Evaluation in the computational calculus is non-confluent

Claudia Faggian[1], Giulio Guerrieri[2], and Riccardo Treglia[3]

[1] Université de Paris, IRIF, CNRS, F-75013 Paris, France
faggian@irif.fr
[2] University of Bath, Department of Computer Science, Bath, UK
giulio.guerrieri@gmail.com
[3] Università di Torino, Department of Computer Science, Turin, Italy
riccardo.treglia@unito.it

Abstract

In Moggi's computational calculus, reduction is the contextual closure of the rules obtained by orienting three monadic laws. In the literature, evaluation is usually defined as the closure under weak contexts (no reduction under binders): $\mathsf{E} = \langle\rangle \mid \mathsf{let}\, x \coloneqq \mathsf{E}\, \mathsf{in}\, M$.

We show that, when considering all the monadic rules, weak reduction is non-deterministic, non-confluent, and normal forms are not unique. However, when interested in returning a value (convergence), the only necessary monadic rule is β, whose evaluation is deterministic.

The *computational λ-calculus*, noted λ_c, was introduced by Moggi [11, 12, 13] as a meta-language to describe computational effects in programming languages. Since then, computational λ-calculi have been developed as foundations of programming languages, formalizing both functional and effectful features [20, 1, 16, 9, 2], in a still active line of research.

To model effectful features at a semantic level, Moggi used the categorical notion of *monad*. A monad can be equivalently presented as a Kleisli triple satisfying three identities [13, 10]. At an operational level, Moggi [11] internalized these identities into the syntax of λ_c, giving rise to three conversion rules—called *monadic laws*—that are added to the usual β and η rules.

Nowadays the literature is rich of computational calculi that refine Moggi's λ_c. Such calculi are presented in at least three different fashions: fully equational systems [9, 15] (all conversion rules are unoriented identities); hybrid systems where β (and η, if considered) are oriented rules while the monadic laws are identities on terms [2]; reduction systems where every rule is oriented [17]. Here we follow the latter approach, which brings to the fore operational aspects of reduction and evaluation which seem to have been neglected in the literature.

Indeed, in the literature of calculi with effects [9, 2], *evaluation* is usually *weak*, that is, it is not allowed in the scope of the binders (λ or let). This is the way evaluation is implemented by functional programming languages such as Haskell and OCaml. Moreover, only β and $\mathsf{let}.\beta$ are considered. However, in Moggi's λ_c and in Sabry and Wadler's [17], the reduction is full, that is, reduction is the compatible closure of all the monadic rules. When considering *all the rules*, we observe—quite unexpectedly—that evaluation (*i.e.* weak reduction) is *non-deterministic, non-confluent, and normal forms are not unique.*

Reduction and Evaluation. Here we focus on a computational λ-calculus which is standard in the literature, namely Sabry and Wadler's λ_{ml^*} [17]. This is a neat and compact refinement of Moggi's untyped λ_c [11]—the relation between the two calculi is formalized by a reflection[17].

λ_{ml^*}—which we display in Figure 1— has a two sorted syntax that separates *values* (*i.e.* variables and abstractions) and *computations*. The latter are either *let*-expressions (aka explicit substitutions, capturing monadic binding), or applications (of values to values), or coercions $[V]$ of values V into computations (corresponding to the `return` operator in Haskell).

$$\begin{array}{lrcl} \textit{Values:} & V, W & ::= & x \mid \lambda x.M \\ \textit{Computations:} & M, N & ::= & [V] \mid \mathsf{let}\, x \coloneqq M \,\mathsf{in}\, N \mid VW \end{array}$$

