HAL
open science

On the Value of Variables

Beniamino Accattoli, Claudio Sacerdoti Coen

To cite this version:

Beniamino Accattoli, Claudio Sacerdoti Coen. On the Value of Variables. Information and Computation, 2017, 255, pp.224 - 242. 10.1016/j.ic.2017.01.003 . hal-01675373

HAL Id: hal-01675373

https://hal.science/hal-01675373v1

Submitted on 4 Jan 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Value of Variables

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

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

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 hypothesis 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 β-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.

Keywords:

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 β-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 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 β-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, 9, 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 λ-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 λ-term [9, 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 β-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 λ-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 β/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 [9], Accattoli and Dal Lago show that for theoretical CBN the overhead of substitutions is quadratic (in the number of β 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 λ-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 of 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 of 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.

Linear Logic. From a linear logic perspective the bound is quite surprising. The exponentials (*i.e.* the substitutions), responsible for duplications, are expected to capture most of the computing time, while the multiplicatives are usually seen as negligible in terms of cost. One may suspect that the number of steps is not a good complexity measure, as substitution may be very costly to implement. But it is not the case here, as our exponential steps can be implemented in time linear in the size of the initial term (because of the properties of the micro-step evaluation strategy we consider), and can thus be taken as a realistic measure of complexity, see [9, 7].

Conference vs Journal Paper. This is the extended version of our previous paper presented at WoLLIC 2014 [10]. Here we provide an even simpler proof for the CBV case and add an alternative proof of the CBNeed case, whose plus is the simplicity, as it avoids labels, and whose minus is that it gives 3 rather than 2 for the constant of the linear factor. We also add the study of a practical version of CBN, and clarify some points left open in [10]. Finally, we provide a more abstract view of the problem, emphasizing the role of elementary operational properties.

Related Work. After the conference we discovered that in [29], Sands, Gustavsson, and Moran proved results similar to ours. They provide linear bounds for every evaluation mechanism, but with a quite different focus. A comparison follows:

1. *Speed-ups vs Variables*: they are interested in speed-up theorems rather than on the role of variables or on the dynamics of substitutions.
2. *Abstract Machines vs ES*: they deal with abstract machines rather than explicit substitutions. In some sense their results are stronger, because they bound also the overhead of the machine. Our study however can be modularly lifted to abstract machines, by composing the results here with those in the companion paper [4]. Moreover, the LSC is considerably simpler and more abstract than abstract machines, while it retains their asymptotic behavior (see [4] and Sect. 8).
3. *Exact Bounds*: for CBV and CBNeed we show that our bounds are exact, exhibiting terms reaching the bound.
4. *Name vs Value / Need? No*: our identification of the value of variables provides symmetric results for CBN / CBV / CBNeed, clarifying the problem. From their work indeed it seems that CBV and CBNeed have naturally lower substitution overhead than CBN. We show, however, that Theoretical CBV and CBNeed have quadratic overheads exactly as CBN.
5. *Call-by-Need*: in [29] the authors deal with CBN and CBV and refer to [24] for CBNeed, but in [24] the result for CBNeed is only outlined. Here instead we dissect its dynamics providing 2 different proofs of the linear overhead.
6. *Practical Call-by-Name*: our study of Practical CBN is adapted from the one in [29]. The differences are that our definition of the strategy is slightly lazier, see Sect. 7.
7. *Decomposed Proofs*: we add an abstract view and an in-depth analysis of the proofs. In particular, we show that, despite all mechanisms have linear bounds, the proofs in fact rely on different principles with subtle properties.

More generally, the issue of renaming chains, and the optimization used in [29] for lowering the overhead of CBN, repeatedly appear in the literature on abstract machines, often with reference to space consumption and *space leaks*, for instance in Wand's [32] (section 2), Friedman et al.'s [17] (section 4), and Sestoft's [30] (section 4). None of these works, however, provides a complexity analysis of the problem nor of the solution.

Another linear bound appears in Dal Lago and Martini's [14], where it is shown that evaluation in the CBV λ-calculus (corresponding to our $\multimap_{\mathtt{m}}$) and eval-

uation in a related graph-rewriting formalism (playing the role of the LSC, and accounting for $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$) are linearly related (and so $\multimap_{\mathtt{e}}$ is linear in $\multimap_{\mathtt{m}}$). They do not discuss the difference between theoretical and practical values, but they employ practical values at the graphical level, exactly as our results prescribe.

Summing up this paper gives a neat view over a recurrent issue in the literature on functional languages and abstract machines, whose solutions became folklore optimizations. For the first time here the issue is studied theoretically, *per se*, and thoroughly, as the problem of substitution overheads induced by the value of variables in a micro-step scenario.

Plan of the Paper. The next section explains why the value of variables is not observable with small-step evaluation. Sect. 2 shows how to obtain the quadratic bound for Theoretical CBN, recalling the results in [9]. Sect. 3 deals with Practical CBV, and Sect. 4 with the theoretical variant. In Sect. 5 we study both Theoretical and Practical CBNeed, and in Sect. 6 we provide an alternative proof for the practical case. In Sect. 7 a practical variant of CBN is analyzed. Sect. 8 concludes with a discussion about the actual complexity of implementing the calculi.

1. Overview

Small-Step and Micro-Step Evaluation. In [27], Plotkin introduces both the CBN and CBV evaluation strategies. As it is nowadays standard, terms are closed and evaluation is weak and small-step. Under such hypotheses, excluding variables from values (or from the substitutable terms in CBN) has no consequences. This fact follows from two easy observations:

1. The argument u of a β-redex is out of abstractions. If evaluation is weak and the term is closed then u cannot be a variable.
2. Closed terms reduce to closed terms.

So variables are never substituted and the value of variables is not observable.

When turning to micro-step evaluation the situation changes. Micro-steps rely on *sharing*, that in this paper is realized by *explicit substitutions*, *i.e.* an additional constructor $t[x \leftarrow u]$ representing the delayed substitution of u for x in t. Explicit substitut/ions (ES) are avatars of `let` expressions, as $t[x \leftarrow u]$ is just a compact notation for $\mathtt{let}\ x = u\ \mathtt{in}\ t$, and in particular ES are *binders*, as x is bound in t (in $t[x \leftarrow u]$).

Weak evaluation with ES necessarily goes under ES, so micro-step variants of CBN and CBV evaluate under binders. Therefore, arguments can look locally

open even if the term is globally closed, as for instance in $((\lambda y.t)x)[x{\leftarrow}I]$, (with I the identity), where $\lambda y.t$ is applied to the locally open argument x even if the term is globally closed.

The Issue with Variables. Let us give a sketch of the issue we are concerned with in this paper. Consider the term $t := x[x{\leftarrow}y][y{\leftarrow}I]$. If variables are duplicable then t would evaluate as follows (the operational semantics will be precisely defined in the next sections):

$$x[x{\leftarrow}y][y{\leftarrow}I] \to y[x{\leftarrow}y][y{\leftarrow}I] \to I[x{\leftarrow}y][y{\leftarrow}I]$$

If variables are not duplicable then the evaluation of t would rather be as follows:

$$x[x{\leftarrow}y][y{\leftarrow}I] \to x[x{\leftarrow}I][y{\leftarrow}I] \to I[x{\leftarrow}I][y{\leftarrow}I]$$

The example shows that the final result is essentially the same, as one always obtains the identity in some environment. There is a substantial difference, however. The second approach modifies the environment as to avoid future repetitions of the substitution sequence. The example does not show it, but if x later comes back in evaluating position then the first approach still needs 2 steps, while the second approach only requires one step. Now the length of the substitution sequence can be arbitrary, and repeated appearances of x can make the overhead of the first approach considerably larger than the second one, up to the point where the asymptotic complexity is affected. This paper provides precise bounds on the overhead.

Summary of the Results. In CBV and CBNeed the difference between the two approaches amounts to different definitions of values. Having variables among values—*i.e.* adopting theoretical values and thus following the first approach—induces a quadratic overhead of the substitution process. Turning to practical values and embracing the second approach, instead, speeds up the substitution process, inducing a linear overhead. In CBN there is no primitive notion of value, but the two approaches still make sense, as one may simply exclude variables from the substitutable terms. The two possibilities, called Theoretical and Practical CBN in analogy to CBV and CBNeed, have the same quadratic/linear bounds, as we show.

For theoretical CBN, CBV, and CBNeed the results that we obtain are all based on the same reasoning and can be expressed with the same statement. Curiously, Theoretical CBV requires some care in the definition, as we show, because its naive definition induces a critical pair.

The practical cases are all linear but each one requires a different reasoning:

- *Practical CBV* (Sect. 3): the proof is extremely simple, but it is necessary to assume that evaluation terminates. The coefficient of the linear bound is 2, and we show it to be tight.

- *Practical CBNeed* (Sect. 5 and Sect. 6): we give two different proofs of the linear bound. A simple but lax one, providing 3 as the coefficient of the bound, and a more sophisticated one, relying on a simple labeling of the terms, that provides the optimal coefficient 2. None of the proofs assumes termination.

- *Practical CBN* (Sect. 7): the proof for CBN is similar to the one for Practical CBV, in particular it rests on a termination hypothesis. It is however less precise than for CBV, as it provides only 3 as coefficient. We do not know if this coefficient is tight, we conjecture not.

On the Relationship Between Micro-Step and Small-Step Strategies. We formulate all systems as strategies in the Linear Substitution Calculus, that is a λ-calculus with explicit substitutions. We do not show that the micro-step strategies implement the corresponding, more traditional small-step strategies without explicit substitutions. On the one hand because we do not want to blur the focus of the paper. On the other hand because these are routine proofs and can either be found in the literature or can be obtained by minimal variations:

- *Theoretical CBN*: the proof can be found in [9].

- *Practical CBN*: informal proof in Sect. 7.

- *Practical CBV*: it follows either by the fact that it is the micro-step strategy implemented by the CEK machine, as shown in [4], that in turn is known to implement also small-step CBV, or as a special case of study done in [6].

- *Theoretical CBV*: it can be obtained as a simple variation on the one for Theoretical CBN.

- *Practical CBNeed*: note that for CBNeed the question is different, because there are no small-step CBNeed strategies. As for CBV, in [4] Practical CBNeed is shown to the strategy implemented by standard CBNeed abstract machines.

- *Theoretical CBNeed*: the typical correctness theorem for CBNeed is that it is *observationally* equivalent to CBN. A slightly weaker form of correctness for Theoretical CBNeed is shown by Delia Kesner in [19], where it is proven to be *termination* equivalent to CBN.

The next section discusses the usual (*i.e.* Theoretical) CBN case, already well-studied in the literature. The rest of the paper will deal with the other combinations of values and strategies.

2. Theoretical Call-by-Name

Terms and Contexts. The language of the *linear substitution calculus*, shared by all the calculi treated in the paper, is generated by the following grammar:

$$t, u, w, r \quad ::= \quad x \mid \lambda x.t \mid tu \mid t[x \leftarrow u]$$

The constructor $t[x \leftarrow u]$ is called an *explicit substitution* (of u for x in t). Both $\lambda x.t$ and $t[x \leftarrow u]$ bind x in t, with the usual notion of α-equivalence and of free/bound variable (occurrence).

A *closed term* is a term without free variables. An *initial term* is a closed term with no explicit substitutions.

