Solving equations in a language with control operators

Stephane Le Roux, Pierre Lescanne

LIP, École Normale Supérieure de Lyon, Lyon, France

{Stephane.Le.Roux,Pierre.Lescanne}@ens-lyon.fr

1 Introduction

Thirty years ago, Huet designed an algorithm for higher-order unification in simply typed λ-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 λ-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 $\overline{\lambda}\mu\widetilde{\mu}$. Roughly speaking $\lambda\mu$ and $\overline{\lambda}\mu\widetilde{\mu}$ are λ-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 λ-calculus. As a starting point we use the $\overline{\lambda}\mu\widetilde{\mu}$ 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 $\lambda\mu$, $\overline{\lambda}\mu\widetilde{\mu}$ 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 $\overline{\lambda}\mu\widetilde{\mu}$-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 $\overline{\lambda}\mu\widetilde{\mu}$, namely the version without $\widetilde{\mu}$ which we call $\overline{\lambda\mu}$. $\overline{\lambda\mu}$ is still large enough to simulate λ-calculus and to interpret à la Curry-Howard classical logic through sequent calculus. Moreover

Lemma 1 *$\overline{\lambda\mu}$ terminates and is confluent .*

In the λ-calculus, beside the β-reduction rule, there is the η-contraction.

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

It captures the extensional equality. Indeed in the simply typed λ-calculus, M and $\lambda x.Mx$ have the same type and in an appropriate context that provides all the "expected" arguments $\lambda\overline{x_n}.(M\ \overline{x_n})$ and M reduce to the same term:

$$(\lambda\overline{x_n}.(M\ \overline{x_n}))\ \overline{P_n} \quad \twoheadrightarrow_\beta \quad M\ \overline{P_n}$$

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

Definition 1 (Terms)

$$\begin{aligned} r &::= x \mid r_c \mid R \mid \lambda x.r \mid \mu\alpha.c \\ e &::= \alpha \mid e_c \mid E \mid r \bullet e \\ c &::= \langle r \| e \rangle \end{aligned}$$

$\overline{\Lambda\mu}$ 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 CalleR, terms of the form e are called CalleE and terms of the form $\langle r\|e\rangle$ are called Capsules. $\overline{\Lambda\mu}$ contains two kinds of bound variables, *x-variables* and *α-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. $\overline{\Lambda\mu}_0 \subset \overline{\Lambda\mu}$ is the set of closed terms i.e. variable free.

Definition 2 *FV is the set of free* variables *and UK is the set of* unknowns. *An index "a" on a meta-term means that the underlying term is atomic.*

$$\begin{array}{rl} \lambda x_1.(\lambda x_2 \ldots (\lambda x_n.r)\ldots) & \text{will be written either } \lambda x_1 x_2 \ldots x_n.r \text{ or } \lambda\overline{x_n}.r \\ r_1 \bullet (r_2 \bullet \ldots \bullet (r_n \bullet e)\ldots) & \text{will be written either } r_1 r_2 \ldots r_n \bullet e \text{ or } \overline{r_n} \bullet e \end{array}$$

As usual, the notation $\overline{foo_n}$ will be extensively used to denote n-tuples The reduction rules for the systems we call $\overline{\lambda\mu}$ (for the two first rules (λ) and (μ)) and $\overline{\lambda\mu\eta}$ (for the five rules) are as follow.

Definition 3 (Rules)

$$\begin{array}{llll}
(\lambda) & \langle \lambda x.r \| r' \bullet e\rangle & \longrightarrow \langle r[x \leftarrow r'] \| e\rangle & \\
(\mu) & \langle \mu\alpha.c \| e\rangle & \longrightarrow c[\alpha \leftarrow e] & \\
(\eta_\mu) & \mu\alpha\langle r \| \alpha\rangle & \longrightarrow r & \textit{if } \alpha \notin FV(r) \\
(\eta_{\lambda\mu}) & \lambda x.\mu\alpha.c & \longrightarrow \mu\alpha'.c[x \bullet \alpha \leftarrow \alpha'] & \textit{with } \alpha' \notin FV(c) \textit{ and } x, \alpha \notin c[x \bullet \alpha \leftarrow \alpha'] \\
(\gamma) & \langle \lambda\overline{x_n}\mu\beta.c \| \alpha\rangle & \longrightarrow \langle \lambda\overline{x_n}\mu\beta c[\alpha \leftarrow \overline{x_n} \bullet \beta] \| \alpha\rangle & \textit{if } \alpha \in FV(c)
\end{array}$$

