitutions to provide a solution. Variables are usually bound, but they can be free in some contexts. \( \Lambda \mu_0 \subset \Lambda \mu \) is the set of closed terms i.e. variable free.

**Definition 2** \( \text{FV} \) is the set of free variables and \( \text{UK} \) is the set of unknowns. An index “\( a \)” on a meta-term means that the underlying term is atomic.

\[
\begin{align*}
    \lambda x_1. (\lambda x_2. \ldots (\lambda x_n.r)\ldots) &\text{ will be written either } \lambda x_1x_2\ldots x_n.r \text{ or } \lambda x_1x_2\ldots x_n.r \\
    r_1 \cdot (r_2 \cdot \ldots \cdot (r_n \cdot e)\ldots) &\text{ will be written either } r_1r_2\ldots r_n \cdot e \text{ or } r_1r_2\ldots r_n \cdot e
\end{align*}
\]

As usual, the notation \( f^{\text{ Boo}_n} \) will be extensively used to denote \( n \)-tuples The reduction rules for the systems we call \( \lambda \mu \) (for the two first rules \( \lambda \) and \( \mu \)) and \( \lambda \mu \eta \) (for the five rules) are as follow.
Definition 3 (Rules)

\[
\begin{align*}
(\lambda) &\quad (\lambda x.r’ \bullet e) \rightarrow (r[x \leftarrow r’] \bullet e) \\
(\mu) &\quad (\mu \alpha. c \bullet e) \rightarrow c[\alpha \leftarrow e] \\
(\eta_0) &\quad \mu \alpha. r(\alpha) \rightarrow r & \text{if } \alpha \notin FV(r) \\
(\eta_{\mu}) &\quad \lambda x. \mu \alpha. c \rightarrow \mu \alpha’ . c[\alpha \leftarrow \alpha’] & \text{if } \alpha’ \notin FV(c) \text{ and } x, \alpha \notin c[\alpha \leftarrow \alpha’] \\
(\gamma) &\quad (\lambda x. \mu \alpha. c \bullet e)[\alpha \leftarrow \lambda x. \mu \alpha. c \bullet e[\alpha \leftarrow \lambda x. \mu \alpha. c \bullet e]](\alpha) & \text{if } \alpha \in FV(c)
\end{align*}
\]

We do not display the typing rules (see [1]).

3 Equations

As in any theory (think about differential equations) before solving equations one put them in canonical forms. Like \(\lambda\)-calculus, this role in \(\lambda\mu\) is played by \(\eta\)-long normal forms \((\eta nf)\). The \(\eta\)-long normal form of a term \(t\) is \(\eta\) convertible to \(t\), but every atomic calle\(\mathbb{R}\) of \(t\) is saturated by arguments according to its types and every atomic calle\(\mathbb{E}\) of \(t\) is saturated by adding as many \(\lambda\)-abstractions (i.e. calle\(\mathbb{R}\)s) as the corresponding calle\(\mathbb{R}\) in the capsule expects.

Due to lack of space (and time!), we do not tell in this abstract how to compute \(\eta\)-long normal forms, but we give some of its properties:

**Lemma 2** For all terms \(s\) and \(t\) such that \(s \xrightarrow{\eta \mu} t\), we have \(\eta(s) = \eta(t)\).

For all term \(t\) we have \(\eta(t) \xrightarrow{\eta \mu} \eta(t)\).

\(\eta\) is idempotent, i.e. \(\eta(\eta(t)) = \eta(t)\).

**Lemma 3 (Surface structure of an \(\eta\)-long normal form)** \(\eta\)-long normal forms are

- **Calle\(\mathbb{R}\):** \(\lambda x. \mu \alpha. c\) where \(\tau(\alpha) = \tau\) is a basic type and \(c \in \eta nf\).
- **Calle\(\mathbb{E}\):** \(r_n \bullet e\) with \(r \eta nf\).
- **Capsules:** \(r_a \| r_n \bullet e\) with \(r \eta nf\) or \(r_n \bullet e\) with \(\lambda x. \mu \alpha. c \eta nf\).

**Definition 4 (Substitution)** A substitution is a mapping \(\theta : UK \rightarrow \bar{\lambda \mu}\). We call \(D(\theta)\) the support of \(\theta\) (where \(\theta\) is not the identity), and \(I(\theta)\) the set of the unknowns brought by the image of the support.

Substitution are extended as maps from \(\bar{\lambda \mu}\) to \(\bar{\lambda \mu}\) as usual.

**Definition 5** Let \(W \subseteq UK\).

We say that two substitutions \(\theta\) and \(\sigma\) are \(\bar{\lambda \mu}\)-equal over \(W\) iff \(\forall X \in W\) one has \(\theta(X) \xrightarrow{\lambda \mu} \sigma(X)\). We then write \(\theta \equiv \lambda \mu \sigma [W]\).

We say that \(\sigma\) is \(\bar{\lambda \mu}\)-more general than \(\theta\) over \(W\) iff there exists another substitution \(\rho\) such that \(\forall X \in W\) \(\theta(X) \xrightarrow{\lambda \mu} \rho \circ \sigma(X)\). We then write \(\sigma \leq \lambda \mu \theta [W]\).
Lemma 4 If \( \theta \) be a substitution and \( u, v \) two \( \lambda \mu \eta \)-convertible terms, then \( \theta(u) \equiv_{\lambda \mu \eta} \theta(v) \).

Definition 6 (Normalized substitutions) Let \( \theta \) be a substitution. The normalized substitution associated with \( \theta \) is \( \eta \) if \( \theta(X) = \lambda \mu \eta X \) and \( X \equiv_{\eta} \theta(X) \) otherwise.

