An Operational Account of Call-By-Value Minimal and Classical λ-calculus in "Natural Deduction" Form

(revision fixing a few errors – January 2010)

Hugo Herbelin[1] and Stéphane Zimmermann[2]

[1] INRIA, France
[2] PPS, University Paris 7, France

Abstract. We give a decomposition of the equational theory of call-by-value λ-calculus into a confluent rewrite system made of three independent subsystems that refines Moggi's computational calculus:

- the *purely operational system* essentially contains Plotkin's β_v rule and is necessary and sufficient for the evaluation of closed terms;
- the *structural system* contains commutation rules that are necessary and sufficient for the reduction of all "computational" redexes of a term, in a sense that we define;
- the *observational system* contains rules that have no proper computational content but are necessary to characterize the valid observational equations on finite normal forms.

We extend this analysis to the case of λ-calculus with control and provide with the first presentation as a confluent rewrite system of Sabry-Felleisen and Hofmann's equational theory of λ-calculus with control. Incidentally, we give an alternative definition of standardization in call-by-value λ-calculus that, unlike Plotkin's original definition, prolongs weak head reduction in an unambiguous way.

Introduction

The study of call-by-value evaluation in the context of λ-calculus goes back to Plotkin [1] who introduced and studied rules β_v and η_v and a continuation-passing-style (cps) semantics of a call-by-value λ-calculus, named λ_v. Significant contributions were made first by Moggi [2] then by Sabry and Felleisen [3] who provided axiomatizations of call-by-value λ-calculus shown by the last authors to be complete with respect to Plotkin's cps semantics. In the same paper, Sabry and Felleisen also give a sound and complete axiomatization of call-by-value λ-calculus with control (hereafter referred as classical call-by-value λ-calculus). Independently, Hofmann [4] gave an alternative axiomatization of the classical call-by-value λ-calculus.

The theory of call-by-value λ-calculus turns out to be de facto more complex to describe than the one of call-by-name λ-calculus. While β-reduction is enough to evaluate terms in call-by-name λ-calculus and η is enough to characterize the observational equality on finite normal forms (what Böhm separability theorem expresses, see [5] for a generic account of Böhm theorem *vs* observational completeness), the situation is more intricate in the call-by-value λ-calculus, where, facing the two rules of the equational theory of call-by-name λ-calculus, the equational theories of call-by-value λ-calculus of Sabry and Felleisen and of Moggi rely on about a half dozen rules[3].

Though, when we observe call-by-name and call-by-value in the context of sequent calculus [6, 7], the complexity of the respective theories is about the same. Then, what is *so* specific to natural deduction (of which the standard λ-calculus is a notation for proofs along the Curry-Howard correspondence) that makes call-by-value more complicated there? Let us first give a more precise look at what happens in sequent calculus.

Call-by-name and call-by-value in sequent calculus

In a previous work of the first author with Curien [6], an interpretation of (a variant of) Gentzen's sequent calculus [8] as a variant of λ-calculus called $\overline{\lambda}\mu\tilde{\mu}$-calculus was given, where μ refers to Parigot's control operator [9]. In this interpretation, the left/right symmetry of sequent calculus pervades into a perfect syntactic duality first between terms and evaluation contexts, and secondly between call-by-name and call-by-value reduction. Especially, the operator $\tilde{\mu}$, dual to μ, builds an evaluation context that binds the term to which it is applied, as in **let** $x = [\,]$ **in** M, in the same way as Parigot's μ builds a term that binds the evaluation context to which it is applied.

The analysis of λ-calculus through sequent calculus given in [7], whether we consider call-by-name or call-by-value, shows a clear separation between operational rules and observation rules (the operational rules are cut-elimination rules and they divide into logical rules that contract the interaction of term constructors and evaluation context constructors into more elementary interactions, and two structural rules, one called (μ) for the contraction of μ and one called $(\tilde{\mu})$ for the contraction of $\tilde{\mu}$, that duplicate or erase information, moving terms around to possibly create new logical interactions).

A noticeable peculiarity of sequent calculus is that even in the intuitionistic case, the μ operator is required and when sequent calculus is compared to intuitionistic natural deduction, it turns out that (μ) acts as a *commutative cut.*

What is the problem with call-by-value λ-calculus in natural deduction syntax?

Call-by-value λ-calculus precisely needs commutative cuts so as to reveal redexes hidden by β redexes that are not β_v redexes. Take as an example the term $((\lambda x.\lambda y.y)(zt))u$ with z, t and u as free variables. With respect to λ_v, this

[3] The theory of Moggi has seven rules; the theory of Sabry and Felleisen has six axioms of which β_{lift} : $E[$**let** $x = N$ **in** $M] \rightarrow$ **let** $x = N$ **in** $E[M]$ is redundant; we will see later on that isolating β_{lift} out is however important from an operational point of view.

term is a normal form, however the reduction between λy and u is possible if one takes a parallel syntax, like proof-nets or parallel application to obtain the term $(\lambda x.u)(zt)$.

This is due to the presence of free variables, that can hide potential reductions. When using a parallel syntax, one does not rely anymore on syntactical shapes, but on calculus dependencies. In our example, the reduction between λx and (zt) is independent from the reduction between λy and u.

This is where the structural rules announced in the abstract come in: they will denote implicit use of the reduction rule for μ. For call-by-name, we have the well-known σ-equivalence [10] that disentangle redexes, but we have to be more careful for call-by-value. So here, the rule $((\lambda x.\lambda y.M)N)P \xrightarrow{\sigma} (\lambda y.(\lambda x.M)N)P$ is not valid, because one changes the evaluation order of N and P and will break confluence in presence of a μ operator. Our structural rules should then preserve head reduction order to avoid such problems.