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 λ-calculus, this role in $\overline{\lambda\mu}$ is plaid by η-*long normal forms* (ηlnf). The η-long normal form of a term t is η convertible to t, but every atomic calleR of t is saturated by arguments according to its types and every atomic calleE of t is saturated by adding as many λ-abstractions (i.e. calleRs) as the corresponding calleR in the capsule expects.

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

Lemma 2 *For all terms s and t such that* $s \underset{\overline{\lambda\mu\eta}}{\twoheadleftarrow\!\!\twoheadrightarrow} t$, *we have* $\eta(s) = \eta(t)$.

For all term t we have $\eta(t) \xrightarrow{*}_\eta t\downarrow_{\overline{\lambda\mu\eta}}$.
η *is idempotent, i.e.* $\eta(\eta(t)) = \eta(t)$.

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

- *CalleR:* $\lambda\overline{x_n}.\mu\alpha.c$ *where* $\tau(\alpha) = \tau$ *is a basic type and* $c \in \eta lnf$.
- *CalleE:* $\overline{r_n} \bullet e_a$ *with* $r_i \eta lnf$.
- *Capsules:* $\langle r_a \| \overline{r_n} \bullet e_a\rangle$ *with* $\overline{r_n} \bullet e_a \in \eta lnf$ *or* $\langle \lambda\overline{x_n}.\mu\alpha.c \| e_a\rangle$ *with* $\lambda\overline{x_n}.\mu\alpha.c \in \eta lnf$.

Definition 4 (Substitution) *A substitution is a mapping* $\theta : UK \to \overline{\Lambda\mu}_0$. *We call* $D(\theta)$ *the support of* θ *(where* θ *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 $\overline{\Lambda\mu}$ to $\overline{\Lambda\mu}$ as usual.

Definition 5 *Let* $W \subseteq UK$.
We say that two substitutions θ *and* σ *are* $\overline{\lambda\mu\eta}$- *equal over W iff* $\forall X \in W$ *one has* $\theta(X) \underset{\overline{\lambda\mu\eta}}{\twoheadleftarrow\!\!\twoheadrightarrow} \sigma(X)$. *We then write* $\theta =_{\lambda\mu\eta} \sigma\ [W]$.

We say that σ *is* $\overline{\lambda\mu\eta}$*-more general than* θ *over W iff there exists another substitution* ρ *such that* $(\forall X \in W) \quad \theta(X) \underset{\overline{\lambda\mu\eta}}{\twoheadleftarrow\!\!\twoheadrightarrow} \rho \circ \sigma(X)$. *We then write* $\sigma \leq_{\lambda\mu\eta} \theta\ [W]$.

Lemma 4 *If* θ *be a substitution and* u, v *two* $\overline{\lambda\mu\eta}$*-convertible terms, then* $\theta(u) \xleftrightarrow[\overline{\lambda\mu\eta}]{} \theta(v)$.

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

Lemma 5 *If* t *is an* ηlnf *and* θ *is a normalized substitution, then* $\theta(t) \downarrow_{\overline{\lambda\mu}}$ *is an* ηlnf.

An equational system is a multi-set of pairs $\{c \stackrel{?}{=} c'\}$ $\{r \stackrel{?}{=} r'\}$ or $\{e \stackrel{?}{=} e'\}$ 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* $\{X \stackrel{?}{=} t\}$.

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 ηlnf.