Lemma 5 If \( t \) is an \( \eta \)lnf and \( \theta \) is a normalized substitution, then \( \theta(t) \equiv_{\lambda \mu} \) is an \( \eta \)lnf.

An equational system is a multi-set of pairs \( \{c \equiv c'\} \text{ or } \{r \equiv r'\} \) of capsules, calleR or calleE. Solving this system means finding a substitution that unifies all its equations. We write \( U(S) \) the set of solutions of \( S \).

Definition 7 (Solved form) An unknown \( X \) is in solved form in a system \( S \) if it appears once and only once in \( S \), in the form \( \{X \equiv t\} \).

Lemma 6 Let \( S \) and \( S' \) be two equational systems. If \( S =_{\lambda \mu \eta} S' \) then \( U(S) = U(S') \).

Thanks to the previous lemma, from now on we consider only systems whose equations have members that are atomic or \( \eta \)lnf.

4 Solving

For didactic purpose, we design first a general equation transformation system (GET) which is not meant to be implemented since it leads to obvious dead ends, cycles and divergences. There are three kinds of rules: decomposition, elimination and addition rule.

CD Capsule decomposition
ED CalleE decomposition
RD CalleR decomposition
UE Unknown elimination
EE Equation elimination
EA Equation addition

\[
\begin{align*}
CD & \quad \{ (r|e) \equiv \langle r'|e'\rangle \} \cup S \Rightarrow \{ r \equiv r', e \equiv e' \} \cup S \\
ED & \quad \{ r \cdot e \equiv r' \cdot e' \} \cup S \Rightarrow \{ r \equiv r', e \equiv e' \} \cup S \\
RD & \quad \{ \lambda \mu \alpha \cdot c \equiv \lambda \mu \alpha \cdot c' \} \cup S \Rightarrow \{ c \equiv c' \} \cup S \\
UE & \quad \{ X \equiv t \} \cup S \Rightarrow \{ X \equiv t \} \cup (S[X \leftarrow t] \mid_{\lambda \mu}) \\
EE & \quad \{ t \equiv t \} \cup S \Rightarrow S \\
EA & \quad S \Rightarrow \{ t \equiv t' \} \cup S
\end{align*}
\]

Lemma 7 The system GET is correct, i.e. if \( S \Rightarrow_{GET} S' \) and \( \theta \in U(S') \) then \( \theta \in U(S) \)

We first restrict the scope where we seek for solutions.
Lemma 8 Let $S$ be an equational system and $\theta \in U(S)$, then there exists a normalized substitution $\sigma$ such that:

1. $D(\sigma) \subseteq UK(S)$ and $I(\sigma) \cap (UK(S) \cup D(\sigma)) = \emptyset$.
2. $\sigma \in U(S)$.
3. $\sigma \leq_{\text{ren}} \theta[UK(S)]$ and $\theta \leq_{\text{ren}} \sigma[UK(S)]$.

We call such a substitution a standard solution. The lemma says that we only need to seek for standard solutions: we can get the others afterwards by a normalized renaming.

As we have already mentioned, the system GET is far too general. A system that prevents non-termination is obtained by restricting applications of the rules. Especially $(EA)$ is changed into seven so-called committing rules which are combination of $(EA)$ with another. As in the higher-order unification in the $\lambda$-calculus, equations added by $(EA)$ are of the form $f \, X \equiv t \, g$ where $t$ is a term with a rigid (i.e. not unknown) surface and only unknowns below that surface.

$t$ is called a partial commitment for the unknown $X$ because it commits only for the surface and postpones questions like “what’s below?”. Of course, commitment are chosen as $\eta/(\eta f)$ and the new unknowns introduced by the commitment are fresh hence outside $UK(S)$. There are two rules for calleE commitments (a calleE unknown can be instantiate either with an atom or with a compound calleE) and five rules for calleR commitments. We give one of each as examples:

**PEC** (Partial commitment for compound calleE)

\[
\{ (\lambda x.r||E) \equiv c \} \cup S \\
\downarrow \\
\{ E \equiv \eta((R')\bullet E') \cup \{ (\lambda x.r||E) \equiv c \} \cup S | E \gets \eta((R')\bullet E') \} \downarrow_{\lambda \mu}
\]

**PRC** (Partial commitment for imitation-projection calleR)

\[
\{ (R||\overline{e_a}) \equiv \langle r_a||\overline{x[p_m,\overline{e_a},R_m]} \bullet e_a \rangle \} \cup S \\
\downarrow \\
\{ (R \equiv t_{l,p}(n, r_a, \overline{R_m})) \cup \} \\
\{ (f(\overline{p_m}, \overline{x}, \overline{e_a}, R_m) \equiv c_m \} \cup S | R \gets t_{l,p}(n, r_a, \overline{R_m}) \} \downarrow_{\lambda \mu}
\]

where \( t_{l,p}(n, r_a, \overline{R_m}) = \lambda x[p_m, \overline{x}, \overline{e_a}, \overline{R_m}] \cdot \gamma \cdot g(p, \overline{x}, \overline{e_a}, R_m, \gamma) \).

\[ f(\overline{p_m}, \overline{x}, \overline{e_a}, R) = \langle R || \lambda x.m \cdot (x||\overline{R}) \bullet \overline{\lambda x.m} \rangle \cdot \langle \overline{\lambda x.m} \bullet E \rangle \cdot \gamma \]

5 Conclusion

The new rules are derived from a correct system, so correctness is preserved. The proof of termination is under way and from it we expect completeness. Currently we build its proof on a three-stage lexicographic order, slightly more complex than this of $\lambda$-calculus.
References