The operational semantics is defined using contexts, *i.e.* terms with one occurrence of the hole $\langle \cdot \rangle$, an additional constant. For CBN evaluation contexts are defined by the following grammar (where H is a mnemonic for *(weak) head*):

$$H ::= \langle \cdot \rangle \mid Ht \mid H[x \leftarrow t]$$

The *plugging* $H\langle t \rangle$ (resp. $H\langle H' \rangle$) of a term t (resp. context H') in a context H is defined as

$$\begin{array}{rcl} \langle t \rangle & := & t \\ (Ht)\langle u \rangle & := & H\langle u \rangle t \\ (H[x \leftarrow t])\langle u \rangle & := & H\langle u \rangle[x \leftarrow t] \end{array} \qquad \begin{array}{rcl} \langle H' \rangle & := & H' \\ (Ht)\langle H' \rangle & := & H\langle H' \rangle t \\ (H[x \leftarrow t])\langle H' \rangle & := & H\langle H' \rangle[x \leftarrow t] \end{array}$$

Plugging will be silently extended to all other notions of context at work in the paper (in the expected way).

Substitution contexts are defined by $L ::= \langle \cdot \rangle \mid L[x \leftarrow t]$.

Rewriting Rules. As usual, the rewriting rules are obtained by first defining the rewriting rules at top level, and then taking their closure by evaluation contexts. A peculiar aspect of the LSC is that contexts are however also used to define the rules at top level. Such a use of contexts is how locality on proof nets (the graphical language for linear logic proofs) is reflected on terms. For Theoretical CBN (a practical variant will be studied in Sect. 7), the rewriting relation is $\multimap_{\mathtt{TCBN}} := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$, where $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ are given by:

Rule at Top Level	Contextual Closure
$L\langle \lambda x.t\rangle u \mapsto_{\mathtt{m}} L\langle t[x\leftarrow u]\rangle$	$H\langle t\rangle \multimap_{\mathtt{m}} H\langle u\rangle$ iff $t \mapsto_{\mathtt{m}} u$
$H\langle x\rangle[x\leftarrow u] \mapsto_{\mathtt{e}} H\langle u\rangle[x\leftarrow u]$	$H\langle t\rangle \multimap_{\mathtt{e}} H\langle u\rangle$ iff $t \mapsto_{\mathtt{e}} u$

Examples:

$$((\lambda x.\lambda y.t)u)w \multimap_{\mathtt{m}} (\lambda y.t)[x\leftarrow u]w \multimap_{\mathtt{m}} t[y\leftarrow w][x\leftarrow u]$$

$$(xt)[y\leftarrow u][x\leftarrow z][z\leftarrow w] \multimap_{\mathtt{e}} (zt)[y\leftarrow u][x\leftarrow z][z\leftarrow w] \multimap_{\mathtt{e}} (wt)[y\leftarrow u][x\leftarrow z][z\leftarrow w]$$

We silently work modulo α-equivalence to avoid variable capture in the rewriting rules, and in $\mapsto_{\mathtt{e}}$ we assume that the context H does not capture the variable x nor the free variables of u. *Derivations* are possibly empty sequences of rewriting steps, noted d, d', d'', etc.

In the literature, $\multimap_{\mathtt{TCBN}}$ is known as *weak linear head reduction*, and it has been shown to be the strategy implemented by both the KAM [15, 4] and the π-calculus [3]. Rule $\multimap_{\mathtt{m}}$, turning (generalized) β-redexes into explicit substitutions, corresponds to the *multiplicative* case of cut-elimination in proof nets, while $\multimap_{\mathtt{e}}$, implementing substitution in micro steps, corresponds to the exponential case. This terminology comes from the linear logic interpretation of the LSC. At times, the literature employs an alternative terminology for which $\multimap_{\mathtt{m}}$ is noted $\rightarrow_{\mathtt{dB}}$ and called B *at a distance* (in some papers on explicit substitutions the B-rule is the variation over the β-rule where the redex introduces the explicit substitutions instead of doing the meta-level substitution) and $\multimap_{\mathtt{e}}$ is noted $\rightarrow_{\mathtt{1s}}$ and called *linear substitution*. Sometimes the LSC is presented with a further rule for garbage collection (or weakening, in the logical terminology), but not here.

Exponential vs Multiplicative Analysis. The LSC provides a simple framework to study the overhead of the substitution process. The overhead itself is given as the number of exponential steps with respect to the number of multiplicative

steps, that are identified with β-redexes. Remarkably, the substitution process is encapsulated in just one rule.

For CBN, the relationship between $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ is already well-known from the literature [9, 7]. Given a derivation d let us note $|d|_e$ and $|d|_m$ the number of exponential and multiplicative steps in d, respectively. Then:

Theorem 2.1 (Quadratic Bound for Theoretical CBN [9])**.** *Let $d : t \multimap^*_{\mathtt{TCBN}} u$ be a derivation from an initial term t. Then $|d|_e = O(|d|_m^2)$ (and so $|d| = O(|d|_m^2)$).*

The Proof, Abstractly. In [7] this result is generalized and its proof is axiomatized. It relies on the fact that CBN micro-step evaluation has the following property, relating the number of potential exponential steps to the number of previous multiplicative steps.

> *Local Boundedness Property*: a micro-step strategy $\multimap$ has the *local boundedness property* if given a derivation $d : t \multimap^* u$ followed by a sequence of $\mathtt{e}$-steps $u \multimap^k_{\mathtt{e}} w$ then $k = O(|d|_m)$.

For the sake of completeness, we show that a locally bounded strategy has a quadratic overhead. This abstract result (together with the proof of local boundedness [9, 7]) provides the proof of Theorem 2.1.

Theorem 2.2 (Local Bound $\Rightarrow$ Quadratic Bound)**.** *Let $\multimap$ be a strategy with the local boundedness property and $d : t \multimap^* u$ be a derivation from an initial term t. Then $|d|_e = O(|d|_m^2)$ (and so $|d| = O(|d|_m^2)$).*

Proof. Since both $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ terminate (it is a general property of the LSC [1]), d has the shape:

$$t = w_1 \multimap^{a_1}_{\mathtt{m}} r_1 \multimap^{b_1}_{\mathtt{e}} w_2 \multimap^{a_2}_{\mathtt{m}} r_2 \multimap^{b_2}_{\mathtt{e}} \dots w_k \multimap^{a_k}_{\mathtt{m}} r_k \multimap^{b_k}_{\mathtt{e}} u$$

By the local boundedness property, we obtain $b_i \leq \sum_{j=1}^{i} a_j$. Then:

$$|d|_e = \sum_{i=1}^{k} b_i \leq \sum_{i=1}^{k} \sum_{j=1}^{i} a_j$$

Note that $\sum_{j=1}^{i} a_j \leq \sum_{j=1}^{k} a_j = |d|_m$ and $k \leq |d|_m$. So

$$|d|_e \leq \sum_{i=1}^{k} \sum_{j=1}^{i} a_j \leq \sum_{i=1}^{k} |d|_m \leq |d|_m^2$$

Finally, $|d| = |d|_m + |d|_e \leq |d|_m + |d|_m^2 = O(|d|_m^2)$. □

Now let us turn to the local boundedness property. It holds in general for any strategy $\multimap$ having the two following syntactic properties (using the notation of the theorem):

> *Trace*: the number $|u|_{[\ ]}$ of explicit substitutions in u is exactly $|d|_m$.
> *Syntactic Boundedness*: the length of a sequence of $\multimap_{\mathtt{e}}$ steps from u is $\leq |u|_{[\ ]}$.

Their proofs for the CBN strategy $\multimap_{\mathtt{TCBN}}$ can be found in [9] or—in a more general form—in [7]. Actually, both the trace and the syntactic boundedness properties follow from the subterm property, also proved in [9]:

> *Subterm*: a strategy $\multimap$ has the *subterm property* if given a derivation $d : t \multimap^* u$ the terms duplicated along d are subterms of t (up to α-equivalence).

Finally, the subterm property is obtained as a corollary of a syntactic invariant of $\multimap_{\mathtt{TCBN}}$, peculiar to micro-step evaluation and owing its name to the linear logic representation of λ-calculus, where arguments and explicit substitutions are wrapped into !-boxes:

> *Box*: if $d : t \multimap^*_{\mathtt{TCBN}} u$ then every argument and every content of a substitution in u is a subterm of t (up to α-equivalence).

This schema will be followed by all theoretical strategies in the paper. We will prove the box, subterm, trace and syntactic boundedness invariants. Therefore, the local boundedness property will hold, implying the quadratic bound by Theorem 2.2.

The quadratic bound holds for *any* derivation (not necessarily to normal form) and is *tight*, as it is reached for instance by $\delta\delta$ (where $\delta = \lambda x.(xx)$). Evaluation of $\delta\delta$ starts as follows

$$
\begin{array}{lll}
\delta\delta & \multimap_{\mathtt{m}} & (x_1x_1)[x_1{\leftarrow}\delta] \qquad\qquad\qquad\qquad \multimap_{\mathtt{e}} \\
 & & (\delta x_1)[x_1{\leftarrow}\delta] \qquad\qquad\qquad\qquad \multimap_{\mathtt{m}} \\
 & & (x_2x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \qquad\qquad \multimap_{\mathtt{e}} \\
 & & (x_1x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \qquad\qquad \multimap_{\mathtt{e}} \\
 & & (\delta x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \qquad\qquad \multimap_{\mathtt{m}} \\
 & & (x_3x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \quad \multimap_{\mathtt{e}} \\
 & & (x_2x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \quad \multimap_{\mathtt{e}} \\
 & & (x_1x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \quad \multimap_{\mathtt{e}} \ldots
\end{array}
\tag{1}
$$

It is easily seen that the sequences of $\mathtt{e}$-steps are strictly increasing in length. After the n-th $\mathtt{m}$-step we have indeed n $\mathtt{e}$-steps:

$$\begin{array}{ll}
(x_n x_n)[x_n \leftarrow x_{n-1}] \dots [x_2 \leftarrow x_1][x_1 \leftarrow \delta] & \multimap_{\mathtt{e}} \\
(x_{n-1} x_n)[x_n \leftarrow x_{n-1}] \dots [x_2 \leftarrow x_1][x_1 \leftarrow \delta] & \multimap_{\mathtt{e}} \\
\dots & \\
(x_1 x_n)[x_n \leftarrow x_{n-1}] \dots [x_2 \leftarrow x_1][x_1 \leftarrow \delta] & \multimap_{\mathtt{e}} \\
(\delta x_n)[x_n \leftarrow x_{n-1}] \dots [x_2 \leftarrow x_1][x_1 \leftarrow \delta] & \multimap_{\mathtt{m}} \\
(x_{n+1} x_{n+1})[x_{n+1} \leftarrow x_n][x_n \leftarrow x_{n-1}] \dots [x_2 \leftarrow x_1][x_1 \leftarrow \delta] & \multimap_{\mathtt{e}} \quad \dots
\end{array} \tag{2}$$

In particular, such steps are all variable renaming but for the last of the sequence, that creates an $\mathtt{m}$-step, and generates a new sequence of $n + 1$ renamings, and so on. In other words, these malicious sequences meet the bound in the syntactic boundedness property.

Let us point out that the bound is reached also by some normalizing terms. For example, we can adapt $\delta\delta$ to run only for n iterations. Let I be the identity function, and let $[\![n]\!]$ be the n-th Scott's numeral [31], defined by $[\![0]\!] = \lambda y.\lambda z.y$ and $[\![n + 1]\!] = \lambda y.\lambda z.z[\![n]\!]$. In ordinary λ-calculus, $[\![n]\!]tf$ reduces to $f(f(\dots(t)))$ where f is applied n times. The modified $\delta\delta$ we are looking for is $\tau\tau[\![n]\!]$ where $\tau = \lambda x.\lambda n.(nI(xx))$ that, in ordinary λ-calculus, reduces as follows:

$$\tau\tau[\![n]\!] \Rightarrow [\![n]\!]I(\tau\tau) \Rightarrow \tau\tau[\![n-1]\!] \Rightarrow \dots \Rightarrow [\![0]\!]I(\tau\tau) \Rightarrow I$$

Evaluating the term in Theoretical CBN takes $(n + 1)(n + 4)/2$ exponential steps but only $4(n + 1)$ multiplicative steps. Such a formula has been tested by running an implementation of Theoretical CBN.