1 Call-by-value $\boldsymbol{\lambda}$-calculus ($\boldsymbol{\lambda_{CBV}}$-calculus)

In this section, we show how the equational theory of call-by-value λ-calculus can be decomposed into three independent subsystems of rules. Note that we consider here only left-to-right call-by-value.

1.1 The splitting of the usual $\boldsymbol{\beta_v}$ rule

To be able to use what sequent calculus teaches us, we need a construction corresponding to the $\tilde{\mu}$ of $\overline{\lambda}\mu\tilde{\mu}$. We extend the λ-calculus grammar with a let construction, as follows:

$$\begin{array}{ll}
V & ::= x \mid \lambda x.M \\
T & ::= MN \mid \textbf{let } x = M \textbf{ in } N \\
M, N & ::= V \mid T
\end{array}$$

The direct counterpart of this splitting is syntactically expressed by the two following rules:

$$\begin{array}{lll}
(\Rightarrow) & (\lambda x.M)N & \rightarrow \textbf{let } x = N \textbf{ in } M \\
(\texttt{let}_v) & \textbf{let } x = V \textbf{ in } M & \rightarrow M[x \leftarrow V]
\end{array}$$

To understand the intuition behind this, it is necessary to look at how head reduction is performed on an application. We have to check first if M is a value or not, and then if N is a value or not to determine what to reduce. So control is not clear on an application, sometimes it is the left term that has it and sometimes it is the right one.

Using the **let** is an elegant way to get rid of this. You only need to look at the left term of an application, and when this term is eventually reduced to a λ, it naturally gives the control to the term of the right thanks to the $(\Rightarrow)$ rule.

Thanks to this decomposition, we can now have a clear notion of control, and be able to determine where to work on. An evaluation context F is now either

an application with a hole on the left, or a hole inside a **let**, say:
$F ::= [\ ]M \mid \textbf{let } x = [\ ] \textbf{ in } M$
and the mechanism of giving control to the "right" part of the application is now explicit. As we will see later, this will help to solve a standardization problem that arises from this ambiguity.

1.2 Dealing with the implicit (μ) rules

Whenever in the calculus there exists a conditional rule $(W)L \to R$, we will call a *p*seudo-redex a term matching L but not satisfying the side condition. For call-by-value λ-calculus, the term $(\lambda x.x)(yz)$ is a pseudo-redex, because yz is not a value.

In our calculus, the pseudo-redexes are not of the shape $(\lambda x.M)T$, but more like **let** $x = T$ **in** M. Still, they can hide reductions. Take for example the term (**let** $x = zt$ **in** $\lambda y.y)u$, the counterpart to the term $((\lambda x.\lambda y.y)(zt))u$. The reduction between λy and u is still hidden, and we need to get the u inside the **let**, in a sense. This can be done using the two rules (let_{let}) and (let_{app}).

$$\begin{array}{ll} (let_{let}) & \textbf{let } x = (\textbf{let } y = M \textbf{ in } N) \textbf{ in } P \\ & \to \textbf{let } y = M \textbf{ in let } x = N \textbf{ in } P \\ (let_{app}) & (\textbf{let } x = M \textbf{ in } N)P \\ & \to \textbf{let } x = M \textbf{ in } NP \end{array}$$

The (let_{app}) rule is here to deal with a pseudo-redex on the left side of an application. If you have a pseudo-redex inside on the left of an application, you can get it outside the application without disturbing the evaluation order, and recover the possible interactions that are independent of the substitution. Using this rule on the example above, we get (**let** $x = zt$ **in** $\lambda y.y)u \to$ **let** $x = zt$ **in** $(\lambda y.y)u$. The subterm zt still has control during the reduction, but we can do the other reductions as well.

The (let_{let}) rule has the same role, only for the right part of an application.

1.3 Basic properties of the calculus

Our presentation of call-by-value λ-calculus, called λ_{CBV}-calculus, is given in Figure 1. The auxiliary definition of evaluation context F (resp. E) is used to find the sub-term locally (resp. globally) in control in the term.

The $(\Rightarrow)$ and (let_v) rules correspond to the (β_v) rule, and the (let_{let}) and (let_{app}) rules are the two rules managing the implicit (μ) reductions.

Regarding notations, we will use as a convention that parentheses on the left of an application are implicit, meaning that the term MNP stands for $((MN)P)$. In the term $\lambda x.M$, we will say that the occurrence of x in M are bound and by free variables, we mean variables that are not bound. The set of the free variables of a term M will be written $FV(M)$.

To avoid problems of capturing variables, we will work modulo α-conversion, i.e. modulo the renaming of bound variables. Therefore, the names for free variables and bound variables in a given term can always be made distinct. By

Syntax

$$\begin{array}{lll} V & ::= x \mid \lambda x.M & F ::= [\ ]M \mid \mathbf{let}\ x = [\ ]\ \mathbf{in}\ M \\ M, N, P & ::= V \mid MN \mid \mathbf{let}\ x = M\ \mathbf{in}\ N & E ::= [\ ] \mid xE \mid F[E] \end{array}$$

Operational rules

$$\begin{array}{lll} (\Rightarrow) & (\lambda x.N)M & \xrightarrow{r} \mathbf{let}\ x = M\ \mathbf{in}\ N \\ (let_v) & \mathbf{let}\ x = V\ \mathbf{in}\ N & \xrightarrow{r} N[x \leftarrow V] \end{array}$$

