Information and Computation 255 (2017) 224–242

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

# On the value of variables

Beniamino Accattoli [a,*], Claudio Sacerdoti Coen [b]

[a] INRIA, UMR 7161, LIX, École Polytechnique, France
[b] Department of Computer Science and Engineering, University of Bologna, Italy

ARTICLE INFO

*Article history:*
Received 2 April 2015
Available online 11 January 2017

*Keywords:*
λ-Calculus
Implementations of functional programming languages
Explicit substitutions
Cost models

ABSTRACT

Call-by-value and call-by-need λ-calculi are defined using the distinguished syntactic category of values. In theoretical studies, values are variables and abstractions. In more practical works, values are usually defined simply as abstractions. This paper shows that practical values lead to a more efficient process of substitution—for both call-by-value and call-by-need—once the usual hypotheses for implementations hold (terms are closed, reduction does not go under abstraction, and substitution is done in micro steps, replacing one variable occurrence at a time). Namely, the number of substitution steps becomes linear in the number of $\beta$-redexes, while theoretical values only provide a quadratic bound. We complete the picture by showing that the same quadratic / linear bounds also hold for theoretical / practical versions of call-by-name.



## 0. Introduction

The theory and the practice of functional programming languages are sometimes far apart. For instance, the theory is based on the λ-calculus, where terms may have free variables, reduction is non-deterministic (but confluent), and can take place everywhere in the term. In practice—*i.e.* in the implementation of functional languages—only closed λ-terms are considered, reduction is deterministic, and in particular it is weak, *i.e.* it does not take place under abstraction.

*Theoretical and practical values* Plotkin's call-by-value λ-calculus [27] is a theoretical object of study introduced to model a concrete case, Landin's SECD machine [20]. In such a calculus there is a primitive notion of *value* and $\beta$-redexes can fire only when the argument is a value. For Plotkin—and for most of the huge theoretical literature that followed—values are *variables* and *abstractions*; let us call them *theoretical values*. However, most CBV abstract machines (or imperative extensions of Plotkin's calculus [13]) employ a notion of *practical value* that includes abstractions and excludes variables. For instance, Paolini and Ronchi della Rocca's book [28] on the *parametric λ-calculus*, a generalization of Plotkin's calculus based on a parametric notion of value, requires that the given notion of value is theoretical (*i.e.* that it includes variables), while Pierce's book [26], driven by programming and implementations, uses practical values. Under the usual practical hypotheses—terms are closed, reduction does not go under abstraction—the difference between the two notions of value is not *extensionally* observable, as it does not affect the result of evaluation. Moreover, with a small-step operational semantics—used in most theoretical works—the difference is also not *intensionally observable*.

In practical works, however, the usual *small-step semantics* is decomposed in a *micro-step semantics*, in which substitution acts on a variable occurrence at a time, *i.e.* with the granularity of abstract machines (or of that of *substructural* operational

* Corresponding author.
*E-mail address:* Beniamino.Accattoli@inria.fr (B. Accattoli).

http://dx.doi.org/10.1016/j.ic.2017.01.003
0890-5401/

semantics [25]). We show that with micro-steps the difference between the two notions of value is *intensionally* observable: the practical variant leads to a more efficient implementation of substitution, where efficiency is measured relatively to the number of $\beta$-redexes, that is the time cost model of reference. Thus, we explain the gap between theory and practice, providing a theoretical justification for practical values.

*The linear substitution calculus* Our framework is the *Linear Substitution Calculus* (LSC) [1,6,5], a calculus with explicit substitutions refining a calculus by Robin Milner [23], that is a flexible tool in between theory and practice. It is theoretically well-founded, as it arises from graphical and logical studies on the $\lambda$-calculus, of which it is a refinement, and practically useful, as it faithfully models most environment-based abstract machines [4], and—remarkably—the number of evaluation steps in the LSC is a reasonable measure of the time complexity of a $\lambda$-term [6,7]. One of its key features is its simplicity: it can model an abstract machine using only two rules, corresponding to multiplicative and exponential cut-elimination in linear logic. The first rule, the multiplicative one $\multimap_{\mathtt{m}}$, deals with $\beta$-redexes, replacing them with explicit substitutions. The second rule, the exponential one $\multimap_{\mathtt{e}}$, replaces a single occurrence of a variable with the content of its associated explicit substitution, mimicking the micro-step mechanism at work in abstract machines.