Reduction rules:

$$\begin{array}{rrcll} (\beta) & (\lambda x.M)V & \mapsto_\beta & M[V/x] & \\ (\eta) & \lambda x.Vx & \mapsto_\eta & V & x \notin \mathsf{fv}(V) \\ (\mathsf{let}.\beta) & \mathsf{let}\, x \coloneqq [V] \,\mathsf{in}\, N & \mapsto_{\mathsf{let}.\beta} & N[V/x] & \\ (\mathsf{let}.\eta) & \mathsf{let}\, x \coloneqq M \,\mathsf{in}\, [x] & \mapsto_{\mathsf{let}.\eta} & M & \\ (\mathsf{let}.\mathit{ass}) & \mathsf{let}\, y \coloneqq (\mathsf{let}\, x \coloneqq L \,\mathsf{in}\, M) \,\mathsf{in}\, N & \mapsto_{\mathsf{let}.\mathit{ass}} & \mathsf{let}\, x \coloneqq L \,\mathsf{in}\, (\mathsf{let}\, y \coloneqq M \,\mathsf{in}\, N) & x \notin \mathsf{fv}(N) \end{array}$$

Figure 1: λ_{ml^*}: Syntax and Reduction

- The *reduction rules* in λ_{ml^*} are the usual β (and η) rules from Plotkin's *call-by-value* λ-calculus [14], plus the oriented version of the three *monadic laws*: $\mathsf{let}.\beta$, $\mathsf{let}.\eta$, $\mathsf{let}.\mathit{ass}$ (see Figure 1).
- *Reduction* $\to$ is the *contextual closure* of the reduction rules.

Following standard practice, we define *evaluation* $\xrightarrow[\mathsf{w}]{}_{ml^*}$ (aka *sequencing*) as the closure of the rules under *evaluation context* E:

$$\mathsf{E} \ ::= \ \langle\,\rangle \ \mid \ \mathsf{let}\, x \coloneqq \mathsf{E} \,\mathsf{in}\, N \qquad\qquad \text{evaluation context}$$

Informally, the operational understanding of weak reduction is that evaluating $\mathsf{let}\, x \coloneqq M \,\mathsf{in}\, N$ amounts to first evaluate M until it returns a value, that is, until a computation of the form $[V]$ is reached. Then V is passed to N by substituting V for x in N, thanks to the rule $\mathsf{let}.\beta$.

Despite the prominent role that weak reduction has in the literature of calculi with effects, its reduction properties are somehow surprising. While full reduction $\to_{ml^*}$ is confluent, the closure of the rules under *evaluation context* turns out to be *non-deterministic*, *non-confluent*, and its *normal forms* are *not unique*.

Note that such issues only come from the monadic rules $\mathsf{let}.\eta$ and $\mathsf{let}.\mathit{ass}$ (sometimes called *identity* and *associativity*, respectively, in the literature), not from β or $\mathsf{let}.\beta$. It is worth to clarify that while the literature on computational λ-calculi often adopts weak reduction (see for instance, [9, 2], where a big-step variant is used), the rules $\mathsf{let}.\mathit{ass}$ and $\mathsf{let}.\eta$ are usually dealt with as *unoriented* identities—the only oriented rules being β and $\mathsf{let}.\beta$.

(Non-)Confluence. In λ_{ml^*}, the reduction $\to_{ml^*}$ is confluent, but weak reduction $\xrightarrow[\mathsf{w}]{}_{ml^*}$ is not. We now give some examples. For every $\gamma \in \{\beta, \eta, \mathsf{let}.\beta, \mathsf{let}.\eta, \mathsf{let}.\mathit{ass}\}$, the *weak γ-reduction* $\xrightarrow[\mathsf{w}]{}_\gamma$ is the closure of the rule $\mapsto_\gamma$ under *weak contexts* E.

Example 1 (Non-confluence)**.** Let M be a computation in normal form, for instance $M = xx$.

$$\begin{array}{ccc} \mathsf{let}\, y \coloneqq (\mathsf{let}\, x \coloneqq zz \,\mathsf{in}\, M) \,\mathsf{in}\, [y] & \xrightarrow[\mathsf{w}]{\mathsf{let}.\eta} & \mathsf{let}\, x \coloneqq zz \,\mathsf{in}\, M \\ {\scriptstyle \mathsf{let}.\mathit{ass}}\big\downarrow{\scriptstyle \mathsf{w}} & & \\ \mathsf{let}\, x \coloneqq zz \,\mathsf{in}\, (\mathsf{let}\, y \coloneqq M \,\mathsf{in}\, [y]) & & \end{array}$$

Both $\mathsf{let}\, x \coloneqq zz \,\mathsf{in}\, M$ and $\mathsf{let}\, x \coloneqq zz \,\mathsf{in}\, (\mathsf{let}\, y \coloneqq M \,\mathsf{in}\, [y])$ are normal for $\xrightarrow[\mathsf{w}]{}_{ml^*}$ (in the latter, the $\mathsf{let}.\eta$-redex $\mathsf{let}\, y \coloneqq M \,\mathsf{in}\, [y]$ cannot be fired by weak reduction), but they are distinct.