Structural rules

$$\begin{array}{lll} (let_{let}) & \mathbf{let}\ z = (\mathbf{let}\ x = M\ \mathbf{in}\ N)\ \mathbf{in}\ P & \xrightarrow{r} \mathbf{let}\ x = M\ \mathbf{in}\ (\mathbf{let}\ z = N\ \mathbf{in}\ P) \\ (let_{app}) & (\mathbf{let}\ x = M\ \mathbf{in}\ N)P & \xrightarrow{r} \mathbf{let}\ x = M\ \mathbf{in}\ NP \end{array}$$

Observational rules

$$\begin{array}{lll} (\eta_{\Rightarrow}) & \lambda x.(yx) & \xrightarrow{r} y \\ (\eta^E_{let}) & \mathbf{let}\ x = M\ \mathbf{in}\ E[x] & \xrightarrow{r} E[M] \quad \text{if } x \notin FV(E) \\ (let_{var}) & z(\mathbf{let}\ x = M\ \mathbf{in}\ N) & \xrightarrow{r} \mathbf{let}\ x = M\ \mathbf{in}\ zN \end{array}$$

Fig. 1. The full λ_{CBV}-calculus

$M[x \leftarrow V]$, we mean the capture-free substitution of every occurrence of the free variable x of M by V.

We write $\rightarrow$ for the compatible closure of $\xrightarrow{r}$ and $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$. We write $=$ for the equational theory associated.

Proposition 1 (Confluence). *The operational subsystem, its extension with structural rules and the full system of reduction rules are all confluent in λ_{CBV}-calculus. Otherwise said, for each of these three systems, if $M \rightarrow^* N$ and $M \rightarrow^* N'$, then there exists P such that $N \rightarrow^* P$ and $N' \rightarrow^* P$.*

Proof (Indication). This statement is proved using Tait – Martin-Löf method, applied to the parallel reduction $\Rrightarrow$ defined by the union (of generalizations) of the above reduction rules and the following congruence:

- $t \Rrightarrow t$
- if $M \Rrightarrow M'$ then $\lambda x.M \Rrightarrow \lambda x.M'$
- if $M \Rrightarrow M'$ and $N \Rrightarrow N'$ then $MN \Rrightarrow M'N'$ and $\mathbf{let}\ x = M\ \mathbf{in}\ N \Rrightarrow \mathbf{let}\ x = M'\ \mathbf{in}\ N'$

Note that (let_{var}) is redundant in the equational theory but necessary in the reduction theory to get the confluence. ■

We say that a reduction relation $\rightarrow$ is *operationally complete* if whenever a closed M is not a value, it is reducible along $\rightarrow$. For instance, usual β-reduction is operationally complete for call-by-name λ-calculus.

Proposition 2. *λ_{CBV} equipped with the rules $(\Rightarrow)$ and (let_v) is operationally complete.*

PROOF. Any closed term which is not a value has either the form $E[\textbf{let } x = \lambda y.M \textbf{ in } N]$ in which case (let_v) is applicable or $E[(\lambda x.M)N]$ in which case $(\Rightarrow)$ is applicable. ■

Remark. For simplicity of the definition of λ_{CBV}, we did not consider a minimal set of operational rules. A minimal operationally complete set can be obtained by restricting both M in $(\Rightarrow)$ and V in (let_v) to be of the form $\lambda y.P$. However, this would force to explicitly add an observational rule $\textbf{let } x = y \textbf{ in } N \xrightarrow{r} N[x \leftarrow y]$ and a structural rule $(\lambda x.N)(E[yM]) \xrightarrow{r} \textbf{let } x = E[yM] \textbf{ in } N$ (where the notion of pseudo-redex and erasing rule, see below, is extended to the case of β-redexes).

Associated to reduction rules with constraint, we can define erasing rules used to erase the pseudo-redex part of the rule. Namely, we will use the system composed of the following rule for this:
$\textbf{let } x = N \textbf{ in } M \xrightarrow{er} N$

A term M is *normal* for a reduction $\rightarrow$ if it is not reducible for $\rightarrow$. It is *structurally normal* if, for all N such that $M \xrightarrow{er}{}^{*} N$, N is normal for the reduction generated by the operational rules. A reduction relation $\rightarrow$ is *structurally complete* if all terms normal for $\rightarrow$ are structurally normal.

Proposition 3. *The set of terms that are normal with respect to the operational and structural rules is described by the entry Q of the following grammar:*

$Q ::= \lambda x.Q \mid S \mid \textbf{let } x = S \textbf{ in } Q$

$S ::= x \mid SQ$

Moreover, the reduction generated by the operational and structural rules of λ_{CBV}-calculus is structurally complete.

PROOF (INDICATION). The two parts are proved by induction quite directly. ■

It is interesting to see that in a normal form like $\textbf{let } x = P \textbf{ in } Q$, the leftmost subterm of P is always a free variable and the evaluation of P is blocked by this variable. This suggests an alternative definition of structural completeness based on the notion of evaluation of open terms: a set of rules is structurally complete iff any term which is neither a value nor of the form $xM_1...M_n$ or $\textbf{let } y = xM_1...M_n \textbf{ in } N$ is reducible.

Let us now focus on the observational rules of Figure 1. Given a confluent reduction $\rightarrow$, we say that an equation $M = N$ between normal terms *belongs to the observational closure* of $\rightarrow$ if for every closed evaluation context E and every substitution ρ of the free variables of M and N by closed values, $E[\rho(M)]$ converges along $\rightarrow^*$ iff $E[\rho(N)]$ converges.