3. Practical Call-by-Value

We present first Practical CBV, because its definition is standard. Theoretical CBV instead requires a slight technical adjustment (of which we were not aware in the conference paper [10]), and it is then postponed to Sect. 4.

Left-to-Right CBV. The underlying language is the same as for CBN, but we distinguish (practical) *values*, noted v, that are given by abstractions only, and *answers* $L\langle v\rangle$, given by a value in a substitution context (see Sect. 2 for the definition of substitution contexts). Evaluation contexts for CBV, implementing left-to-right CBV, are defined by:

$$E ::= \langle\cdot\rangle \mid Et \mid L\langle v\rangle E \mid E[x \leftarrow t]$$

Rewriting Rules. For Practical CBV the rewriting relation is $\multimap_{\mathtt{PCBV}} := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$, where $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ are re-defined as follows:

Rule at Top Level	Contextual closure
$L\langle\lambda x.t\rangle L'\langle v\rangle \mapsto_{\mathtt{m}} L\langle t[x{\leftarrow}L'\langle v\rangle]\rangle$	$E\langle t\rangle \multimap_{\mathtt{m}} E\langle u\rangle$ iff $t \mapsto_{\mathtt{m}} u$
$E\langle x\rangle[x{\leftarrow}L\langle v\rangle] \mapsto_{\mathtt{e}} L\langle E\langle v\rangle[x{\leftarrow}v]\rangle$	$E\langle t\rangle \multimap_{\mathtt{e}} E\langle u\rangle$ iff $t \mapsto_{\mathtt{e}} u$

As for CBN, we silently work modulo α-equivalence and in $\mapsto_{\mathtt{e}}$ the context E does not capture x nor the free variables of v.

Let us revisit the $\delta\delta$ example of Sect. 2, used to show that the quadratic bound is tight for CBN. Practical values give:

$$
\begin{array}{lll}
\delta\delta \multimap_{\mathtt{m}} & (x_1x_1)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & (\delta x_1)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & (\delta\delta)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} \\
 & (x_2x_2)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & (\delta x_2)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & (\delta\delta)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} \\
 & (x_3x_3)[x_3{\leftarrow}\delta][x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \ldots
\end{array}
\tag{3}
$$

Where it is easily seen that for any $d : \delta\delta \multimap^* t$ we have the linear relationship $|d|_e \leq 2|d|_m$. This fact suggests that any CBV derivation d verifies $|d|_e = O(|d|_m)$. Curiously, it turns out that this is not true for *any* derivation, as forthcoming Lemma 3.2 will show, but only for those reaching a normal form.

Exponential vs Multiplicative Analysis. We first need some invariants of Practical CBV. With respect to Theoretical CBN the box invariant is replaced by the value invariant but it plays exactly the same role. The proper invariant is a new simple property peculiar to CBV, while the syntactic boundedness property does not hold, as we discuss after the lemma.

Lemma 3.1 (Practical CBV Invariants)**.** *Let t be initial and $d : t \multimap^*_{\mathtt{PCBV}} u$.*

1. Value*: every value in u is a value in t (up to α);*
2. Subterm*: the terms duplicated along d are subterms of t (up to α);*
3. Proper*: every substitution in u contains an answer;*
4. Trace*: the number $|u|_{[\ ]}$ of explicit substitutions in u is exactly $|d|_m$.*

Proof. Easy inductions on the length of d. Point 1 is used to prove Point 2 ($\multimap_{\mathtt{PCBV}}$ only duplicates values), in turn used to prove Point 3 and Point 4. □

Somewhat surprisingly, in CBV the local boundedness property does not hold. Let us show it. Let t^n stand for t applied to itself n times, associating to the right, *i.e.* $t^n := t(t(t(t\ldots)))$ n times, and set $I := \lambda y.y$. We have

Lemma 3.2 (No Local Boundedness for $\multimap_{\mathtt{PCBV}}$)**.** *The derivation* $(\lambda x.x^n)I \multimap_{\mathtt{m}} x^n[x{\leftarrow}I] \multimap_{\mathtt{e}}^n I^n[x{\leftarrow}I]$ *is a counter-example to both the local and syntactic boundedness properties*[1]*.*

It seems even worse than in CBN, while instead, *globally*, it is a faster mechanism, of a different nature. Note, for instance, that the $\mathtt{e}$-steps in the counter-example are independent, *i.e.* they are not generated by malicious chains of substitutions as in CBN. Note also that if evaluation keeps going, the term $I^n[x{\leftarrow}I]$ needs n multiplicative steps to reach its normal form. This suggests that observing evaluations to normal form one can find a linear *global bound*. We will show that the gap between $|d|_e$ and $|d|_m$ is linearly bounded by the number of values in the end term, and this will give us the bound when the end term is a value, *i.e.* a normal form.

Let us provide an intuition for the forthcoming proof of the linear bound. An exponential step makes a *new copy* of a value. A multiplicative step *consumes* the value in its left subterm. Therefore it is possible to bound the number of $\mathtt{e}$-steps with the number of values in the term plus the number of those already consumed (that is the number of multiplicative steps). To be formal, let us introduce the *CBV size* of the term.

Definition 3.3 (CBV Size)**.** *The* CBV size $|\cdot|_{CBV}$ *of a term counts the number of values that are not inside another value. It is defined recursively as follows:*

$$\begin{array}{rcl} |x|_{CBV} & := & 0 \\ |v|_{CBV} & := & 1 \\ |tu|_{CBV} & := & |t|_{CBV} + |u|_{CBV} \\ |t[x{\leftarrow}u]|_{CBV} & := & |t|_{CBV} + |u|_{CBV} \end{array}$$

In just one surprisingly simple lemma we obtain the main invariant relating $\multimap_{\mathtt{e}}$, $\multimap_{\mathtt{m}}$, and the CBV size. The corollary uses the previous invariants to instantiate it in the terminating case, obtaining the linear bound.

[1] In another work (done during the preparation of the final version of this paper) we proved that there is a bound on the exponentials if one takes into account also the size of the initial term. Precisely, the number of $\multimap_{\mathtt{e}}$ steps is linear in the number of $\multimap_{\mathtt{m}}$ steps and the size of the initial term. The result is contained in the submitted paper [8], formulated for abstract machines, but it holds for the LSC too, *mutatis mutandis*.

Lemma 3.4 (Main Practical CBV Invariant)**.** *Let* $d : t \multimap^{n}_{\mathtt{PCBV}} u$*. Then* $|d|_e \leq |d|_m + |u|_{CBV}$*.*

In the conference version of this paper [10] we actually proved a stronger invariant, namely $|d|_e \leq |d|_m + |u|_{CBV} - |t|_{CBV}$. Here we simplified it because such a stronger form is not exploited in the proof of the global linear bound. The stronger invariant would actually allow to remove the +1 in the linear bound given by forthcoming Corollary 3.5, providing the same bound of CBNeed (see Theorem 6.4). Since the gain is minimal, we prefer to present the lighter invariant.

Proof. By induction on n. Case $n = 0$ is obvious. Otherwise $d' : t \multimap^{n-1}_{\mathtt{PCBV}} w$ and d extends d' with $w \multimap_{\mathtt{PCBV}} u$. By *i.h.*, $|d'|_e \leq |d'|_m + |w|_{CBV}$. Cases:

- *the last step is exponential.* Then

$$w = E\langle E'\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle \multimap_{\mathtt{e}} E\langle L\langle E'\langle v\rangle[x{\leftarrow}v]\rangle\rangle = u$$

  and $|u|_{CBV} = |w|_{CBV} + 1$. Thus

$$|d|_e \;=\; |d'|_e + 1 \;\leq_{i.h.}\; |d'|_m + |w|_{CBV} + 1 = |d|_m + |w|_{CBV} + 1 = |d|_m + |u|_{CBV}$$

- *the last step is multiplicative*. Then

$$w = E\langle L\langle \lambda x.r\rangle L'\langle v\rangle\rangle \multimap_{\mathtt{m}} E\langle L\langle r[x{\leftarrow}L'\langle v\rangle]\rangle\rangle = u$$

  and $|u|_{CBV} = |w|_{CBV} - 1 + |r|_{CBV}$, so that $|w|_{CBV} = |u|_{CBV} + 1 - |r|_{CBV}$. Note also that $|d|_m = |d'|_m + 1$. Thus

$$\begin{array}{rcll} |d|_e & = & |d'|_e & \leq_{i.h.} \; |d'|_m + |w|_{CBV} \\ & & & = \; |d|_m - 1 + |w|_{CBV} \\ & & & = \; |d|_m - 1 + |u|_{CBV} + 1 - |r|_{CBV} \\ & & & = \; |d|_m + |u|_{CBV} - |r|_{CBV} \qquad \leq \; |d|_m + |u|_{CBV} \end{array}$$

□

Corollary 3.5 (Linear Bound for Practical CBV)**.** *Let t be initial and* $d : t \multimap^{*}_{\mathtt{PCBV}} L\langle v\rangle$*. Then* $|d|_e \leq 2|d|_m + 1$*.*

Proof. By the proper invariant (Lemma 3.1.3) every substitution contains a value plus some substitutions, each one recursively having the same shape, so $|L\langle v\rangle|_{CBV} = |L\langle v\rangle|_{[\ ]} + 1$, where 1 accounts for the value v. By the trace invariant (Lemma 3.1.4) $|L\langle v\rangle|_{[\ ]} = |d|_m$, and so $|L\langle v\rangle|_{CBV} \leq |d|_m + 1$. Then the main invariant (Lemma 3.4) gives: $|d|_e \leq_{L.3.4} |d|_m + |L\langle v\rangle|_{CBV} \leq |d|_m + |d|_m + 1 = 2|d|_m + 1$. □

Right-to-Left CBV. In this section we studied left-to-right CBV. The dual right-to-left strategy can be obtained modularly by simply redefining the grammar of evaluation context as

$$E ::= \langle\cdot\rangle \mid EL\langle v\rangle \mid tE \mid E[x{\leftarrow}t]$$

and by using this new notion also at top level in the definition of the exponential rule. Our proof for the bound with practical values holds unchanged also for the right-to-left strategy. Note indeed that our proof does not rely on the fact that the strategy is left-to-right. We use the fact that evaluation is *weak*, it involves *proper terms*, and it has the *trace property*. All these facts are easily seen to hold for the right-to-left strategy as well, for which the linear bound then follows.

4. Theoretical Call-by-Value

In this section we show that Theoretical CBV has a quadratic overhead, for both left-to-right and right-to-left strategies. We postponed the study of these cases because of a subtlety, that requires a small change to the exponential rule. In the conference version of this paper [10], we did not yet fully understand the issue.

The technical point is that switching from practical to theoretical values—leaving everything else unchanged—introduces a critical pair. The pair is benign, as it does not impact on the result of evaluation. Since programming languages are modeled by deterministic strategies, however, one of the two paths of the diagram has to be fixed, and the choice has an impact on the substitution overhead. Both right-to-left and left-to-right naïve Theoretical CBV admit the following critical pair:

$$E\langle t[x{\leftarrow}y]\rangle[y{\leftarrow}v] \quad {}_{\mathtt{m}}\!\!\multimap \quad E\langle(\lambda x.t)y\rangle[y{\leftarrow}v] \quad \multimap_{\mathtt{e}} \quad E\langle(\lambda x.t)v\rangle[y{\leftarrow}v]$$

The natural solution is to give precedence to the multiplicative step, as it takes place in a outer evaluation context. Additionally, remark that giving precedence to the exponential redex induces exactly Practical CBV, and such a choice has already been studied. We believe that such a critical pair is (one of) the hidden reason for the discrepancy between the theoretical and practical use of values in the literature.

