Open Call-by-Value

Beniamino Accattoli
INRIA, UMR 7161, LIX, École Polytechnique
beniamino.accattoli@inria.fr

Giulio Guerrieri
Aix-Marseille Université, Centrale Marseille, I2M UMR 7373
giulio.guerrieri@univ-amu.fr, gguerrieri@uniroma3.it

The elegant theory of the call-by-value lambda-calculus relies on weak evaluation and closed terms, that are natural hypotheses in the study of programming languages. To model proof assistants, however, strong evaluation and open terms are required, and it is well known that the operational semantics of call-by-value becomes problematic in this case, as first pointed out by Paolini and Ronchi della Rocca. Here we study the intermediate setting—that we call Open Call-by-Value—of weak evaluation with open terms, on top of which Grégoire and Leroy designed the abstract machine of Coq. Various calculi for Open Call-by-Value already exist, coming from logical, semantical, or implementative points of view, each one with its pros and cons. This paper presents a detailed comparative study of their operational semantics. First, we show that all calculi are equivalent from a termination point of view, justifying the slogan Open Call-by-Value. Second, we compare their equational theories. Third, we present a detailed quantitative analysis of the time cost model. Four, we introduce a new simple abstract machine, and prove it a reasonable implementation of Open Call-by-Value with respect to its cost model. Along the way, there emerges a sharp deconstruction of call-by-value evaluation and of the complexity of its implementations.

Plotkin's call-by-value λ-calculus [18] is at the heart of programming languages such as Ocaml and proof assistants such as Coq. In the study of programming languages, call-by-value (CBV) evaluation is usually *weak*, *i.e.* not under abstractions, and terms are assumed to be *closed*. These constraints give rise to a beautiful theory—let us call it *Closed CBV*—having the following *harmony property*, that relates rewriting and normal forms:

Closed normal forms are values (and values are normal forms)

Harmony expresses a form of internal completeness with respect to unconstrained β-reduction: the restriction to CBV β-reduction (referred to as β_v*-reduction*) has an impact on the order in which redexes are evaluated, but evaluation never gets stuck, as every β-redex will eventually become a β_v-redex and be fired (unless evaluation diverges).

It often happens, however, that one needs to go beyond the perfect setting of Closed CBV, by considering *Strong CBV*, where reduction under abstractions is allowed, or the intermediate setting of *Open CBV*, where evaluation is weak but terms are not necessarily closed. The need arises, most notably, when trying to describe the implementation model of Coq [8], but also from other motivations, as denotational semantics [17, 20, 2, 5], monad and CPS translations and the associated equational theories [15, 21, 22, 7, 11], bisimulations [13], partial evaluation [12], linear logic proof nets [1], cost models [3].

Naïve Open CBV In call-by-name (CBN) turning to open terms or strong evaluation is harmless because CBN does not impose any special form to the arguments of β-redexes. On the contrary, turning to Open or Strong CBV is delicate. If one simply considers Plotkin's weak β_v-reduction on open terms—let us call it *Naïve Open CBV*—then harmony does no longer hold, as there are open β-normal forms that are not values (*i.e.* not a variable nor an abstraction), *e.g.* xx, $x(\lambda y.y)$, $x(yz)$ or xyz. As a consequence, there are *stuck* β*-redexes* such as $(\lambda y.t)(xx)$, *i.e.* β-redexes that will never be fired because their argument

Submitted to:
DICE 2016

is normal, but it is not a value, nor will it ever become one. Such stuck β-redexes are a disease typical of (Naïve) Open CBV, but they spread to Strong CBV as well, because evaluating under abstraction forces to deal with locally open terms (*e.g.* the variable x is locally open with respect to $(\lambda y.t)(xx)$ in $s = \lambda x.((\lambda y.t)(xx))$, even if s is closed).

The real issue with stuck β-redexes is that they prevent the creation of other redexes, and provide *premature* β_v-normal forms. The issue is serious, as it can affect termination, and thus impact on notions of observational equivalence. Let $\delta = \lambda x.(xx)$. The problem is exemplified by the terms t and u in Eq. (1):

$$t := ((\lambda y.\delta)(zz))\delta \qquad\qquad u := \delta((\lambda y.\delta)(zz)). \tag{1}$$

In Naïve Open CBV, t and u are premature β_v-normal forms because they both have a stuck β-redex forbidding evaluation to keep going, while one would expect them to behave like the famous divergent term $\Omega := \delta\delta$ (see [17, 20, 2, 1, 5, 10]).