Proposition 4. *The observational rules from Figure 1 belong to the observational closure of the set of operational and structural rules of λ_{CBV}-calculus.*

PROOF. When instantiated by closed values, both sides of $(\eta_{\Rightarrow})$ and (let_{var}) converge by respectively using $(\Rightarrow)$ and (let_v), $(\Rightarrow)$ and (let_{let}). For (η^E_{let}), the result follows by standardization (see below) since at some point, for ρ a substitution of the free variables of $E'[yM]$, the reduction of $\rho(E'[yM])$ eventually yields a value and (let_v) is applicable. ■

1.4 The subformula property and structural completeness

The λ_{CBV}-calculus can be typed by natural deduction just like the conventional (call-by-name) λ-calculus. The only new construction is the **let**, which can be typed by the cut rule. The whole typing system is the following:

$$\frac{}{\Gamma,\, x : A \vdash x : A} \qquad \frac{\Gamma,\, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \Rightarrow B} \qquad \frac{\Gamma \vdash M : A \Rightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma,\, x : A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash \textbf{let } x = N \textbf{ in } M : B}$$

Contrary to the call-by-name λ-calculus, the original call-by value λ-calculus does not satisfy the subformula property for its normal forms. We can distinguish two kinds of "breaking" of this property.

The first one happens with the pseudo-redexes. The term $(\lambda x.x)(yz)$ is a normal form, which can be typed this way, with $\Gamma = y : B \Rightarrow A, z : B$:

$$\frac{\Gamma \vdash \lambda x.x : A \Rightarrow A \qquad \Gamma \vdash yz : A}{\Gamma \vdash (\lambda x.x)(yz) : A}$$

The problem here comes from the type $A \Rightarrow A$ that appears after the cut between $\lambda x.x$ and yz. If we remember that a cut, in natural deduction, is an introduction rule followed by a elimination rule, then neither of the two rules, taken alone, are problematic. Think of normal forms like yM, corresponding to the elimination of implication, and $\lambda x.M$, corresponding to the introduction of implication. It is really the cut that is causing problems.

In λ_{CBV}, because the pseudo-redexes correspond to sequent calculus cuts, we can get rid of this problem. The $A \Rightarrow A$ formula become $A \vdash A$, and our term become $(\lambda x.x)(yz) \to \textbf{let } x = yz \textbf{ in } x$, whose proof has the subformula property.

However, this is not enough to get rid of all our problems. With $\Gamma = u : A,\ z : C \Rightarrow D,\ t : C$ the term $(\textbf{let } x = zt \textbf{ in } \lambda y.y)u$ can be typed this way :

$$\frac{\Gamma \vdash \textbf{let } x = zt \textbf{ in } \lambda y.y : A \Rightarrow B \qquad \Gamma \vdash u : A}{\vdash (\textbf{let } x = zt \textbf{ in } \lambda y.y)u : B}$$

Again, there is a $A \Rightarrow B$ type appearing and breaking the subformula property. This time, it is not a problem of pseudo-redex, but rather one of hidden redex. To solve this, we need the structural (let_{app}) rule to reduce the interaction between y and u and, in a more general manner our terms have to be structurally complete otherwise hidden redexes will create unnecessary arrow types.

Pushing pseudo-redexes to more atomic **let** terms and having structural rules is sufficient to gain back the subformula property for λ_{CBV}, which is expressed by the following property.

Proposition 5. *Let M be a term of the λ_{CBV} calculus that is normal with respect to the operational and structural rules. Then if $\Gamma \vdash M : A$, its proof satisfies the subformula property.*

PROOF. We shall reason by induction on the proof of $\Gamma \vdash M : A$. The only technical point is to know that on a normal form, the leftmost term on an application is a variable which is in the context.

■

1.5 Standardization

Plotkin's definition of standard reduction sequences in λ_v does not characterize canonical standard reduction sequences: the standard reduction of a term is not always unique. We first recall the definition of head reduction $\xrightarrow{h}$ and standard reduction sequences (s.r.s.) in Plotkin [1]:

- $(\lambda x.M)V \xrightarrow{h} M[x \leftarrow V]$
- if $M \xrightarrow{h} M'$ then $MN \xrightarrow{h} M'N$
- if $M \xrightarrow{h} M'$ then $VM \xrightarrow{h} VM'$

- any variable is a s.r.s.
- if $M_1 \xrightarrow{h} M_2$ and $M_2, \ldots, M_n$ is a s.r.s., then $M_1, M_2, \ldots, M_n$ is a s.r.s.
- if $M_1, \ldots, M_n$ is a s.r.s. then $\lambda x.M_1, \ldots, \lambda x.M_n$ is a s.r.s.
- if $M_1, \ldots, M_n$ and $N_1, \ldots, N_p$ are s.r.s., then $M_1N_1, \ldots, M_nN_1, \ldots, M_nN_p$ is a s.r.s.

Now, if we take M and N such that $M \xrightarrow{h} M'$ and $N \xrightarrow{h} N'$, then the two following reductions are standard:

$$(\lambda x.M)N \xrightarrow{s} (\lambda x.M)N' \xrightarrow{s} (\lambda x.M')N'$$
$$(\lambda x.M)N \xrightarrow{s} (\lambda x.M')N \xrightarrow{s} (\lambda x.M')N'$$

The first one is built as an extension of head reduction, while the second one is built only using the application rule of standard reduction. The explanation is that in λ_v, there is no direct way to determine what is in control in an application, so there is no possibility to have a unique general rule for application.

In λ_{CBV}, since the (β_v) rule was split, the control is unambiguous, so we can get rid of this problem. Weak-head reduction for λ_{CBV} and an alternative, non ambiguous, definition of standard reduction sequences are given below.

- $M \xrightarrow{h} M'$ for any subset[4] of rules of λ_{CBV}

[4] If (let_v) is present in the subset, we assume that M in the rules (let_{app}), (let_{let}), (let_{var}) and (η^E_{let}) is restricted to the form $E[yM']$ so that head reduction favors let_v. If (η^E_{let}) is present, we assume that N in the rules (let_{app}), (let_{let}) and (let_{var}) is not of the form $E[x]$ so that head reduction favors (η^E_{let}).