In order to solve the pair in favor of the multiplicative step the exponential rule has to be refined, asking that only variable occurrences in applicative contexts are replaced.

Definition 4.1 (Applicative Context)**.** *An evaluation context E is* applicative *if $E = E'\langle Lt\rangle$, i.e. if its hole is applied to an argument, possibly with some substitutions in between (given by L).*

Now we define $\multimap_{\mathtt{TCBV}} := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$ where $\multimap_{\mathtt{m}}$ is as for practical CBV and $\multimap_{\mathtt{e}}$ is re-defined as follows:

$$\begin{array}{c}\textsc{Rule at Top Level}\\ E\langle x\rangle[x{\leftarrow}L\langle v\rangle] \mapsto_{\mathtt{e}} L\langle E\langle v\rangle[x{\leftarrow}v]\rangle \qquad \text{if } E \text{ is applicative}\end{array}$$

$$\begin{array}{c}\textsc{Contextual Closure}\\ E\langle t\rangle \multimap_{\mathtt{e}} E\langle u\rangle \quad \text{if } t \mapsto_{\mathtt{e}} u\end{array}$$

Note that such a definition is not ad-hoc, as it matches CBV weak linear head reduction as in [3], where it is defined by mimicking evaluation in the π-calculus (according to a CBV translation). It is easily seen that $\multimap_{\mathtt{TCBV}}$ is deterministic. Moreover, an answer now is a *theoretical* value in a substitution context.

The typical quadratic example for Theoretical CBN, given by $\delta\delta$, evaluates exactly in the same way with Theoretical CBV, for both left-to-right and right-to-left strategies:

$$\begin{array}{lll}\delta\delta & \multimap_{\mathtt{m}} & (x_1x_1)[x_1{\leftarrow}\delta] \qquad\qquad\qquad \multimap_{\mathtt{e}}\\ & & (\delta x_1)[x_1{\leftarrow}\delta] \qquad\qquad\qquad \multimap_{\mathtt{m}}\\ & & (x_2x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \qquad \multimap_{\mathtt{e}}\\ & & (x_1x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \qquad \multimap_{\mathtt{e}}\\ & & (\delta x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \qquad \multimap_{\mathtt{m}}\\ & & (x_3x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \quad \multimap_{\mathtt{e}}\\ & & (x_2x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \quad \multimap_{\mathtt{e}}\\ & & (x_1x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] \quad \multimap_{\mathtt{e}} \ldots\end{array} \tag{4}$$

In contrast to the practical case, the theoretical exponential rule has the syntactic boundedness property, and thus the abstract reasoning for Theoretical CBN can be applied unchanged (to both left-to-right and right-to-left strategies), obtaining a quadratic overhead.

Lemma 4.2 (Theoretical CBV Invariants)**.** *Let t be initial and $d : t \multimap^*_{\mathtt{TCBV}} u$.*

1. Value*: every value in u is a value in t (up to α);*
2. Subterm*: the terms duplicated along d are subterms of t (up to α);*
3. Proper*: every substitution in u contains an answer;*

4. Trace*: the number $|u|_{[\ ]}$ of explicit substitutions in u is exactly $|d|_m$;*
5. Syntactic Boundedness*: the length of a sequence of $\multimap_{\mathsf{e}}$ steps from u is* $\leq |u|_{[\ ]}$.

Proof. Points 1-4 are as for the practical case (*i.e.* they are easy inductions on the length of d, as in Lemma 3.1). Point 5 follows from the fact that a sequence of exponential steps can only keep renaming on the same place until an abstraction is substituted and thus a multiplicative redex is created (because the context is now required to be applicative). For scoping reasons, the substitutions involved in the sequence are all different and appearing from left to right, *i.e.* the number of e-steps is $\leq |u|_{[\ ]}$. □

Theorem 4.3 (Quadratic Bound for Theoretical CBV)**.** *Let $d : t \multimap^*_{\mathtt{TCBV}} u$ be a Theoretical CBV derivation from an initial term t. Then $|d|_e = O(|d|_m^2)$ (and so $|d| = O(|d|_m^2)$).*

Proof. The trace and syntactic boundedness invariants of Lemma 4.2, together, give the local boundedness property, in turn implying the quadratic bound by Theorem 2.2. □

5. Theoretical and Practical Call-by-Need

For call-by-need (CBNeed), the analysis is, again, different. At first sight, CBNeed is very similar to CBN, as it has the local boundedness property. CBNeed, however, has also the flavor of CBV. While in CBN *any* substitution sequence can have length $|d|_m$, in CBNeed it is *the concatenation of all chains* that is bound by (twice) $|d|_m$. As for CBV, there is a *matching*, or *consumption* phenomenon: firing a substitution chain of length k *consumes* k preceding multiplicative steps, decreasing the bound *for the chains to come* (note that in CBV multiplicative steps consume exponential steps, while here it is the other way around).

We will provide two proofs of the linear bound. A first easy one, and a second one providing a tighter bound, at the price of some technicalities. Both proofs lead to the bound by distinguishing between different forms of substitutions, but in different ways. The first proof focuses on the occurrences on which substitutions act, distinguishing those that give rise to a $\multimap_{\mathtt{m}}$ redex, called *applicative*, and those that do not, called *inert*. The second proof distinguishes between substitutions that have already substituted somewhere, labeled with a *black* dot, and those that did not act yet, labeled with a *white* circle.

This section deals with Theoretical CBNeed and with the first proof for Practical CBNeed. The definition of the labeled system and the second proof for Practical CBNeed are instead treated in Sect. 6.

The CBNeed Calculus. Terms, values, and answers are defined as before. CBNeed evaluation contexts are defined by:

$$N ::= \langle\cdot\rangle \mid Nt \mid N[x{\leftarrow}t] \mid N'\langle x\rangle[x{\leftarrow}N]$$

Note that CBNeed evaluation contexts extend the weak head contexts for call-by-name with a clause ($N'\langle x\rangle[x{\leftarrow}N]$) that turns them into *hereditary weak head contexts*. This new clause is how sharing is implemented by the strategy.

A pleasant fact about CBNeed is that the theoretical and practical variant differ only in the definition of value, as the rewriting rules can be kept unchanged. Then we define value-agnostic rewriting relation $\multimap_{\mathtt{Need}} := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$ as:

Rule at Top Level	Contextual closure
$L\langle\lambda x.t\rangle u \mapsto_{\mathtt{m}} L\langle t[x{\leftarrow}u]\rangle$	$N\langle t\rangle \multimap_{\mathtt{m}} N\langle u\rangle$ iff $t \mapsto_{\mathtt{m}} u$
$N\langle x\rangle[x{\leftarrow}L\langle v\rangle] \mapsto_{\mathtt{e}} L\langle N\langle v\rangle[x{\leftarrow}v]\rangle$	$N\langle t\rangle \multimap_{\mathtt{e}} N\langle u\rangle$ iff $t \mapsto_{\mathtt{e}} u$

And then use $\multimap_{\mathtt{TNeed}}$ and $\multimap_{\mathtt{PNeed}}$ for the strategies obtained by instantiating $\multimap_{\mathtt{Need}}$ with theoretical and practical values, respectively. Note that the multiplicative rule is taken from the CBN calculus. Therefore the definiens of a substitution is not necessarily an answer. The exponential rule comes instead from the CBV calculus, and requires arguments to be evaluated to answers before being substituted, reflecting the *by need* content of the strategy. Such a simple presentation of CBNeed is adopted also in [4].

Theoretical CBNeed. Both Theoretical and Practical CBNeed satisfy the same invariants of CBN, but for the box invariant that is replaced by the value invariant, as in CBV. We then state the invariants lemma in a value-agnostic way.

Lemma 5.1 (CBNeed Invariants)**.** *Let t be initial and $d : t \multimap^*_{\mathtt{Need}} u$.*

1. Value*: every value in u is a value in t (up to α);*
2. Subterm*: the terms duplicated along d are subterms of t (up to α);*
3. Trace*: the number $|u|_{[\,]}$ of explicit substitutions in u is exactly $|d|_m$;*
4. Syntactic Boundedness*: the length of a sequence of $\multimap_{\mathtt{e}}$ steps from u is $\leq |u|_{[\,]}$.*

Exactly as for the other theoretical cases, the quadratic bound follows abstractly from the invariants.

Theorem 5.2 (Quadratic Bound for Theoretical CBNeed)**.** *Let $d : t \multimap^*_{\texttt{TNeed}} u$ be a derivation from an initial term t. Then $|d|_e = O(|d|_m^2)$ (and so $|d| = O(|d|_m^2)$).*

Proof. The trace and syntactic boundedness properties of Lemma 5.1, together, give the local boundedness property, in turn implying the quadratic bound by Theorem 2.2. □

Finally, the quadratic bound for $\multimap_{\texttt{TNeed}}$ is tight as it is reached by $\delta\delta$, that evaluates exactly as in CBN (see Sect. 2).

Practical CBNeed. The practical strategy $\multimap_{\texttt{PNeed}}$ evaluates $\delta\delta$ as follows:

$$
\begin{array}{llll}
\delta\delta & \multimap_{\mathtt{m}} & (x_1x_1)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & & (\delta x_1)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} \\
 & & (x_2x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & & (x_2x_2)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & & (\delta x_2)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} \\
 & & (x_3x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & & (x_3x_3)[x_3{\leftarrow}\delta][x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} \\
 & & (\delta x_3)[x_3{\leftarrow}\delta][x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} \quad \dots
\end{array}
\tag{5}
$$

Where it is easily seen that for any $d : \delta\delta \multimap^*_{\texttt{PNeed}} t$ we have $|d|_e \leq 2|d|_m$. We are going to show that—in contrast to CBV—this bound holds for *any* CBNeed derivation, *i.e.* the derivation does not need to end on a normal form.

A First Easy Proof of the Linear Bound for Practical CBNeed. The idea is to distinguish between variable occurrences whose replacement creates a $\multimap_{\mathtt{m}}$ redex, called applicative, and those that do not, called inert.

Applicative (A) and *inert* (I) *evaluation contexts* are defined by:

$$
\begin{array}{rcl}
A & ::= & Lt \mid N\langle A\rangle; \\
I & ::= & L \mid N\langle N'\langle x\rangle[x{\leftarrow}I]\rangle
\end{array}
$$

Lemma 5.3. *Let N be a CBNeed evaluation context. Then either N is applicative or it is inert.*

Proof. By induction on N. Cases:

1. *Empty*, *i.e.* $N = \langle\cdot\rangle$. Then N is inert.

2. *Left Application*, *i.e.* $N = N't$. By *i.h.*, N' is either applicative or inert. If it is applicative then N is applicative (and not inert). If N' is inert and has the form L then $N = Lt$ is applicative and not inert, otherwise N is inert and not applicative.
3. *Left of a Substitution*, *i.e.* $N = N'[x{\leftarrow}t]$. It follows from the *i.h.*, as the addition of an explicit substitution cannot change the applicative/inert nature of a context.
4. *Right of a Substitution*, *i.e.* $N = N'\langle x\rangle[x{\leftarrow}N'']$. Note that N is applicative/inert iff N'' is. Then it follows from the *i.h.*.