Open CBV Starting with the seminal work of Paolini and Ronchi Della Rocca [17, 16, 20], the dissonance between open terms and CBV has been repeatedly pointed out and studied *per se* via various calculi [8, 2, 1, 5, 10, 9, 3]. An important point is that the focus of most of these works is on Strong CBV. Studies about monad, CPS, and logical translations [15, 21, 22, 14, 7, 11] introduced further proposals.

These solutions inevitably extend β_v-reduction with some other rule(s) or constructor (as `let`-expressions) to deal with stuck β-redexes. They arise from different perspectives and each one has its pros and cons. By design, these calculi (when looked at in the context of Open CBV) are never observationally equivalent to Naïve Open CBV, as they all manage to (re)move stuck β-redexes and may diverge when Naïve Open CBV is instead stuck. Each one of these calculi, however, has its own notion of evaluation and normal form, and their mutual relationships are not evident.

The aim of this paper is to draw the attention of the community on Open CBV. We believe that it is somewhat deceiving that the mainstream operational theory of CBV, however elegant, has to rely on closed terms, because it restricts the modularity of the framework, and raises the suspicion that the true essence of CBV has yet to be found. There is a real gap, indeed, between Closed and Strong CBV, as Strong CBV cannot be seen as an iteration of Closed CBV under abstractions because such an iteration has to deal with open terms. To improve the implementation of Coq [8], Grégoire and Leroy see Strong CBV as the iteration of the intermediate case of Open CBV, but they do not explore its theory. Here we exalt their point of view, providing a thorough operational study of Open CBV, as well as of the differences with respect to Closed/Strong CBV. Motivations for insisting on Open CBV rather than Strong CBV are:

1. The issue with stuck β-redexes and premature β_v-normal forms is already visible in Open CBV;
2. Open CBV has a simpler rewriting theory than Strong CBV;
3. Our previous studies of Strong CBV in [2] and [5] naturally organized themselves as properties of Open CBV that were lifted to Strong CBV by a simple iteration under abstractions.

Our contributions are along two axes:

1. *Equivalence of the Proposals*: we show that the proposed generalizations of Naïve Open CBV are equivalent, in the sense that they have exactly the same sets of normalizing and diverging λ-terms. Therefore, *there is just one notion of Open CBV*, independently of its specific syntactic incarnation. We compare the equational theories of the various proposals, and indicate the finest one for Open CBV.
2. *Cost Models and an Abstract Machine*: the termination results are complemented with quantitative analyses. Moreover we provide insights into the size-explosion problem for Closed/Open/Strong CBV, that lead to a new abstract machine for Open CBV, simpler than others in the literature.

Equivalence of the Proposals We focus on three proposals for Open CBV, as other solutions, *e.g.* Moggi's [15] or Herbelin and Zimmerman's [11], are already known to be equivalent to these ones:

1. *The Fireball Calculus* $\lambda_{\mathtt{fire}}$, that extend values to *fireballs* by adding so-called *inert terms* in order to restore harmony—it was introduced without a name by Paolini and Ronchi Della Rocca [17, 20], then rediscovered independently first by Leroy and Grégoire [8] to improve the implementation of Coq, and then by Accattoli and Sacerdoti Coen [3] to study cost models;
2. *The Value Substitution Calculus* $\lambda_{\mathtt{vsub}}$, coming from the linear logic interpretation of CBV and using explicit substitutions and contextual rewriting rules to circumvent stuck β-redexes—it was introduced by Accattoli and Paolini [2] and it is a graph-free presentation of proof nets for CBV λ-calculus [1];
3. *The Shuffling Calculus* $\lambda_{\mathtt{sh}}$, that has rules to shuffle constructors, similar to Regnier's σ-rules for CBN [19], as an alternative to explicit substitutions—it was introduced by Carraro and Guerrieri [5] (and further analysed in [10, 9]) to study the adequacy of Open/Strong CBV with respect to denotational semantics related to linear logic.