- if $M \xrightarrow{h} M'$ then $F[M] \xrightarrow{h} F[M']$

- any variable is a s.r.s.
- if $M_1 \xrightarrow{h} M_2$ and $M_2, \ldots, M_n$ is a s.r.s., then $M_1, M_2, \ldots, M_n$ is a s.r.s.
- if $M, \ldots, E_1[V_1]$[5] is a head reduction sequence, and $V_1, \ldots V_n$ and $E_1, \ldots, E_p$ are s.r.s., then $M, \ldots, E_1[V_1], \ldots E_1[V_n], \ldots E_p[V_n]$ is a s.r.s.
- if $M_1, \ldots, M_n$ is a s.r.s., then $\lambda x.M_1, \ldots, \lambda x.M_n$ is a s.r.s.

- $[\,]$ is a s.r.s. on contexts
- if $M_1, \ldots, M_n$ and $E_1, \ldots, E_p$ are s.r.s. on terms and on contexts, then the sequences $E_1[\mathbf{let}\ x = [\,]\ \mathbf{in}\ M_1], \ldots, E_1[\mathbf{let}\ x = [\,]\ \mathbf{in}\ M_n], \ldots, E_p[\mathbf{let}\ x = [\,]\ \mathbf{in}\ M_n]$ and $E_1[[\,]M_1], \ldots, E_1[[\,]M_n], \ldots, E_p[[\,]M_n]$ are s.r.s. on contexts

Note that the definition applies to λ_v too by using only (β_v) in the definition of head reduction.

Theorem 1 (Standardization). *For all M and N, if $M \rightarrow^* N$ using any set of reduction rules there is a* unique *standard reduction sequence $M, \ldots, N$ extending head reduction. We shall note this reduction $M \xrightarrow{s} N$*

Proof (Indication). We shall proceed by induction, with permutation of non-standard reductions. Uniqueness is by canonicity of the definition of being a standard reduction sequence. ∎

But as we said before, we even got a stronger result. If M is a closed term, its head reduction (so its standard reduction) only uses the two rules $(\Rightarrow)$ and (let_v), and the other rules are not necessary. We shall prove this, but we need to notice that if M is a closed term, then M cannot be decomposed through $E[yP]$, and if $M = E[yP]$, then M never reduces to a value.

Proposition 6. *If M is a closed term such that $M \rightarrow^* V$, then there exists V' such that $M \xrightarrow{h}{}^* V'$ and $V = V'$ with $\xrightarrow{s}$ using only the rules $(\Rightarrow)$ and (let_v).*

Proof (Indication). As it is a consequence of the standard reduction, we just use an induction on the standard reduction of M. ∎

This result is not surprising, if you think that the $(\Rightarrow)$ and (let_v) rules correspond to the (β_v) rule, and that (β_v) rule is sufficient to deal with closed terms.

So we have a calculus where all rules have their own purpose, which is clearly defined.

1.6 Comparison with Moggi's λ_c-calculus

In [2], Moggi gives a call-by-value calculus, on the same syntax as λ_{CBV}. The reduction rules and the grammar characterizing normal forms with respect to the $(\beta_v), (let.1), (let.2)$ and $(let.let)$ rules are given in Figure 2 (T denotes terms that are not values). This calculus was made to allow more reductions than the original call-by-value calculus while remaining call-by-value.

[5] Remember that $E[]$ can be empty, so that M reduces to a value.

$$\begin{array}{lll}
(\beta_v) & (\lambda x.M)V & \to M[x \leftarrow V] \\
(let_v) & \mathbf{let}\ x = V\ \mathbf{in}\ M & \to M[x \leftarrow V] \\
(let.1) & TM & \to \mathbf{let}\ x = T\ \mathbf{in}\ xM \text{ with } x \text{ fresh} \\
(let.2) & VT & \to \mathbf{let}\ x = T\ \mathbf{in}\ Vx \text{ with } x \text{ fresh} \\
(let.let) & \mathbf{let}\ y = (\mathbf{let}\ x = M\ \mathbf{in}\ N)\ \mathbf{in}\ P & \to \mathbf{let}\ x = M\ \mathbf{in}\ \mathbf{let}\ y = N\ \mathbf{in}\ P \\
\\
(\eta_{\Rightarrow}) & \lambda x.Vx & \to V \text{ if } x \text{ is not free in } V \\
(let_{id}) & \mathbf{let}\ x = M\ \mathbf{in}\ x & \to M
\end{array}$$

$$\begin{array}{l}
W ::= x \mid \lambda x.Q \\
Q\ ::= \mathbf{let}\ x = yW\ \mathbf{in}\ Q \mid xW \mid W
\end{array}$$

Fig. 2. Moggi's λ_C-calculus and its normal forms

We say that two operational theories are in equational correspondence (see Sabry-Felleisen for the original notion [3]) if there is a bijection between the equivalence classes of the theories. Especially, we can show that our theory of λ_{CBV} is in equational correspondence with the one of Moggi.

Proposition 7. *The two calculi λ_c and λ_{CBV} are in equational correspondence.*

PROOF (INDICATION). We only show here the interesting cases of the simulation of the rules.

$let_{app} = let.1 + let.let + (let.1)^{-1}$
$let_{var} = let.2 + let.let + (let.2)^{-1}$
$let.1 = (\eta_{let}^{\mathbf{let}\ x=[]\ \mathbf{in}\ N})^{-1}$

For $let.2$ we proceed case by case. If V is y, then $let.2 = (\eta_{let}^{x[]})^{-1}$. If $V = \lambda y.M$ then $let.2 = \Rightarrow + (\Rightarrow)^{-1}$. ■

Our claim is now that λ_{CBV} has a finer structure than λ_c. First, observe that in Moggi's calculus, for the same reason as in λ_{CBV}, the rules (β_v) and (let_v) are sufficient for operational completeness.