□

A substitution step, that writes explicitly as

$$N'\langle N\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle \multimap_{\mathtt{e}} N'\langle L\langle N\langle v\rangle[x{\leftarrow}v]\rangle\rangle$$

is *applicative* or *inert* depending on the nature of the context of the substituted occurrence of x, *i.e.* on the nature of $N'\langle N\langle\cdot\rangle[x{\leftarrow}L\langle v\rangle]\rangle$.

The number of consecutive inert exponential steps can easily be bounded. By the syntactic boundedness property (Lemma 5.1.4) we already know that it is bounded by the number of substitutions in the term, but the bound can be improved.

Let a substitution $t[x{\leftarrow}u]$ be *basic* if u has the form $L\langle y\rangle$. The *basic size* $|t|_{\mathtt{b}}$ of a term t is the number of its substitutions that are basic. We are about to prove refinements of the trace and syntactic properties, where the role of the number of substitutions in the term is replaced by the role of basic substitutions.

For the trace property note that basic substitutions are created by multiplicative steps, never duplicated, and consumed by inert steps. Formally,

Lemma 5.4 (Basic Trace)**.** *Let t be initial and $d : t \multimap^*_{\mathtt{PNeed}} u$. Then $|u|_{\mathtt{b}} \leq 2|d|_m - |d|_{\mathtt{ei}}$.*

Intuition tells that a multiplicative step creates at most one basic substitution, which is the intended semantics. Multiplicative steps indeed create a basic substitution whenever the argument has the form $L'\langle y\rangle$, *e.g.* as in

$$N\langle L\langle\lambda x.t\rangle L'\langle y\rangle\rangle \multimap_{\mathtt{m}} N\langle L\langle t[x{\leftarrow}L'\langle y\rangle]\rangle\rangle$$

Unfortunately, they may create *two* basic substitutions at once, which is the reason why the lemma is forced to count 2 for every multiplicative step. Basic substitutions, in fact, can also be obtained as follows

$$N\langle t[x{\leftarrow}(\lambda y.y)u]\rangle \multimap_{\mathtt{m}} N\langle t[x{\leftarrow}y[y{\leftarrow}u]]\rangle$$

i.e. a multiplicative step turns the substitution in which it takes place, *i.e.* $[x \leftarrow (\lambda y.y)u]$ into a basic one, beyond possibly introducing a new basic substitution with $[y \leftarrow u]$. These accidents are responsible for the lax bound of this section. In the next section we will provide a different, finer analysis, where every multiplicative step will count for 1.

Proof. By induction on the length k of d. If $k = 0$ the statement holds. Then consider $d' : t \multimap^{k-1}_{\mathtt{PNeed}} w$, for which the *i.h.* provides $|w|_{\mathtt{b}} \leq 2|d'|_m - |d'|_{\mathtt{ei}}$. Consider the additional step $w \multimap_{\mathtt{PNeed}} u$. Cases:

1. *Multiplicative step.* As remarked right before the proof we have $|u|_{\mathtt{b}} \leq |w|_{\mathtt{b}} + 2$. Since $|d|_m = |d'|_m + 1$ and $|d|_e = |d'|_e$, the statement holds:

$$|u|_{\mathtt{b}} \leq |w|_{\mathtt{b}} + 2 \leq_{i.h.} 2|d'|_m - |d'|_{\mathtt{ei}} + 2 = 2(|d|_m - 1) - |d'|_{\mathtt{ei}} + 2 = 2|d|_m - |d|_{\mathtt{ei}}$$

2. *Applicative exponential.* Then none of the involved quantities changes.
3. *Inert exponential.* Then $|u|_{\mathtt{b}} = |w|_{\mathtt{b}} - 1$, $|d|_m = |d'|_m$ and $|d|_e = |d'|_e + 1$, and the statement holds.

□

The next lemma is the refinement of the syntactic boundedness property, and it bounds the number of consecutive inert steps. Actually, it shows a bit more, as it also says something about applicative steps.

Lemma 5.5 (Almost Inert Chains, or Basic Syntactic Boundedness)**.** *If $t \multimap^k_{\mathtt{e}} u$ then $k \leq |t|_{\mathtt{b}} + 1$ and all steps except possibly the last one are inert.*

Proof. If $k = 1$ the statement holds by Lemma 5.3. Then let $k > 1$, so that $t \multimap_{\mathtt{e}} w \multimap^{k-1}_{\mathtt{e}} u$, and consider the step $t \multimap_{\mathtt{e}} w$. It cannot be applicative, otherwise the next step would be multiplicative. So it is inert (Lemma 5.3). Let I be the inert context around the substituted variable. Two cases

1. *I is a list of substitutions L.* Then

$$t = L\langle x\rangle[x \leftarrow L'\langle v\rangle] \multimap_{\mathtt{e}} L'\langle L\langle v\rangle[x \leftarrow v]\rangle = w$$

and w is normal, contradicting the hypothesis $k > 1$.

2. *I has the hole in a basic substitution* $I = N\langle N'\langle x\rangle[x{\leftarrow}L]\rangle$. By *i.h.*, $k-1 \leq |w|_{\mathtt{b}}+1$ and all steps of $w \multimap_{\mathtt{e}}^{k-1} u$ except possibly the last one are inert. Note that the step $t \multimap_{\mathtt{e}} w$ is inert and—by the subterm property (Lemma 5.1.2) and because values are practical—it turns exactly one basic substitution in t into a non-basic one in w, leaving the others unchanged. Therefore $|t|_{\mathtt{b}} = |w|_{\mathtt{b}} + 1$. Then $k \leq |w|_{\mathtt{b}} + 2 = |t|_{\mathtt{b}} + 1$.

□

Theorem 5.6 (Linear Bound for Practical CBNeed)**.** *Let t be initial and d : $t \multimap^*_{\mathtt{PNeed}} u$. Then $|d|_e \leq 3|d|_m + 1$.*

Proof. Given that both $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ terminate (for $\multimap_{\mathtt{m}}$ it is evident, for $\multimap_{\mathtt{e}}$ it follows from the (basic) syntactic boundedness property, *i.e.* Lemma 5.1.4 or Lemma 5.5), d writes uniquely as:

$$t = t_1 \multimap_{\mathtt{m}}^{a_1} w_1 \multimap_{\mathtt{e}}^{b_1} t_2 \ldots t_k \multimap_{\mathtt{m}}^{a_k} w_k \multimap_{\mathtt{e}}^{b_k} u$$

Note that, since chains are almost inert (Lemma 5.5), in b_i there is at most one applicative substitution step. Let $d_i : t \multimap^*_{\mathtt{PNeed}} w_i$ be the prefix of d ending on w_i (including a_j and b_j for $j < i$, plus a_i, but not b_i). Defining $b_0 := 0$ we obtain $|d_i|_{\mathtt{ei}} \geq \sum_{j=0}^{i-1}(b_j - 1)$. The basic trace property (Lemma 5.4) then gives:

$$b_i \leq |w_i|_{\mathtt{b}} + 1 \leq_{L.5.4} 2|d_i|_m - |d_i|_{\mathtt{ei}} + 1 \leq 2|d_i|_m - \sum_{j=1}^{i-1}(b_j - 1) + 1$$

We conclude with the following chain, where the last step is given by observing that $k \leq |d|_m$:

$$\begin{array}{rcccll} |d|_e & = & \sum_{i=0}^{k} b_i & \leq & b_k + \sum_{i=0}^{k-1} b_i & \leq \\ & & & & 2|d_k|_m - \sum_{j=0}^{k-1}(b_j - 1) + 1 + \sum_{i=0}^{k-1} b_i & = \\ & & & & 2|d|_m - \sum_{j=0}^{k-1}(-1) + 1 & = \\ & & & & 2|d|_m + k + 1 & \leq \quad 3|d|_m + 1 \end{array}$$

□

6. The Exact Bound for Practical Call-by-Need, via Labels

The analysis of the previous section relies on a *syntactic* invariant, providing 3 as constant for the linear factor. As shown before, however, the evaluation of $\delta\delta$—*i.e.* the typical worst case—suggests the constant rather to be 2. Here we obtain

such an improved bound. The analysis is based on a different *dynamic* invariant, captured syntactically via an elementary notion of labeled substitution. It works according to the following schema:

1. *White Substitutions*: every multiplicative step produces a new substitution, distinguished by being *white* $t[x{\leftarrow}u]^\circ$;
2. *Black Substitutions*: the first time a white substitution $[x{\leftarrow}u]^\circ$ is used, replacing x somewhere, it changes, becoming *black* $[x{\leftarrow}u]^\bullet$;
3. *Almost White Chains*: a chain of substitution steps can contain at most one black step, which is the first one, if any (note the difference with the previous section, where almost basic chains can have at most one applicative step, but at then end, not at the beginning);
4. *Bound on Black Steps*: thus black steps are bound by the number of alternations between exponential and multiplicative evaluation, *i.e.* are linear in $|d|_m$;
5. *Bound on White Steps*: white substitutions are created by multiplicative steps, never duplicated, and consumed on first use, so the total number of white steps is bound by $|d|_m$;
6. *Bound on Exponential Steps*: therefore all substitution steps together are bound by $2|d|_m$.

This approach can be seen as a simple form of amortized analysis.

The Labeled Calculus. The labeled language is given by:

$$\begin{array}{rcl} t, u, w, r & ::= & x \mid v \mid tu \mid t[x{\leftarrow}u]^\circ \mid t[x{\leftarrow}u]^\bullet \\ v & ::= & \lambda x.t \end{array}$$

A *white* substitution $t[x{\leftarrow}u]^\circ$ represents a substitution that has never substituted its content yet. A *black* substitution $t[x{\leftarrow}u]^\bullet$ instead is an already evaluated substitution, *i.e.* one that has already acted on some variable occurrence. An invariant of evaluation will be that black substitutions contain values. We use $t[x{\leftarrow}u]^\circ_\bullet$ for $t[x{\leftarrow}u]^\circ$ or $t[x{\leftarrow}u]^\bullet$. Of course, we need to redefine also substitution and evaluation contexts, duplicating the cases for substitution:

$$\begin{array}{rcl} L & ::= & \langle\cdot\rangle \mid L[x{\leftarrow}t]^\circ \mid L[x{\leftarrow}t]^\bullet; \\ N, M & ::= & \langle\cdot\rangle \mid Nt \mid N[x{\leftarrow}t]^\circ \mid N[x{\leftarrow}t]^\bullet \mid N\langle x\rangle[x{\leftarrow}N]^\circ \mid N\langle x\rangle[x{\leftarrow}N]^\bullet. \end{array}$$

According to the informal semantics, the rewriting rules are:

Rule at Top Level	Contextual closure
$L\langle\lambda x.t\rangle u \mapsto_{\mathtt{m}} L\langle t[x{\leftarrow}u]^{\circ}\rangle$	$N\langle t\rangle \multimap_{\mathtt{m}} N\langle u\rangle$ iff $t \mapsto_{\mathtt{m}} u$
$N\langle x\rangle[x{\leftarrow}L\langle v\rangle]^{\circ} \mapsto_{\mathtt{e}\circ} L\langle N\langle v\rangle[x{\leftarrow}v]^{\bullet}\rangle$	$N\langle t\rangle \to_{\mathtt{e}\circ} N\langle u\rangle$ iff $t \mapsto_{\mathtt{e}\circ} u$
$N\langle x\rangle[x{\leftarrow}L\langle v\rangle]^{\bullet} \mapsto_{\mathtt{e}\bullet} L\langle N\langle v\rangle[x{\leftarrow}v]^{\bullet}\rangle$	$N\langle t\rangle \to_{\mathtt{e}\bullet} N\langle u\rangle$ iff $t \mapsto_{\mathtt{e}\bullet} u$

The rewriting relation is $\multimap_{\mathtt{PNeed}^{\circ}_{\bullet}} := \multimap_{\mathtt{m}} \cup \to_{\mathtt{e}\circ} \cup \to_{\mathtt{e}\bullet}$. Let $\to_{\mathtt{e}^{\circ}_{\bullet}}$ stay for $\to_{\mathtt{e}\circ}$ or $\to_{\mathtt{e}\bullet}$.