The termination equivalences proved in the paper are more or less expected. Nonetheless we think they are interesting, at least for the following reasons:

1. *Uniform View*: we provide a new uniform view on a known problem, that will hopefully avoid further proliferations of CBV calculi for open/strong settings.
2. *Simple Rewriting Theory*: the relationships between the systems are developed using basic rewriting concepts. The technical development is simple, according to the best tradition of the CBV λ-calculus, and yet it provides a sharp and detailed decomposition of Open CBV evaluation.
3. *Connecting Different Worlds*: while $\lambda_{\mathtt{fire}}$ is related to Coq and implementations, $\lambda_{\mathtt{vsub}}$ and $\lambda_{\mathtt{sh}}$ have a linear logic background. With respect to linear logic, $\lambda_{\mathtt{vsub}}$ has been used for syntactical studies while $\lambda_{\mathtt{sh}}$ for semantical ones. Our results therefore establish bridges between these different (sub)communities.

Equational Theories We compare the equational theories of these three calculi. In contrast to termination, the calculi are all equationally different. The theory of $\lambda_{\mathtt{sh}}$ is strictly contained in that of $\lambda_{\mathtt{vsub}}$—*i.e.* the one induced by linear logic proof nets—in turn strictly contained in the theory of $\lambda_{\mathtt{fire}}$. We show, however, that the latter is not stable by context closure, *i.e.* it is not a congruence, and so it equates too much. Moreover, we show how to generalize $\lambda_{\mathtt{sh}}$ so as to obtain exactly the same equational theory of $\lambda_{\mathtt{vsub}}$, that is—in our opinion—*the* equational theory of Open CBV.

Cost Models and an Abstract Machine The number of β_v-steps is the canonical time cost model of Closed CBV, as first proved by Blelloch and Greiner [4, 23, 6]. In last year's LICS [3], Accattoli and Sacerdoti Coen generalized this result, showing that the number of steps in $\lambda_{\mathtt{fire}}$ is a reasonable cost model for Open CBV. Here we show that the number of steps in $\lambda_{\mathtt{vsub}}$ is linearly related to the number of steps in $\lambda_{\mathtt{fire}}$, and so it provides a reasonable cost model as well. For $\lambda_{\mathtt{sh}}$ we obtain a similar but strictly weaker result, due to some structural difficulties suggesting that the shuffling calculus is less apt to complexity analyses.

We then analyse the *size-explosion problem*, that is the degenerate behavior making the study of cost models for λ-calculi a delicate issue. We provide insights and refinements of some of the results of [3]. Our contributions with respect to cost models are:

1. *Open vs Strong Size-Explosion, a Simpler Abstract Machine*: we show that iterating Open CBV under abstractions introduces a malicious behavior that is not visible without iterations. In fact, the solution for such an issue is known from [3]. We show that the sophisticated techniques developed in [3]

are in some sense not needed: if one tolerates a slight asymptotic slowdown (*i.e.* a quadratic rather than linear overhead with respect to the size of the initial term), a much simpler solution is possible. We then introduce an abstract machine, called Easy GLAMOUr, that is proved to be reasonable (*i.e.* its overhead is proved to be polynomial in the number of fireball steps) and that in contrast to the reasonable machines in [3] does not need to refine the environment with labels. We also provide a fine analysis of how different implementation techniques impact on the complexity of the corresponding machines.

2. *Minimality of the Cost Model*: the time cost model of Open CBV is the number of fireball steps. Fireballs extend values with inert terms, but reasonable implementations as the Easy GLAMOUr handle β-steps involving inert terms in constant time. It is natural to wonder if the cost model for Open CBV can simply be the number of steps *by value* (instead of *by fireball*, *i.e. by value* plus *by inert term*). We give a sort of negative result by exhibiting a family of terms that evaluate to normal form in n steps by value plus $O(2^n)$ steps by inert term, which shows that steps by inert term cannot be ignored, unless a radically different evaluation algorithm able to work up to inert redexes will be discovered.

Summing Up Our work is a collection of qualitative, quantitative, and implementative results about Open CBV. A further contribution, we believe, is the full picture, *i.e.* the recognition of Open CBV as a rich framework, meant to model a CBV discipline for the implementation of proof assistants, simpler than Strong CBV, and grounded in linear logic. From our comparative study it emerges that $\lambda_{\texttt{vsub}}$ is the most versatile of the studied calculi, and might be taken as the reference presentation of Open CBV. Different incarnations of Open CBV, however, serve different purposes, and the relationships established here provide a flexible framework to transfer concepts and result from one incarnation to the other.