Proposition 8. *The λ_c-calculus restricted to the rules (β_v) and (let_v) is operationally complete.*

The λ_c-calculus equipped with (β_v), (let_v), $(let.1)$, $(let.2)$ and $(let.let)$ is structurally complete.

A weird point is the presence of an observational rule, namely $(\eta_{\Rightarrow}^{[]})$ in the simulation of the $(let.1)$ and $(let.2)$ rules, providing an ambiguous status to these two rules, used for computation but containing a bit of observation too. This can be related to another not so likable behavior of λ_c. We remember that the calculus has the (β_v) rule which is sufficient to reduce closed terms to values, but if we reduce the term $(\lambda x.x)((\lambda x.x)(\lambda x.x))$ with head reduction, we obtain the following reduction sequence:
$(\lambda x.x)((\lambda x.x)(\lambda x.x)) \to \mathbf{let}\ y = (\lambda x.x)(\lambda x.x)\ \mathbf{in}\ (\lambda x.x)y \to^* \lambda x.x$.

The head reduction uses the rule $(let.2)$ as the first reduction rule, and for the term $((\lambda x.x)(\lambda xy.y))(\lambda x.x)(\lambda x.x)$ the head reduction uses the rules $(let.1)$ and (*let.let*). So, even if we do not need them, these rules appear and the λ_c-calculus tends to do useless expansions.

This was already noticed by Sabry and Wadler in [11] where they showed that λ_c is isomorphic to one of its sub-calculi where all the **let** rules have been reduced, and applications occur only between two variables. The **let** construction is here a way to make the head-reduction flow explicit and to reduce a term, λ_c firstly encodes its evaluation flow with some **let** manipulation, and only then reduces the term.

By doing this, we lose almost completely a "hidden agent" of the reduction : the structural congruence. In λ_v, the (β_v) rule was sufficient because it could go inside a term to find a redex, as here we almost never investigate in the term. The evaluation flow is encoded in a way that the topmost term always contains the redex, but is then superseding the structural congruence.

So, the status of the **let** rules is not clearly defined. They are used for reducing closed terms to values, but are also necessary for the structural completeness and contain a taste of observational rules.

Interestingly enough, we can switch between the normal forms or λ_C and λ_{CBV} by using the $\eta^{E}_{\Rrightarrow}$. Oriented from left-to-right we go from λ_C to λ_{CBV} and conversely with the right-to-left orientation.

2 Call-by-value $\boldsymbol{\lambda\mu}$tp-calculus ($\boldsymbol{\lambda\mu}$tp$_{\boldsymbol{CBV}}$-calculus)

2.1 Confluence and standardization

Our calculus is not hard to extend with continuation variables and control operators. We follow the approach of [12] and adopt Parigot's μ operator [9] together with a toplevel continuation constant $\mathtt{tp}$, what simplifies the reasoning on closed computations and the connection with the λ-calculi based on *callcc* and $\mathcal{A}$ or Felleisen's $\mathcal{C}$ operator. We write $C[\alpha \leftarrow [\beta]E]$ for the generalization of Parigot's structural substitution and we abbreviate $C[\alpha \leftarrow [\beta]([\,])]$ as $C[\alpha \leftarrow \beta]$.

Since the reductions associated to μ absorb their context, structural rules should not change which subterm is in control, but happily structural rules are only like $E[T] \rightarrow E'[T]$, so we cannot break confluence with them.

We must be careful with μ reductions however. If μ does not have any reduction restrictions, **let** still can hide μ redexes. Therefore, we need a new structural rule. The full calculus is given in Figure 3.

Proposition 9. *The operational subsystem, its extension with structural rules and the full system of reduction rules are all confluent in the $\lambda\mu\mathtt{tp}_{CBV}$-calculus.*

Proof. The proof is the same as before, using generalizations of the rules involving μ and **let** and with the parallel reduction extended with the two following rules:

- if $M \Rrightarrow M'$ then $[\alpha]M \Rrightarrow [\alpha]M'$
- if $C \Rrightarrow C'$ then $\mu\alpha.C \Rrightarrow \mu\alpha.C'$

Syntax

$$\begin{array}{lll} V & ::= x \mid \lambda x.M & F ::= [\,]M \mid \textbf{let } x = [\,] \textbf{ in } M \\ M,N,P & ::= V \mid MN \mid \textbf{let } x = M \textbf{ in } N \mid \mu\alpha.C & E ::= [\,] \mid xE \mid F[E] \\ C & ::= [q]M & \\ q & ::= \alpha \mid \mathsf{tp} & \end{array}$$

Operational rules

$$\begin{array}{lll} (\Rightarrow) & (\lambda x.N)M & \overset{r}{\to} \textbf{let } x = M \textbf{ in } N \\ (let_v) & \textbf{let } x = V \textbf{ in } N & \overset{r}{\to} N[x \leftarrow V] \\ (\mu_v) & F[\mu\alpha.C] & \overset{r}{\to} \mu\alpha.C[\alpha \leftarrow [\alpha]F] \\ (\mu_{base}) & [q]\mu\alpha.C & \overset{r}{\to} C[\alpha \leftarrow q] \end{array}$$