Note that a used substitution—*i.e.* a black one—has to contain a (practical) value, while on white substitutions there is no constraint. Clearly, there can be terms that are ill-labeled, *e.g.* $t[x{\leftarrow}y]^{\bullet}[y{\leftarrow}u]^{\circ}$, because y is not a practical value. Then we need a notion of well-labeled term. A term is *black-proper* if every black substitution contains a practical value. As expected, black-properness is stable under evaluation. We will also need the subterm property, proved exactly as for the unlabeled case of the previous section (but now concerning the syntax with labels). We restate it for the sake of completeness.

Lemma 6.1 (Labeled CBNeed Invariants)**.** *Let t be a λ-term and $d : t \multimap^{*}_{\mathtt{PNeed}^{\circ}_{\bullet}} u$.*

1. Subterm*: the terms duplicated along d are subterms of t (up to α);*
2. Black-Proper*: u is black-proper.*

Proof. By induction on the length k of $t \multimap^{k}_{\mathtt{PNeed}^{\circ}_{\bullet}} u$. □

We omit the details of the relationship between the unlabeled and the labeled systems, which is straightforward and would only distract the reader from the focus of the paper. Note indeed that the rewriting rules act on labels, but do not depend on them, so that both *lifting unlabeled derivations to the labeled system*, and *projecting labeled derivations by erasing labels* commute with evaluation.

Multiplicative vs Exponential Analysis. As for the previous proof of the linear bound, the reasoning here is based on two facts that refine the trace and the syntactic boundedness properties. We use $|t|_{\circ}$ for the number of white substitutions in t and $|d|_{\mathtt{e}\circ}$ for the number of $\to_{\mathtt{e}\circ}$ steps in d.

White substitutions are created by multiplicative steps and consumed on first use. Therefore their number is exactly $|d|_m - |d|_{\mathtt{e}\circ}$. This is the labeled refinement of the trace property. Formally,

Lemma 6.2 (White Trace)**.** *Let t be initial and $d : t \multimap^{*}_{\mathtt{PNeed}^{\circ}_{\bullet}} u$. Then $|u|_{\circ} = |d|_m - |d|_{\mathtt{e}\circ}$.*

Proof. By induction on the length k of d.

1. *Base case*, *i.e.* $k = 0$. Then $|u|_\circ = 0$ because t is a λ-term (it has no explicit substitution) and $|d|_\circ = |d|_{\mathtt{e}\circ} = 0$, so the statement holds.
2. *Inductive case*, *i.e.* $k > 0$. Then $t \multimap^{k-1}_{\mathtt{PNeed}^\circ_\bullet} w \multimap_{\mathtt{PNeed}^\circ_\bullet} u$ and let d' be the derivation $t \multimap^{k-1}_{\mathtt{PNeed}^\circ_\bullet} w$. By *i.h.*, $|w|_\circ = |d'|_m - |d'|_{\mathtt{e}\circ}$. Cases of $w \multimap u$:
   (a) $w \multimap_{\mathtt{m}} u$. The step creates a new white substitution and does not duplicate/erase any other white substitution, so $|u|_\circ = |w|_\circ + 1$. Since $|d|_m = |d'|_m + 1$ and $|d|_{\mathtt{e}\circ} = |d'|_{\mathtt{e}\circ}$, the statement holds.
   (b) $w \to_{\mathtt{e}\circ} u$. By the subterm property (Lemma 6.1.1) the copied value has no substitution, so we have $|u|_\circ = |w|_\circ - 1$. Since $|d|_m = |d'|_m$ and $|d|_{\mathtt{e}\circ} = |d'|_{\mathtt{e}\circ} + 1$, the statement holds.
   (c) $w \to_{\mathtt{e}\bullet} u$. By the subterm property the copied value has no substitution, so $|u|_\circ = |w|_\circ$. Since $|d|_m = |d'|_m$ and $|d|_{\mathtt{e}\circ} = |d'|_{\mathtt{e}\circ}$, the statement holds.

□

Substitution sequences satisfy the following bounds.

Lemma 6.3 (Syntactic Boundedness)**.** *Let t be an initial term and $t \multimap^*_{\mathtt{PNeed}^\circ_\bullet} u$.*

1. Almost White Chains*: if $u \to_{\mathtt{e}^\circ_\bullet} \to_{\mathtt{e}^\circ_\bullet} w$ then $u \to_{\mathtt{e}^\circ_\bullet} \to_{\mathtt{e}\circ} w$, i.e. the second step is not black;*
2. Local Syntactic Boundedness on White Steps*: if $u \to^k_{\mathtt{e}\circ} w$ then $k \leq |u|_\circ$.*

The first point states that sequences of $\to_{\mathtt{e}\bullet}$ steps are degenerated, as they have at most length one, and can only appear after multiplicative steps. The second point is a refined version of the syntactic boundedness property for CBN (see Sect. 2).

Apparently, both our proofs rely on the same *almost white/basic chains* property. Note however that here the eventual black step is at the beginning of the chain, while in the previous section the eventual applicative step is at the end of the chain.

Proof.

1. By a technical and ordinary analysis of the possibilities, that can be found in the appendix of the conference paper [10].
2. By induction on k. If $k = 0$ the statement trivially holds. If $u \to_{\mathtt{e}\circ} r \to^{k-1}_{\mathtt{e}\circ} w$ by the subterm property (Lemma 6.1.1) the substitution step does not duplicate any substitution and turns exactly one white substitution into a black one. So, $|r|_\circ = |u|_\circ - 1$. By *i.h.* we obtain $k - 1 \leq |u|_\circ - 1$ and so $k \leq |u|_\circ$.

□

The *exact* bound on the overhead for practical CBNeed, valid for *any* derivation (no termination hypothesis) and matched by *e.g.* $\delta\delta$, can now easily be proved.

Theorem 6.4 (Linear Bound for Practical CBNeed)**.** *Let t be initial and d : $t \multimap^*_{\mathtt{PNeed}^\circ_\bullet} u$. Then*

1. Global Linear Black Bound*:* $|d|_{\mathsf{e}\bullet} \leq |d|_m$*;*
2. Global Linear White Bound*:* $|d|_{\mathsf{e}\circ} \leq |d|_m$*;*
3. Global Linear Bound*:* $|d|_e \leq 2|d|_m$.

The proof of Point 2 is the interesting part, where the local syntactic boundedness on white steps and the white trace property are used together.

Proof. Given that $\multimap_{\mathtt{m}}$ is evidently terminating, and according to Lemma 6.3, d writes uniquely as (where $\to^{(1)}_{\mathsf{e}\bullet}$ means 0 or 1 steps of $\to_{\mathsf{e}\bullet}$):

$$t = t_1 \multimap^{a_1}_{\mathtt{m}} w_1 \to^{(1)}_{\mathsf{e}\bullet} u_1 \to^{b_1}_{\mathsf{e}\circ} t_2 \ldots t_k \multimap^{a_k}_{\mathtt{m}} w_k \to^{(1)}_{\mathsf{e}\bullet} u_k \to^{b_k}_{\mathsf{e}\circ} u$$

Clearly $|d|_{\mathsf{e}\bullet} \leq |d|_m$, and Point 1 is proved.

For Point 2, let $d_i : t \multimap^*_{\mathtt{PNeed}^\circ_\bullet} w_i$ be the prefix of d ending on w_i (including a_j and b_j for $j < i$, plus a_i, but not b_i). Note that defining $b_0 := 0$ we obtain $|d_i|_{\mathsf{e}\circ} = \sum_{j=0}^{i-1} b_j$ for $i \in \{1, \ldots, k\}$. Now we can easily estimate the generic term b_i and conclude the proof of the point:

$$b_i \leq_{L.6.3} |u_i|_\circ =_{L.6.2} |d_i|_m - |d_i|_{\mathsf{e}\circ} = |d_i|_m - \textstyle\sum_{j=0}^{i-1} b_j$$

$$|d|_{\mathsf{e}\circ} = \textstyle\sum_{i=0}^{k} b_i = b_k + \sum_{i=0}^{k-1} b_i \leq |d_k|_m - \sum_{j=0}^{k-1} b_j + \sum_{i=0}^{k-1} b_i = |d_k|_m = |d|_m$$

Point 3, follows from Point 1, Point 2, and $|d|_e = |d|_{\mathsf{e}\circ} + |d|_{\mathsf{e}\bullet}$. □

Let us conclude with a comment. The call-by-need LSC, used here and in [4], can be seen as a variant of Chang and Felleisen's calculus [12], that is a λ-calculus without explicit substitutions implementing call-by-need by micro-step evaluation and only one contextual rewriting rule. The result we just obtained shows that a syntax having an explicit constructor for substitutions may provide insights that are not accessible using the traditional syntax of λ-calculus.

7. Practical Call-by-Name

In this section we reconsider CBN and define a *practical* variant. We simply remove variables from the set of substitutable terms, but since the rules are not defined by means of values, the variant is slightly more technical to define. We show that also this further practical case has a linear overhead.

The case study of Practical CBN, however, is quite peculiar. On the one hand it is expected to be linear, in analogy to CBV and CBNeed. On the other hand CBN allows for generalized chains of substitutions—impossible in CBV and CBNeed—that look suspicious, and seem to suggest that the overhead is in fact quadratic. Let us sketch the issue.

Generalized Chains are Harmless. Think of renaming chains as terms of the form:

$$t[x_0 \leftarrow x_1][x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow x_n][x_n \leftarrow u]$$

where u is an abstraction. Renaming chains are in fact more general, as the substitutions of the chain are not necessarily next to each other. We will soon have to deal with the general form, but for the sake of the present explanation let us consider such a simple form.

In CBV and CBNeed these are the only possible chains. In CBN however the generic variable x_i can be replaced by an application having x_i as head variable and u need not to be an abstraction. Let us then consider *generalized chains* of the form:

$$t[x_0 \leftarrow H_1\langle x_1\rangle][x_1 \leftarrow H_2\langle x_2\rangle] \dots [x_{n-1} \leftarrow H_n\langle x_n\rangle][x_n \leftarrow u]$$

Turning to practical values removes renaming chains, but it does not remove such generalized chains. One then suspects that the quadratic overhead still affects Practical CBN. Somewhat surprisingly, instead, generalized chains are harmless and the overhead is linear.

Defining Practical CBN. To implement Practical CBN the exponential rule has to be changed, forbidding the substitution of variables. Since this obviously blocks evaluation we have to add a further exponential rule, whose role is to walk through a chain of renaming substitutions until it finds a non renaming one that can act. This is analogous to what happens in CBNeed, where evaluation enters into substitutions as well, but it is simpler. Practical CBN, indeed, enters substitutions only to shorten renaming chains, not to share evaluation. In particular, CBNeed multiplicative steps can take place inside substitutions, while in Practical CBN this cannot happen.

The *walk* is implemented via a new notion of context. We first give the definition and then provide explanations.