This work has been submitted to LICS 2016. A preprint version of this work, with proofs, is available at `http://www.pps.univ-paris-diderot.fr/~giuliog/opencbv.pdf`.

References

[1] Beniamino Accattoli (2015): *Proof nets and the call-by-value λ-calculus*. *Theor. Comput. Sci.* 606, pp. 2–24.

[2] Beniamino Accattoli & Luca Paolini (2012): *Call-by-Value Solvability, revisited*. In: *FLOPS*, pp. 4–16.

[3] Beniamino Accattoli & Claudio Sacerdoti Coen (2015): *On the Relative Usefulness of Fireballs*. Accepted at LICS 2015.

[4] Guy E. Blelloch & John Greiner (1995): *Parallelism in Sequential Functional Languages*. In: *FPCA*, pp. 226–237.

[5] Alberto Carraro & Giulio Guerrieri (2014): *A Semantical and Operational Account of Call-by-Value Solvability*. In: *FOSSACS 2014*, pp. 103–118.

[6] Ugo Dal Lago & Simone Martini (2008): *The weak lambda calculus as a reasonable machine*. *Theor. Comput. Sci.* 398(1-3), pp. 32–50. Available at `http://dx.doi.org/10.1016/j.tcs.2008.01.044`.

[7] Roy Dyckhoff & Stéphane Lengrand (2007): *Call-by-Value lambda-calculus and LJQ*. *J. Log. Comput.* 17(6), pp. 1109–1134.

[8] Benjamin Grégoire & Xavier Leroy (2002): *A compiled implementation of strong reduction*. In: *(ICFP '02)*, pp. 235–246.

[9] Giulio Guerrieri (2015): *Head reduction and normalization in a call-by-value lambda-calculus*. In: *WPTE 2015*, pp. 3–17.

[10] Giulio Guerrieri, Luca Paolini & Simona Ronchi Della Rocca (2015): *Standardization of a Call-By-Value Lambda-Calculus*. In: *TLCA 2015*, pp. 211–225.

[11] Hugo Herbelin & Stéphane Zimmermann (2009): *An Operational Account of Call-by-Value Minimal and Classical lambda-Calculus in "Natural Deduction" Form*. In: *TLCA*, pp. 142–156. Available at `http://dx.doi.org/10.1007/978-3-642-02273-9_12`.

[12] Neil D. Jones, Carsten K. Gomard & Peter Sestoft (1993): *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[13] Soren Lassen (2005): *Eager Normal Form Bisimulation*. In: *LICS 2005*, pp. 345–354.

[14] John Maraist, Martin Odersky, David N. Turner & Philip Wadler (1999): *Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus*. *Theor. Comput. Sci.* 228(1-2), pp. 175–210. Available at `http://dx.doi.org/10.1016/S0304-3975(98)00358-2`.

[15] Eugenio Moggi (1989): *Computational Lambda-Calculus and Monads*. In: *LICS '89*, pp. 14–23.

[16] Luca Paolini (2002): *Call-by-Value Separability and Computability*. In: *ICTCS*, pp. 74–89.

[17] Luca Paolini & Simona Ronchi Della Rocca (1999): *Call-by-value Solvability*. *ITA* 33(6), pp. 507–534. Available at `http://dx.doi.org/10.1051/ita:1999130`.

[18] Gordon D. Plotkin (1975): *Call-by-Name, Call-by-Value and the lambda-Calculus*. *Theor. Comput. Sci.* 1(2), pp. 125–159. Available at `http://dx.doi.org/10.1016/0304-3975(75)90017-1`.

[19] Laurent Regnier (1994): *Une équivalence sur les lambda-termes*. *Theoretical Computer Science* 2(126), pp. 281–292.

[20] Simona Ronchi Della Rocca & Luca Paolini (2004): *The Parametric λ-Calculus*. Springer Berlin Heidelberg.

[21] Amr Sabry & Matthias Felleisen (1993): *Reasoning about Programs in Continuation-Passing Style*. *Lisp and Symbolic Computation* 6(3-4), pp. 289–360.

[22] Amr Sabry & Philip Wadler (1997): *A Reflection on Call-by-Value*. *ACM Trans. Program. Lang. Syst.* 19(6), pp. 916–941.

[23] David Sands, Jörgen Gustavsson & Andrew Moran (2002): *Lambda Calculi and Linear Speedups*. In: *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, pp. 60–84.