Example 2 (Non-confluence)**.** Let $R = P = Q = L = zz$ and:

$$M \;\coloneqq\; \overline{\mathtt{let}\ z = \underline{(\mathtt{let}\ x = (\mathtt{let}\ y = L\ \mathtt{in}\ Q)\ \mathtt{in}\ P)}\ \mathtt{in}\ R}$$

There are two weak $\mathsf{let}.ass$-redexes, the overlined one and the underlined one. So,

$$\begin{aligned} M &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, x \coloneqq (\mathsf{let}\, y \coloneqq L \,\mathsf{in}\, Q) \,\mathsf{in}\, (\mathsf{let}\, z \coloneqq P \,\mathsf{in}\, R) \\ &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, y \coloneqq L \,\mathsf{in}\, (\mathsf{let}\, x \coloneqq Q \,\mathsf{in}\, (\mathsf{let}\, z \coloneqq P \,\mathsf{in}\, R)) \eqqcolon M' \end{aligned}$$

$$\begin{aligned} M &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, z \coloneqq (\mathsf{let}\, y \coloneqq L \,\mathsf{in}\, (\mathsf{let}\, x \coloneqq Q \,\mathsf{in}\, P)) \,\mathsf{in}\, R \\ &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, y \coloneqq L \,\mathsf{in}\, (\mathsf{let}\, z \coloneqq (\mathsf{let}\, x \coloneqq Q \,\mathsf{in}\, P) \,\mathsf{in}\, R) \eqqcolon M'' \end{aligned}$$

Both M' are M'' are normal for $\underset{\mathsf{w}}{\to}_{ml^*}$ (in M'', the $\mathsf{let}.ass$-redex $\mathsf{let}\, z \coloneqq (\mathsf{let}\, x \coloneqq Q \,\mathsf{in}\, P) \,\mathsf{in}\, R$ is under the scope of a let and so cannot be fired by weak reduction), but they are distinct.

Example 3.
Non-determinism—but confluence—of $\to_{\mathsf{let}.\eta}$. Let $M = yy$ and $N = zz$:

$$\begin{tikzcd} \mathsf{let}\, x \coloneqq (\mathsf{let}\, y \coloneqq (\mathsf{let}\, z \coloneqq N \,\mathsf{in}\, [z]) \,\mathsf{in}\, M) \,\mathsf{in}\, [x] \arrow[r, "\mathsf{let}.\eta", "\mathsf{w}"'] \arrow[d, "\mathsf{let}.\eta"', "\mathsf{w}"] & \mathsf{let}\, y \coloneqq (\mathsf{let}\, z \coloneqq N \,\mathsf{in}\, [z]) \,\mathsf{in}\, M \arrow[d, "\mathsf{let}.\eta"', "\mathsf{w}"] \\ \mathsf{let}\, x \coloneqq (\mathsf{let}\, y \coloneqq N \,\mathsf{in}\, M) \,\mathsf{in}\, [x] \arrow[r, "\mathsf{let}.\eta", "\mathsf{w}"'] & \mathsf{let}\, y \coloneqq N \,\mathsf{in}\, M \end{tikzcd}$$

Summing up the situation:

1. $\underset{\mathsf{w}}{\to}_{\beta}$ and $\underset{\mathsf{w}}{\to}_{\mathsf{let}.\beta}$ and $\underset{\mathsf{w}}{\to}_{\beta,\,\mathsf{let}.\beta} \coloneqq \underset{\mathsf{w}}{\to}_{\beta} \cup \underset{\mathsf{w}}{\to}_{\mathsf{let}.\beta}$ are deterministic.
2. $\underset{\mathsf{w}}{\to}_{\mathsf{let}.\eta}$ is non-deterministic, but it is confluent.
3. $\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass}$ is non-deterministic, non-confluent and normal forms are not unique.
4. $\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \cup \underset{\mathsf{w}}{\to}_{\mathsf{let}.\beta} \cup \underset{\mathsf{w}}{\to}_{\beta}$ is non-deterministic, non-confluent and normal forms are not unique.
5. $\underset{\mathsf{w}}{\to}_{ml^*}$ is non-deterministic, non-confluent and normal forms are not unique.

(Non-)Factorization. Another remarkable aspect making the reduction theory for λ_{ml^*} (and for other computational λ-calculi) tricky to study is the lack of *factorization*, which is the simplest possible form of *standardization*.

In Plotkin's call-by-value λ-calculus [14] (which can be seen as the restriction of λ_{ml^*} where the reduction is generated only by the β-rule), weak reduction satisfies factorization, that is any reduction sequence can be *reorganized* as weak steps followed by non-weak steps:

$$\to_{\beta}^{*} \;\subseteq\; \underset{\mathsf{w}}{\to}_{\beta}^{*} \cdot \underset{\neg\mathsf{w}}{\to}_{\beta}^{*} \tag{1}$$

But in λ_{ml^*} (and similar computational λ-calculi), weak factorization *does not hold.* The problem is here the $\mathsf{let}.\eta$ rule, as shown by the following counterexample, due to van Oostrom [19].

Example 4 (Non-factorization [19]). Consider

$$M \coloneqq \mathsf{let}\, y \coloneqq (zz)\, \mathsf{in}\, (\mathsf{let}\, x \coloneqq [y]\, \mathsf{in}\, [x]) \;\; \underset{\neg\mathsf{w}}{\rightarrow}_{\mathsf{let}.\eta} \;\; \mathsf{let}\, y \coloneqq (zz)\, \mathsf{in}\, [y] \;\; \underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.\eta} \;\; (zz) =: N$$

Weak steps are not possible from M, so it is *impossible* to factorize the reduction form M to N as $M \underset{\mathsf{w}}{\rightarrow}^*_{ml^*} \cdot \underset{\neg\mathsf{w}}{\rightarrow}^*_{ml^*} N$.

A bridge between Evaluation and Reduction. On the one hand, computational λ-calculi such as λ_{ml^*} have an unrestricted *non-deterministic reduction* that generates the equational theory of the calculus, studied for foundational and semantic purposes. On the other hand, *weak reduction* has a prominent role in the literature of computational λ-calculi, because it models an ideal programming language. Indeed, when restricted to closed terms (which are the terms corresponding to programs), normal forms of weak reduction coincide with values; and when restricted to β and $\mathsf{let}.\beta$ steps, weak reduction is deterministic and corresponds to an abstract machine, implementing a programming language. It is then natural to wonder what is the relation between reduction and evaluation.

In Plotkin's call-by-value λ-calculus [14], the following *convergence result* provides a bridge between reduction and evaluation: if a term M β-reduces to a value, then M only needs *weak* β-reduction to reach a value.

$$M \rightarrow^*_\beta V \text{ (for some value } V) \iff M \underset{\mathsf{w}}{\rightarrow}^*_\beta V' \text{ (for some value } V') \tag{2}$$

In λ_{ml^*}, despite several drawbacks of weak reduction, we can still prove a *convergence* result similar to (2) relating reduction and evaluation: to reach a value in λ_{ml^*}, weak β-steps and weak $\mathsf{let}.\beta$-steps suffice.