Chain contexts are defined by:

$$C ::= H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle] \mid C\langle x\rangle[x{\leftarrow}\langle\cdot\rangle] \mid H\langle C\rangle$$

The exponential rule is then split in two, the *shallow* and the *chain* case:

$$\begin{array}{rll} & \textsc{Rule at Top Level} & \\ \text{Shallow :} & H\langle x\rangle[x{\leftarrow}u] \mapsto_{\mathsf{se}} H\langle u\rangle[x{\leftarrow}u] & \text{if } u \text{ is not a variable} \\ \text{Chain :} & C\langle x\rangle[x{\leftarrow}u] \mapsto_{\mathsf{ce}} C\langle u\rangle[x{\leftarrow}u] & \text{if } u \text{ is not a variable} \end{array}$$

$$\begin{array}{rll} & \textsc{Contextual Closure} & \\ \text{Shallow :} & H\langle t\rangle \multimap_{\mathsf{se}} H\langle u\rangle & \text{iff } t \mapsto_{\mathsf{se}} u \\ \text{Chain :} & H\langle t\rangle \multimap_{\mathsf{ce}} H\langle u\rangle & \text{iff } t \mapsto_{\mathsf{ce}} u \end{array}$$

The multiplicative rule is left unchanged.

Explaining Chain Contexts. Let us give a few examples, motivating the definitions of chain contexts. The first case of chain context enters in a chain. It is used in $\multimap_{\mathsf{ce}}$ when the length of the chain is 1. Consider the following example (I is the identity):

$$x[x{\leftarrow}y][y{\leftarrow}I] \multimap_{\mathsf{ce}} x[x{\leftarrow}I][y{\leftarrow}I]$$

where the chain context is $x[x{\leftarrow}\langle\cdot\rangle]$, *i.e.* of the form $H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]$.

The second case is used to walk through the chain, handling chains of length greater than 1. For instance

$$x[x{\leftarrow}y][y{\leftarrow}z][z{\leftarrow}I] \multimap_{\mathsf{ce}} x[x{\leftarrow}y][y{\leftarrow}I][z{\leftarrow}I]$$

where the chain context is $x[x{\leftarrow}y][y{\leftarrow}\langle\cdot\rangle]$, *i.e.* of the form $C\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]$.

The third case is $C := H\langle C'\rangle$, that could equivalently be replaced by the two productions $C := C't$ and $C := C'[x{\leftarrow}t]$. It is used to intertwine the substitutions of a chain with other independent constructors in the term, like in:

$$(x[x{\leftarrow}y]t)[y{\leftarrow}z][x'{\leftarrow}u][z{\leftarrow}I] \multimap_{\mathsf{ce}} (x[x{\leftarrow}y]t)[y{\leftarrow}I][x'{\leftarrow}u][z{\leftarrow}I]$$

where the substitutions of the previous example are alternated with applications and unrelated substitutions.

Last, one could easily merge the two exponential rules, by defining chain contexts to include weak head contexts, *i.e.* using the following definition of chain context:

$$C ::= H \mid C\langle x\rangle[x\leftarrow\langle\cdot\rangle] \mid H\langle C\rangle$$

However in our analysis it will be necessary to distinguish between shallow and chain steps, which is why we did not employ this more compact presentation.

Correctness. One should of course prove that our practical strategy implements CBN evaluation. In particular, that our notion of chain context is well-defined, so that evaluation does not stop prematurely because a case has been forgotten. Such a study is however omitted: it is routine and would only distract from the focus of the paper. Anyway, for the skeptical reader we provide a high-level perspective, based on two remarks:

1. *The unfolding does not change*. Practical CBN only refines the exponential rule. The routine proof that the strategy implements small-step CBN is obtained by projecting multiplicative steps on β-steps via *unfolding*, *i.e.* by turning explicit substitutions into ordinary meta-level substitutions. A key point is that exponential steps do not change the unfolding, so that the modification of the rules is safe with respect to projection.
2. *Evaluation does not stop prematurely*. One may suspect that our exponential rules do not cover all possible cases. Let us explain why our definition is correct. Remember that evaluation is weak and only considers closed terms. Now, if $t = H\langle x\rangle$ it is easy to see that there must be a chain of substitutions s.t. $t = C\langle u\rangle$ with u not a variable, otherwise t would be open.

Multiplicative vs Exponential Analysis. The evaluations of $\delta\delta$ in practical CBN and practical CBNeed (page 21) coincide. This is what makes us conjecture that the exact bound is $|d|_e \leq 2|d|_m$. Our proof provides only $|d|_e \leq 3|d|_m + 1$. It is along the lines of the CBV case, *i.e.* it uses a notion of size and requires that evaluation terminates. First of all, the usual invariants.

Lemma 7.1 (Practical CBN Invariants)**.** *Let t be initial and $d : t \multimap^*_{\mathtt{PCBN}} u$.*

1. Box*: every argument and the content of every substitution in u are subterms of t (up to α);*
2. Subterm*: the terms duplicated along d are subterms of t (up to α);*
3. Trace*: the number $|u|_{[\ ]}$ of explicit substitutions in u is exactly $|d|_m$.*

Proof. Easy inductions on the length of d. Point 1 is used to prove Point 2, in turn used to prove Point 3. □

Let $|u|_{\neg x}$ be 0 if u is a variable and 1 otherwise.

Definition 7.2 (CBN Size)**.** *The* CBN size $|\cdot|_{CBN}$ *for terms, evaluation, and chain contexts is defined recursively as follows:*

$$\begin{array}{rcl}
|x|_{CBN} & := & 0 \\
|v|_{CBN} & := & 1 \\
|tu|_{CBN} & := & |t|_{CBN} + |u|_{\neg x} + 1 \\
|t[x{\leftarrow}u]|_{CBN} & := & |t|_{CBN} + |u|_{\neg x} + 1 \\
 & & \\
|\langle\cdot\rangle|_{CBN} & := & 0 \\
|Hu|_{CBN} & := & |H|_{CBN} + |u|_{\neg x} + 1 \\
|H[x{\leftarrow}u]|_{CBN} & := & |H|_{CBN} + |u|_{\neg x} + 1 \\
 & & \\
|H\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]|_{CBN} & := & |H\langle x\rangle|_{CBN} + 1 \\
|C\langle x\rangle[x{\leftarrow}\langle\cdot\rangle]|_{CBN} & := & |C\langle x\rangle|_{CBN} + 1 \\
|H\langle C\rangle|_{CBN} & := & |H|_{CBN} + |C|_{CBN}
\end{array}$$

We have the following immediate property of CBN size.

Lemma 7.3 (Contextual Factorization)**.** *Let H and C be a CBN evaluation and a chain context, respectively. Then*

1. $|H\langle t\rangle|_{CBN} = |H|_{CBN} + |t|_{CBN}$
2. $|C\langle t\rangle|_{CBN} = |C|_{CBN} + |t|_{\neg x}$.

Proof.

1. By induction on H.
2. By induction on C.

□

As for the CBV case just one simple lemma relates $\multimap_{\mathtt{e}}$, $\multimap_{\mathtt{m}}$, and the CBN size, providing the global linear bound for the terminating case as a corollary.

Lemma 7.4 (Main Practical CBN Invariant)**.** *Let $d : t \multimap^{n}_{\mathtt{PCBN}} u$. Then $|d|_e \leq |d|_m + |u|_{CBN}$.*

Proof. By induction on n. Case $n = 0$ is obvious. Otherwise $d' : t \multimap^{n-1}_{\mathtt{PCBN}} w$ and d extends d' with $w \multimap_{\mathtt{PCBN}} u$. By *i.h.*, $|d'|_e \leq |d'|_m + |w|_{CBN}$. Cases of the last step:

- *Shallow Exponential.* Then

$$w = H\langle H'\langle x\rangle[x\leftarrow r]\rangle \multimap_{\mathtt{se}} H\langle H'\langle r\rangle[x\leftarrow r]\rangle = u$$

with r not a variable. Note that since r is not a variable we have $|r|_{CBN} \geq 1$ (this is the key property that fails in Theoretical CBN). Then

$$\begin{array}{rcll} |d|_e & = & |d'|_e + 1 & \leq_{i.h.} \quad |d'|_m + |w|_{CBN} + 1 \\ & & & = \quad |d|_m + |w|_{CBN} + 1 \\ & & & \leq \quad |d|_m + |w|_{CBN} + |r|_{CBN} \\ & & & =_{L.7.3.1} \quad |d|_m + |H\langle H'[x\leftarrow r]\rangle|_{CBN} + |r|_{CBN} \\ & & & =_{L.7.3.1} \quad |d|_m + |u|_{CBN} \end{array}$$

- *Chain Exponential.* Then

$$w = H\langle C\langle x\rangle[x\leftarrow r]\rangle \multimap_{\mathtt{ce}} H\langle C\langle r\rangle[x\leftarrow r]\rangle = u$$

with r not a variable. Note that $|H\langle C\langle r\rangle[x\leftarrow r]\rangle|_{CBN} =_{L.7.3.2} |H\langle C[x\leftarrow r]\rangle|_{CBN} + |r|_{\neg x}$ and that $|r|_{\neg x} = 1$ (again, this would fail in Theoretical CBN). Then

$$\begin{array}{rcll} |d|_e & = & |d'|_e + 1 & \leq_{i.h.} \quad |d'|_m + |w|_{CBN} + 1 \\ & & & = \quad |d|_m + |w|_{CBN} + 1 \\ & & & = \quad |d|_m + |w|_{CBN} + |r|_{\neg x} \\ & & & =_{L.7.3.2} \quad |d|_m + |H\langle C[x\leftarrow r]\rangle|_{CBN} + |r|_{\neg x} \\ & & & =_{L.7.3.2} \quad |d|_m + |u|_{CBN} \end{array}$$

- *Multiplicative*. Then

$$w = H\langle L\langle \lambda x.q\rangle r\rangle \multimap_{\mathtt{m}} H\langle L\langle q[x\leftarrow r]\rangle\rangle = u$$

Note also that $|d|_m = |d'|_m + 1$ and $|L\langle q[x\leftarrow r]\rangle|_{CBN} = |L\langle \lambda x.q\rangle r|_{CBN} - 1 + |q|_{CBN}$, *i.e.* $|L\langle \lambda x.q\rangle r|_{CBN} = |L\langle q[x\leftarrow r]\rangle|_{CBN} + 1 - |q|_{CBN}$. Thus

$$\begin{array}{rcll} |d|_e & = & |d'|_e & \leq_{i.h.} \quad |d'|_m + |w|_{CBN} \\ & & & = \quad |d|_m - 1 + |w|_{CBN} \\ & & & =_{L.7.3.1} \quad |d|_m - 1 + |H|_{CBN} + |L\langle \lambda x.q\rangle r|_{CBN} \\ & & & = \quad |d|_m - 1 + |H|_{CBN} + |L\langle q[x\leftarrow r]\rangle|_{CBN} + 1 - |q|_{CBN} \\ & & & \leq \quad |d|_m + |H|_{CBN} + |L\langle q[x\leftarrow r]\rangle|_{CBN} - |q|_{CBN} \\ & & & =_{L.7.3.1} \quad |d|_m + |u|_{CBN} - |q|_{CBN} \\ & & & \leq \quad |d|_m + |u|_{CBN} \end{array}$$

□

Corollary 7.5 (Linear Bound for Practical CBN)**.** *Let t be initial and $d : t \multimap^*_{\mathtt{PCBN}} L\langle v\rangle$. Then $|d|_e \leq 3|d|_m + 1$.*

Proof. Let's estimate $L\langle v\rangle$. Every substitution in L contributes at most 2, as 1 is given by the substitution itself, and 1 bounds the value of $|\cdot|_{\neg x}$ on its content. The trace invariant (Lemma 7.1.3) gives $|L\langle v\rangle|_{[\ ]} = |d|_m$. Finally, the value v counts for 1. Summing up, $|L\langle v\rangle|_{CBN} \leq 2|d|_m + 1$. Substituting such a bound in the main invariant (Lemma 7.4) gives: $|d|_e \leq_{L.7.4} |d|_m + |L\langle v\rangle|_{CBN} \leq |d|_m + 2|d|_m + 1 = 3|d|_m + 1$. □

We conclude with a few comments:

- *Exact Bound*. We do not know if the obtained bound is exact. We were not able to find an example reaching the bound, nor to improve the bound. We conjecture that the bound can be improved to $2|d|_m$, as for CBV and CBNeed. The proof schema used here, relying on a static measure as for CBV, seems to not be suited for the exact bound, as it forces constraints on the measure that prevents to obtain $2|d|_m$. Probably a refinement along the lines of the labeled CBNeed proof, based on a more dynamic analysis, could provide the exact bound. Such a refinement however is non-trivial, as Practical CBN lacks some of the properties of Practical CBNeed, in particular substituted terms are not necessarily abstractions.