Structural rules

$$\begin{array}{lll} (let_{let}) & \textbf{let } z = (\textbf{let } x = M \textbf{ in } N) \textbf{ in } P & \overset{r}{\to} \textbf{let } x = M \textbf{ in } (\textbf{let } z = N \textbf{ in } P) \\ (let_{app}) & (\textbf{let } x = M \textbf{ in } N)P & \overset{r}{\to} \textbf{let } x = M \textbf{ in } NP \\ (let_{\mu}) & \textbf{let } x = M \textbf{ in } \mu\alpha.[\beta]N & \overset{r}{\to} \mu\alpha[\beta].\textbf{let } x = M \textbf{ in } N \end{array}$$

Observational rules

$$\begin{array}{lll} (\eta_{\Rightarrow}) & \lambda x.yx & \overset{r}{\to} y \\ (\eta^{E}_{let}) & \textbf{let } x = M \textbf{ in } E[x] & \overset{r}{\to} E[M] \quad \text{if } x \notin FV(E) \\ (\eta_{\mu}) & \mu\alpha.[\alpha]M & \overset{r}{\to} M \quad \text{if } \alpha \text{ not free in } M \\ (let_{var}) & z(\textbf{let } x = M \textbf{ in } N) & \overset{r}{\to} \textbf{let } x = M \textbf{ in } zN \\ (\mu_{var}) & z(\mu\alpha.C) & \overset{r}{\to} \mu\alpha.C[\alpha \leftarrow [\alpha](z[\,])] \end{array}$$

Fig. 3. The full $\lambda\mu\mathsf{tp}_{CBV}$-calculus

Note again the redundancy of (let_{var}) and (μ_{var}) which are here for the confluence. ∎

As head reduction can be extended with the new rules, we obtain a notion of standardization for this calculus too.

The rule let_{μ} is here to deal with hidden μ-redexes, obfuscated by a **let**. As an example, the term $(\textbf{let } x = yz \textbf{ in } \mu\alpha.[\alpha]\lambda x.x)t$ needs first to be reduced to $(\mu\alpha.[\alpha]\textbf{let } x = yz \textbf{ in } \lambda x.x)t$, then only the reduction between $\mu\alpha$ and $[\,]t$ can occur.

It is also interesting to see that being in call-by-value, we immediately dodge any problems with David and Py's critical pair [13]. The term $\lambda x.(\mu\alpha.c)x$ only reduces to $\lambda x.(\mu\alpha.c[\alpha \leftarrow [\alpha][\,]x])$, because of the η restriction to values.

We say that M evaluates to V in $\lambda\mu\mathsf{tp}_{CBV}$ if $[\mathsf{tp}]M$ reduces to $[\mathsf{tp}]V$ and we say that a reduction relation $\to$ is *operationally complete* if whenever a closed M is not a value, $[\mathsf{tp}]M$ is reducible along $\to$.

2.2 Normal forms

Not only do we keep the confluence, telling us that the extension is reasonable, but we also keep all the other good properties of the intuitionistic calculus. The grammar generating normal forms with respect to the operational and structural rules, with T as an entry point, and the typing rules associated with the new μ construction are given in Figure 4 (T is a global parameter of the type system corresponding to the type of tp and all rules are generalized with a context Δ on the right). Properties on the normal forms are preserved, as the proposition below says.

$$\begin{array}{rcl} S & ::= & x \mid ST \\ Q & ::= & \lambda x.T \mid S \mid \textbf{let } x = S \textbf{ in } Q \\ T & ::= & Q \mid \mu\alpha.[\beta]Q \end{array} \qquad \frac{\Gamma \vdash u : N \mid \beta : M, \alpha : N, \Delta}{\Gamma \vdash \mu\beta.[\alpha]u : M \mid \alpha : N, \Delta} \qquad \frac{\Gamma \vdash u : T \mid \beta : M, \Delta}{\Gamma \vdash \mu\beta.[\mathsf{tp}]u : M \mid \Delta}$$

Fig. 4. Normal forms and new typing rules for the $\lambda\mu\mathsf{tp}_{CBV}$-calculus

Proposition 10. *The $\lambda\mu\mathsf{tp}_{CBV}$-calculus equipped with its set of operational rules is operationally complete and the $\lambda\mu\mathsf{tp}_{CBV}$-calculus equipped with its sets of operational and structural rules is structurally complete.*

If T is a term in normal form relatively to the operational and structural rules of $\lambda\mu\mathsf{tp}_{CBV}$-calculus and $\Gamma \vdash T : A \mid \Delta$ then its proof satisfies the subformula property.

Proof (Indication). The first point is direct by the same proof as for the intuitionistic case. The second point is solved by induction, with most of the cases not being changed.

2.3 Equational theory

In [3], Sabry and Felleisen devised an equational theory of call-by-value λ-calculus with control operators that is sound and complete with respect to call-by-value continuation-passing-style (cps) semantics. We recall only the part of the equations concerning the control operators below, because the other one is verified by Moggi's calculus and so by λ_{CBV}.

$$\begin{array}{lll}
(C_{current}) & callcc\ (\lambda k.kM) & = callcc\ (\lambda kM) \\
(C_{elim}) & callcc\ (\lambda d.M) & = M \text{ if } d \notin FV(M) \\
(C_{lift}) & E[callcc\ M] & = callcc\ (\lambda k.E[M(\lambda f.(kE[f]))]) \\
 & & \quad \text{if } k, f \notin FV(E, M) \\
(C_{abort}) & callcc\ (\lambda k.C[E[kM]]) & = callcc\ (\lambda k.C[kM]) \\
 & & \quad \text{for } C \text{ a term with a hole not binding } k \\
(C_{tail}) & callcc\ (\lambda k.((\lambda z.M)N)) & = \lambda z.(callcc\ (\lambda k.M))N \text{ if } k \notin FV(N) \\
(\mathcal{A}bort) & E[\mathcal{A}M] & = \mathcal{A}M
\end{array}$$

Because the language of the equations is not the same as ours, we need some translations: *callcc* and $\mathcal{A}$ are encoded by the terms $(\lambda x.\mu\alpha.[\alpha](x(\lambda y.\mu\delta.[\alpha]y)))$ and $\lambda x.\mu\delta.[tp]x$ while, conversely, we encode $\mu\alpha.[\beta]M$ by *callcc* $(\lambda k_\alpha.\mathcal{A}(k_\beta M))$, $\mu\alpha.[\mathsf{tp}]M$ by *callcc* $(\lambda k_\alpha.\mathcal{A}(M))$ and **let** $x = N$ **in** M by $(\lambda x.M)N$.