*Contributions* In this paper we study the overhead of the substitution process in micro-step operational semantics of $\lambda$-calculus. The complexity of the overhead is obtained by bounding the number of substitution/exponential steps ($\multimap_{\mathtt{e}}$) as a function of the number of $\beta$/multiplicative steps ($\multimap_{\mathtt{m}}$). We provide bounds for the main evaluation strategies, *i.e.* call-by-name, call-by-value, and call-by-need, studying two variants for each strategy, one with practical and one with theoretical values. Uniformly, for theoretical values the bound is quadratic, while for practical values is linear, and we also provide terms reaching the bounds.

*Call-by-name (CBN)* Call-by-name does not rely on values, or, equivalently, everything, including variables, is a value. At first sight, then, CBN is theoretical. In [6], Accattoli and Dal Lago show that for theoretical CBN the overhead of substitutions is quadratic (in the number of $\beta$-steps). The worst cases, *i.e.* those reaching the quadratic bound, are given by malicious *chains of renamings*, *i.e.* of substitutions of a variable for a variable.

*Call-by-value (CBV)* In CBV, if values are *theoretical* then the overhead is quadratic, as malicious chains are still possible. On the other hand, we show that it is enough to remove variables from values—therefore switching to *practical* values—to avoid these expensive chains and obtain a *globally linear* overhead. The proof of the bound is particularly simple and, curiously, it holds only under the assumption that evaluation terminates. We also show that theoretical/practical values can be seen as different ways of closing a (benign) critical pair arising from the switch to micro-step evaluation (see Sect. 4), providing a further explanation for the discrepancy in the literature.

*Call-by-need (CBNeed)* Call-by-need evaluation is usually defined using practical values [21,11,22,12,16] and can be modularly expressed in the LSC. As for CBV, theoretical values induce a quadratic bound, while practical values provide a linear bound. The proof for the practical case, however, is inherently different. In particular, it does not rely on any termination hypotheses. We actually provide two proofs. The first one is simpler, but provides a laxer bound (linear, but with a larger constant). The second one adds labels to refine the analysis, and establishes the exact bound.

*Practical CBN* We complete the picture by studying a practical version of CBN, where the substitution rules are refined so that variables are never substituted. We prove that Practical CBN has a linear overhead, by adapting a result from the literature [29] (discussed below, among related works). Curiously, the proof follows the structure of the CBV case rather than the CBNeed case.

*Justifying practical values* One of the motivations of this work is to find a theoretical justification for practical values, that escape usual argument based on logic or rewriting. Logically, both CBV and CBNeed have a foundation in the so-called *boring* translation of $\lambda$-calculus into linear logic [18,2], but such a translation wraps both variables and abstractions inside the ! modality—the connective allowing non-linear behavior—thus enabling the substitution of both. At the rewriting level, the strategies implemented by abstract machines can be justified as being standard strategies, in the sense of the standardization theorem. Now, the strategies with practical values are not standard in the wider calculi with theoretical values, thus the switch to practical values cannot be justified that way. Our results provide an alternative explanation, based on the relative complexity of the substitution process.

*Abstract machines* The difference between the LSC and abstract machines is that the former isolates the logical and computational essence of evaluation, removing the search for the next redex implemented by the latter. This claim is made precise in a companion paper [4] by Accattoli, Barenbaum, and Mazza, that studies the relationship with several abstract machines from the literature. The result is that abstract machines and their search for the redex induce only a linear overhead with respect to the number of steps in the corresponding calculi. Via that work, our bounds apply to concrete implementation models.