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

$$\begin{array}{lr}
CD & \{\langle r \| e\rangle \stackrel{?}{=} \langle r' \| e'\rangle\} \cup S \Rightarrow \{r \stackrel{?}{=} r', e \stackrel{?}{=} e'\} \cup S \\
ED & \{r \bullet e \stackrel{?}{=} r' \bullet e'\} \cup S \Rightarrow \{r \stackrel{?}{=} r', e \stackrel{?}{=} e'\} \cup S \\
RD & \{\lambda\overline{x_n}\mu\alpha.c \stackrel{?}{=} \lambda\overline{x_n}\mu\alpha.c'\} \cup S \Rightarrow \{c \stackrel{?}{=} c'\} \cup S \\
UE & \{X \stackrel{?}{=} t\} \cup S \Rightarrow \{X \stackrel{?}{=} t\} \cup (S[X \leftarrow t] \downarrow_{\lambda\mu}) \\
EE & \{t \stackrel{?}{=} t\} \cup S \Rightarrow S \\
EA & S \Rightarrow \{t \stackrel{?}{=} t'\} \cup S
\end{array}$$

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 σ such that:*

1. $D(\sigma) \subset UK(S)$ *and* $I(\sigma) \cap (UK(S) \cup D(\sigma)) = \emptyset$.
2. $\sigma \in U(S)$.
3. $\sigma \leq_{\lambda\mu\eta} \theta[UK(S)]$ *and* $\theta \leq_{\lambda\mu\eta} \sigma[UK(S)]$.

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 λ-calculus, equations added by *EA* are of the form $\{X \stackrel{?}{=} t\}$ 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 ηlnf 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_c(Partial commitment for compound calleE)

$$\{\langle \lambda x.r \| E\rangle \stackrel{?}{=} c\} \cup S$$
$$\Downarrow$$
$$\{E \stackrel{?}{=} \eta(R') \bullet E'\} \cup (\{\langle \lambda x.r \| E\rangle \stackrel{?}{=} c\} \cup S)[E \leftarrow \eta(R') \bullet E'] \downarrow_{\lambda\mu}$$

$PRC_{i,p}$(Partial commitment for imitation-projection calleR)

$$\{\langle R \| \overline{r_n} \bullet e_a\rangle \stackrel{?}{=} \langle r_a \| \overline{\lambda \overline{z_{p_m}} \mu\beta . c_m} \bullet e_a\rangle\} \cup S$$
$$\Downarrow$$
$$\{R \stackrel{?}{=} t_{i,p}(n, r_a, \overline{R_m})\} \cup$$
$$\{(f(\overline{z_{p_m}}, \beta, \overline{r_n}, e_a, R_m) \stackrel{?}{=} c_m\} \cup S)[R \leftarrow t_{i,p}(n, r_a, \overline{R_m})] \downarrow_{\lambda\mu}$$

where

$$t_{i,p}(n, r_a, \overline{R_m}) = \lambda \overline{x_n} . \mu\gamma . \langle r_a \| \overline{g(p_m, \overline{x_n}, \gamma, R_m) . \gamma}\rangle$$
$$g(p, \overline{x_n}, \gamma, R) = \lambda \overline{z_p} . \mu\beta . f(\overline{z_p}, \beta, \overline{x_n}, \gamma, R)$$
$$f(\overline{z_p}, \beta, \overline{x_n}, \gamma, R) = \langle R \| \lambda x . \mu\alpha . \langle x \| \overline{z_p} \bullet \beta\rangle \bullet \overline{\lambda \overline{y_{k_n}} \mu\varepsilon . \langle x_n \| \overline{y_{k_n}} \bullet \varepsilon\rangle} \bullet \gamma\rangle$$

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 λ-calculus.

References

1. Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the 5 th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 233–243. ACM, 2000.
2. S. Ghilezan and P. Lescanne. Classical proofs and typed processes: intersection types and strong normalization. In *TYPES 03 workshop*, volume 3085 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
3. G. Huet. A unifi cation algorithm for typed lambda calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.
4. G. Huet. *Résolution d'équations dans les langages d'ordre 1,2, ...,*ω. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.
5. M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proc. of Int. Conf. on Logic Programming and Automated Reasoning, LPAR'92*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.
6. C. Prehofer. *Solving Higher-Order Equations From Logic to Programming*. Progress in Theoretical Computer Science. Birkhäuser, 1997.
7. Wayne Snyder. *A proof theory for general unification*, volume 11 of *Progress in computer science and applied logic*. Birkhäuser, 1991.