By unfolding the definitions and doing basic calculations, we obtain the equational correspondence with the equations of Sabry and Felleisen. But instead of only having equations, we now have an oriented reduction system.

Proposition 11. *There is an equational correspondence at the level of closed expressions between $\lambda\mu\mathsf{tp}_{CBV}$ and Sabry and Felleisen's axiomatization of λ-calculus with callcc and $\mathcal{A}$.*

Moreover, since we can equip $\lambda\mu\mathsf{tp}_{CBV}$ with a cps-semantics that matches the one considered in Sabry and Felleisen as soon as $\mu\alpha.c$ is interpreted by $\lambda k_\alpha.c$, $[\alpha]M$ by Mk_α and $[\mathsf{tp}]M$ by $M\lambda x.x$, we can transfer Sabry and Felleisen completeness result to the full theory of $\lambda\mu\mathsf{tp}_{CBV}$.

Corollary 1. *The full calculus $\lambda\mu\mathsf{tp}_{CBV}$ is sound and complete with respect to $\beta\eta$ along its cps-semantics.*

Since tp has a passive role in the reduction system of $\lambda\mu\mathsf{tp}_{CBV}$, the confluence of the different subsystems and the structural and cps completenesses also hold for $\lambda\mu_{CBV}$ which is $\lambda\mu\mathsf{tp}_{CBV}$ without tp.

Summary

We studied the equational theory of call-by-value λ-calculus and $\lambda\mu$-calculus from a reduction point of view and provided what seems to be the first confluent rewrite systems for $\lambda\mu$-calculus with control that is complete with respect to the continuation-passing-style semantics of call-by-value λ-calculus.

The rewrite system we designed is made of three independent blocks that respectively address operational completeness (ability to evaluate closed terms), structural completeness (ability to contract hidden redexes in open terms) and purely observational properties.

The notion of structural completeness is related to the ability to enforce the subformula property in simply-typed λ-calculus but we failed to find a purely computational notion that universally captures the subformula property. For instance, any call-by-value reduction system that does not use a **let** and does not smash abstraction and application from redexes of the form $(\lambda x.M)(yz)$ cannot be accompanied with a typing system whose normal forms type derivation satisfies the subformula property.

It would have been desirable to characterize the block of observational rules as a block of rules providing observational completeness, in a way similar to the call-by-name control-free case where β provides operational completeness and η provides observational completeness. Obtaining such a result would however require a Böhm-style separability result for call-by-value λ-calculus and $\lambda\mu$-calculus, what goes beyond the scope of this study.

Regarding intuitionistic call-by-value λ-calculus we slightly refined Moggi and Sabry and Felleisen rewrite systems by precisely identifying which rules pertain to the operational block, the structural block or the observational block.

Finally, we proposed a new definition of standard reduction sequences for call-by-value λ-calculus that ensures the uniqueness of standard reduction paths between two terms (when such a path exists).

References

1. Plotkin, G.D.: Call-by-name, call-by-value and the lambda-calculus. Theor. Comput. Sci. **1** (1975) 125–159
2. Moggi, E.: Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Edinburgh Univ. (1988)
3. Sabry, A., Felleisen, M.: Reasoning about programs in continuation-passing style. Lisp and Symbolic Computation **6**(3-4) (1993) 289–360
4. Hofmann, M.: Sound and complete axiomatisations of call-by-value control operators. Mathematical Structures in Computer Science **5**(4) (1995) 461–482
5. Dezani-Ciancaglini, M., Giovannetti, E.: From Böhm's theorem to observational equivalences: an informal account. Electr. Notes Theor. Comput. Sci. **50**(2) (2001)
6. Curien, P.L., Herbelin, H.: The duality of computation. In: Proceedings of ICFP 2000. SIGPLAN Notices 35(9), ACM (2000) 233–243
7. Herbelin, H.: C'est maintenant qu'on calcule: au cœur de la dualité. Habilitation thesis, University Paris 11 (December 2005)
8. Gentzen, G.: Untersuchungen über das logische Schließen. Mathematische Zeitschrift **39** (1935) 176–210,405–431 English Translation in The Collected Works of Gerhard Gentzen, Szabo, M. E. editor, pages 68-131.
9. Parigot, M.: Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In: International Conference LPAR '92 Proceedings, St. Petersburg, Russia, Springer-Verlag (1992) 190–201
10. Regnier, L.: Une équivalence sur les lambda-termes. Theor. Comput. Sci. **126**(2) (1994) 281–292
11. Sabry, A., Wadler, P.: A reflection on call-by-value. ACM Trans. Program. Lang. Syst. **19**(6) (1997) 916–941
12. Ariola, Z.M., Herbelin, H.: Control reduction theories: the benefit of structural substitution. Journal of Functional Programming **18**(3) (May 2008) 373–419 with a historical note by Matthias Felleisen.
13. David, R., Py, W.: Lambda-mu-calculus and Böhm's theorem. J. Symb. Log. **66**(1) (2001) 407–413