Theorem 5 (Convergence). *Let M be a computation in λ_{ml^*} and let $\rightarrow_{ml^*_{-}} \coloneqq \rightarrow_{ml^*} \setminus \rightarrow_\eta$.*

$$M \rightarrow^*_{ml^*_{-}} [V] \textit{ (for some value } V) \iff M \underset{\mathsf{w}}{\rightarrow}^*_{\beta,\, \mathsf{let}.\beta} [V'] \textit{ (for some value } V') \tag{3}$$

Because of the issues which we have presented, this result is non-trivial. We obtain it via the study of a calculus recently introduced by de'Liguoro and Treglia's, namely the *computational core* $\lambda_©$ [4]. $\lambda_©$ has the same issues, but a different syntax, which is more closely related to calculi inspired by linear logic [18, 5, 8, 6], whose properties and tools we can then use. The analysis of the reduction theory of $\lambda_©$ is carried-out in [7]. We then transfer the convergence of $\lambda_©$ to that of λ_{ml^*}, via a rather sophisticated analysis of the translation.

Conclusion. Convergence in λ_{ml^*} relates full reduction to evaluation, and provides a theoretical justification to the following facts:

1. functional programming languages with computational effects use weak reduction as evaluation mechanism; indeed, weak reduction is enough to *return values*.

2. in computational λ-calculi, when interested in returning a value, the only *rules* of interest *for weak reduction* are β and $\mathsf{let}.\beta$—which are deterministic and do not have unpleasant rewriting properties—while the rules $\mathsf{let}.ass$ and $\mathsf{let}.\eta$ can be safely considered as unoriented identities external to the reduction.

IWC2021 vs IWC2020. We present work which has been developed after de'Liguoro and Treglia's presentation at IWC20 [3], and thanks to the interactions there. The developments benefited of the discussion at the workshop, in particular of subsequent crucial comments by Vincent van Oostrom [19], and of new collaborations prompted there. In [3], preliminary—and incomplete—work on weak factorization for de'Liguoro and Treglia's computational calculus $\lambda_{©}$ [4] was presented. Such a work has then evolved in the analysis of the reduction theory for $\lambda_{©}$ in [7]. One may wonder if the properties discovered there are specific to that specific calculus, or how that relates to the literature of computational calculi.

Here, we focus on *mainstream* and well-established formalizations of the computational calculus. We consider a standard calculus which is well-studied in the literature, namely Sabry and Wadler's λ_{ml^*} [17]. We show that the properties of non-confluence and non-factorization of evaluation which are studied in [7] actually do hold also in λ_{ml^*}—and in fact in any calculus in which the monadic rules are oriented. We find this fact quite surprising, and worth to be explicitly stated. To our knowledge, it does not appear in the literature.

Furthermore, we are able to show that the convergence result which is established in [7] transfers to λ_{ml^*}, even though the translation between the two calculi does not directly preserve weak reduction (a more sophisticated analysis is needed).

References

[1] Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In *Applied Semantics, International Summer School, APPSEM 2000*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer, 2002.

[2] Ugo Dal Lago, Francesco Gavazzo, and Paul B. Levy. Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[3] Ugo de'Liguoro and Riccardo Treglia. On the reduction of the type-free computational lambda-calculus. Presentation at the 9th International Workshop on Confluence, 2020.

[4] Ugo de'Liguoro and Riccardo Treglia. The untyped computational λ-calculus and its intersection type discipline. *Theor. Comput. Sci.*, 846:141–159, 2020.

[5] Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming (PPDP 2016)*, pages 174–187. ACM, 2016.

[6] Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021.

[7] Claudia Faggian, Giulio Guerrieri, Ugo de'Liguoro, and Riccardo Treglia. On reduction and normalization in the computational core. *CoRR*, abs/2104.10267, 2021. Submitted to Math. Struct. Comp. Sci., special issue of IWC 2020.

[8] Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two Girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 15–30, 2019.

[9] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2):182 – 210, 2003.

[10] Saunders MacLane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 2 edition, 1997.

[11] Eugenio Moggi. Computational Lambda-calculus and Monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland, October 1988.

[12] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, pages 14–23. IEEE Computer Society, 1989.

[13] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[14] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[15] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[16] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[17] Amr Sabry and Philip Wadler. A reflection on call-by-value. In Robert Harper and Richard L. Wexelblat, editors, *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996*, pages 13–24. ACM, 1996.

[18] Alex Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[19] Vincent van Oostrom. Private communication via electronic mail, 2020.

[20] Philip Wadler and Peter Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Log.*, 4(1):1–32, 2003.