- *Comparison with Sands, Gustavssons, and Moran*. The analysis in this section essentially adapts the measure for the optimized CBN machine of [29]. The difference is that our approach is lazier. In [29], an optimized CBN is obtained by modifying the multiplicative rather than the exponential rule. The idea is to look at the argument of a multiplicative redex, and, in case it is a variable, the argument is substituted on-the-fly. Namely the multiplicative rule splits as:

$$\begin{array}{lll} L\langle \lambda x.t\rangle y & \multimap_{\mathtt{m1}} & L\langle t\{x\leftarrow y\}\rangle \\ L\langle \lambda x.t\rangle u & \multimap_{\mathtt{m2}} & L\langle t[x\leftarrow u]\rangle \quad \text{if } u \text{ is not a variable} \end{array}$$

  In such an eager approach the advantage is that renaming chains are never created, obtaining a very compact environment. The same optimization appears in Wand's [32] (section 2), Friedman et al.'s [17] (section 4), and Sestoft's [30] (section 4), motivated as an optimization about *space* rather

than *time*. The drawback of the eager approach are that chains are *always* shortened, even if they will never be used, and that multiplicative steps are no longer implementable in constant time (even if the change does not affect the overall complexity).

- *General Optimization*. Beyond the obvious attempt to say something new, we preferred to study the lazier variant because 1) it does not touch β-redexes but only the substitution process, which is from where the problem originates, and 2) we employ it also in our recent work [6], where we study the micro-step evaluation of an extended CBV calculus, suffering of malicious renaming chains. The aim, then, is to show that *walking through renaming chains* is a general optimization, independent of the contingent setting.

8. Remarks about the Complexity of Evaluation

This paper provides linear and quadratic bounds on the number of substitution *steps*. To extract proper complexity bounds one should also consider the complexity of implementing a rewriting step. Of course, it depends very much on the actual representation of terms and on the overhead of the actual abstract machine implementing the strategy of interest. A uniform abstract view is however possible.

Multiplicative steps are in general implementable in constant time. Exponential steps are more complex, as they require the copy of a subterm. Thanks to the subterm property, an exponential step takes time at most linear in the size of the initial term. Consequently, the substitution overhead is actually *bilinear*, *i.e.* linear in the *number of β-steps* and in the *size of the initial term*.

Environment-based abstract machines implement the rewriting steps within these bounds. Some machines are implemented without ever copying subterms, but this comes at other expenses (typically the handling of local environments, bound by the size of the initial term, see [4]) that re-introduce a dependency from the initial term. Therefore, the complexity is bilinear even when the implementation does not actually copy subterms.

Beyond such a size factor due to exponential steps, abstract machines add a further overhead given by the search of the next redex to reduce. In [4] such an overhead is shown to be bilinear for *any* machine execution of a number of CBN/CBV/CBNeed machines. In [29] the authors show that for executions to normal form the overhead for the search of the redex becomes *linear* (they prove

it for a CBV machine and an optimized CBN machine). Such an improvement is somewhat surprising and worth to be better understood. Asymptotically, however, it does not change the situation because the bilinearity of exponential steps cannot be removed.

Summing up, with practical values all weak evaluation schemes are implementable with a *bilinear* overhead, and such an asymptotical behavior seems to be independent of the actual implementation technique (de Bruijn indexes, graph-reduction, local/global environments, copy of subterms, etc). Dually, with theoretical values all weak evaluation schemes have an overhead *quadratic* in the number of steps and *linear* in the size of the initial term.

Conclusions

This paper provided a theoretical explanation for a subtle point about the implementation of functional languages: avoiding the substitution of variables improves the substitution overhead from quadratic to linear, for every evaluation scheme. Linear bounds already appeared in the literature [29, 14]. Our additions to the picture were:

1. A high-level point of view on the role of variables, identifying malicious chains of renaming as the responsible of inefficient overheads.
2. The identification of a critical pair for micro-step CBV whose different closing paths correspond to employ theoretical or practical values, explaining the lack of uniformity in the literature.
3. Symmetric results for CBN, CBV, and CBNeed. The literature suggests an asymmetric reading of overheads, for which CBV and CBNeed have a naturally lower overhead than CBN, that can be made equally efficient with a further optimization. We restore the symmetry showing that if variables are substituted then CBV and CBNeed are as inefficient as CBN. Said differently, substitution overheads are determined by the value of variables and not by the evaluation scheme.
4. A detailed, abstract, and modular study of the problem. We provided both a high-level decomposition of the proofs and a sharp low-level analysis of the dynamics.

Acknowledgements

To Pablo Barenbaum, for help with some technical details.

Bibliography

[1] Accattoli, B., 2012. An abstract factorization theorem for explicit substitutions. In: 23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan. pp. 6–21.
URL http://dx.doi.org/10.4230/LIPIcs.RTA.2012.6

[2] Accattoli, B., 2012. Proof nets and the call-by-value lambda-calculus. In: Proceedings Seventh Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2012, Rio de Janeiro, Brazil, September 29-30, 2012. pp. 11–26.
URL http://dx.doi.org/10.4204/EPTCS.113.5

[3] Accattoli, B., 2013. Evaluating functions as processes. In: Proceedings 7th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2013, Rome, Italy, 23th March 2013. pp. 41–55.
URL http://dx.doi.org/10.4204/EPTCS.110.6

[4] Accattoli, B., Barenbaum, P., Mazza, D., 2014. Distilling abstract machines. In: Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014. pp. 363–376.
URL http://doi.acm.org/10.1145/2628136.2628154

[5] Accattoli, B., Bonelli, E., Kesner, D., Lombardi, C., 2014. A nonstandard standardization theorem. In: The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014. pp. 659–670.
URL http://doi.acm.org/10.1145/2535838.2535886

[6] Accattoli, B., Coen, C. S., 2015. On the relative usefulness of fireballs. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015. pp. 141–155.
URL http://dx.doi.org/10.1109/LICS.2015.23

[7] Accattoli, B., Dal Lago, U., 2014. Beta reduction is invariant, indeed. In: Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July

14 - 18, 2014. p. 8.
URL http://doi.acm.org/10.1145/2603088.2603105

[8] Accattoli, B., Guerrieri, G., 2016. Implementing open call-by-value. Submitted.
URL https://sites.google.com/site/beniaminoaccattoli/Accattoli%2C%20Guerrieri%20-%20Implementing%20Open%20CbV.pdf?attredirects=0

[9] Accattoli, B., Lago, U. D., 2012. On the invariance of the unitary cost model for head reduction. In: 23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan. pp. 22–37.
URL http://dx.doi.org/10.4230/LIPIcs.RTA.2012.22

[10] Accattoli, B., Sacerdoti Coen, C., 2014. On the value of variables. In: Logic, Language, Information, and Computation - 21st International Workshop, WoLLIC 2014, Valparaíso, Chile, September 1-4, 2014. Proceedings. pp. 36–50.
URL http://dx.doi.org/10.1007/978-3-662-44145-9_3

[11] Ariola, Z. M., Felleisen, M., 1997. The call-by-need lambda calculus. J. Funct. Program. 7 (3), 265–301.

[12] Chang, S., Felleisen, M., 2012. The call-by-need lambda calculus, revisited. In: Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. pp. 128–147.
URL http://dx.doi.org/10.1007/978-3-642-28869-2_7

[13] Crank, E., Felleisen, M., 1991. Parameter-passing and the lambda calculus. In: Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando, Florida, USA, January 21-23, 1991. pp. 233–244.
URL http://doi.acm.org/10.1145/99583.99616

[14] Dal Lago, U., Martini, S., 2009. On constructor rewrite systems and the lambda-calculus. In: Automata, Languages and Programming, 36th Internatilonal Collogquium, ICALP 2009, Rhodes, greece, July 5-12, 2009, Pro-

ceedings, Part II. pp. 163–174.
URL http://dx.doi.org/10.1007/978-3-642-02930-1_14

[15] Danos, V., Regnier, L., 2004. Head linear reduction. Tech. rep.

[16] Danvy, O., Zerny, I., 2013. A synthetic operational account of call-by-need evaluation. In: 15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013. pp. 97–108.
URL http://doi.acm.org/10.1145/2505879.2505898

[17] Friedman, D. P., Ghuloum, A., Siek, J. G., Winebarger, O. L., 2007. Improving the lazy krivine machine. Higher-Order and Symbolic Computation 20 (3), 271–293.
URL http://dx.doi.org/10.1007/s10990-007-9014-0

[18] Girard, J.-Y., 1987. Linear logic. Theoretical Computer Science 50, 1–102.

[19] Kesner, D., 2016. Reasoning about call-by-need by means of types. In: Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. pp. 424–441.
URL http://dx.doi.org/10.1007/978-3-662-49630-5_25

[20] Landin, P. J., Jan. 1964. The Mechanical Evaluation of Expressions. The Computer Journal 6 (4), 308–320.
URL http://dx.doi.org/10.1093/comjnl/6.4.308

[21] Launchbury, J., 1993. A natural semantics for lazy evaluation. In: Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993. pp. 144–154.
URL http://doi.acm.org/10.1145/158511.158618

[22] Maraist, J., Odersky, M., Wadler, P., 1998. The call-by-need lambda calculus. J. Funct. Program. 8 (3), 275–317.

[23] Milner, R., 2007. Local bigraphs and confluence: Two conjectures. Electr. Notes Theor. Comput. Sci. 175 (3), 65–73.

[24] Moran, A., Sands, D., 1999. Improvement in a lazy context: An operational theory for call-by-need. In: POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999. pp. 43–56.
URL http://doi.acm.org/10.1145/292540.292547

[25] Pfenning, F., Simmons, R. J., 2009. Substructural operational semantics as ordered logic programming. In: Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA. pp. 101–110.
URL http://dx.doi.org/10.1109/LICS.2009.8

[26] Pierce, B. C., 2002. Types and Programming Languages. MIT Press, Cambridge, MA, USA.

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

[28] Ronchi Della Rocca, S., Paolini, L., 2004. The Parametric λ-Calculus. Springer Berlin Heidelberg.

[29] Sands, D., Gustavsson, J., Moran, A., 2002. Lambda calculi and linear speedups. In: The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]. pp. 60–84.
URL http://dx.doi.org/10.1007/3-540-36377-7_4

[30] Sestoft, P., 1997. Deriving a lazy abstract machine. J. Funct. Program. 7 (3), 231–264.
URL http://journals.cambridge.org/action/displayAbstract?aid=44087

[31] Wadsworth, C. P., 1980. Some unusual λ-calculus numeral systems. In: Seldin, J., Hindley, J. (Eds.), To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press.

[32] Wand, M., 2007. On the correctness of the krivine machine. Higher-Order and Symbolic Computation 20 (3), 231–235.
URL http://dx.doi.org/10.1007/s10990-007-9019-8