On the Value of Variables

Beniamino Accattoli and Claudio Sacerdoti Coen

Università di Bologna

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 the time). Namely, the number of substitution steps becomes linear in the number of β-redexes, while theoretical values only provide a quadratic bound.

1 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 weak, *i.e.* it does not take place under abstraction.

Theoretical and Practical Values. Plotkin's call-by-value λ-calculus [1] is a theoretical object of study introduced to model a concrete case, Landin's SECD machine [2]. 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 call-by-value abstract machines (or imperative extensions of Plotkin's calculus [3]) employ a notion of *practical value* that includes abstractions and excludes variables. For instance, Paolini and Ronchi della Rocca's book [4] 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 [5], 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.

In this paper we close the gap between theory and practice, providing a theoretical justification for practical values. We show that 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 in relation to the number of β-redexes. To state and prove our claim it is necessary to switch to a refinement of the call-by-value λ-calculus where the usual *small-step semantics* is decomposed in a *micro-step semantics*, in which substitution acts on a variable occurrence at the time, *i.e.* with the granularity of abstract machines (or of that of *substructural* operational semantics [6]).

The Linear Substitution Calculus. Our framework is the Linear Substitution Calculus (LSC) [7,8,9], a calculus with explicit substitutions that is 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 [10], and—remarkably—the number of evaluation steps in the LSC is a reasonable measure of the time complexity of a λ-term [8,11]. 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 an explicit substitution. 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 mechanism at work in abstract machines.

Call-by-Name. Call-by-name does not rely on values, or, equivalently, everything, including variables, is a value. Using the call-by-name LSC, in [8] it is shown that the number of substitutions steps ($\multimap_{\mathtt{e}}$) is quadratic in the number of β steps ($\multimap_{\mathtt{m}}$). The worst cases, *i.e.* those reaching the quadratic bound, are given by sequences where between any two multiplicative steps (corresponding to β-redexes) there is a chain of substitution steps of length linear in the number of preceding multiplicative steps.

Call-by-Value. In the call-by-value LSC, if values are *theoretical* then the chains of substitution steps at work in call-by-name case are still possible, and so the bound is quadratic. 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* relationship between the number of substitution steps ($\multimap_{\mathtt{e}}$) and the number of β steps ($\multimap_{\mathtt{m}}$). The proof of the bound is particularly simple and, curiously, it holds only under the assumption that evaluation terminates.

Call-by-Need. We then deal with call-by-need evaluation, that is usually defined using practical values [12,13,14,15,16] and that can be modularly expressed in the LSC. As for call-by-value, theoretical values induce a quadratic bound, while practical values provide a linear bound. The proof, however, is inherently different. It is technically more involved and it does not require the termination assumption.

New Speed-Up. Summing up, the two contributions of the paper are the linear bounds for call-by-value and call-by-need. These evaluation strategies are usually considered to speed up call-by-name evaluation because they reduce redexes in arguments *before* the arguments are substituted, implementing a form of sharing. Our results show that they also provide a subtler and deeper speed-up with respect to call-by-name: there are terms that take the same number k of β-steps to evaluate to normal form in call-by-name/value, and yet their micro-step evaluation takes $O(k^2)$ steps in call-by-name and $O(k)$ steps in call-by-value/need.

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. Indeed, while both call-by-value and call-by-need have a logical foundation in the so-called *boring* translation of λ-calculus into linear logic [17,18], such translation wraps both variables and abstractions inside the ! modality—the connective allowing non-linear behaviour—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, so that 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. Let us conclude pointing out a companion paper [10], where for the LSC calculi considered here and several abstract machines from the literature, we show that the number of execution steps of the abstract machine is *linear* in the number of steps in the calculus. Via that work, our bounds apply to concrete implementation models.

Related Work. The only similar work we are aware of is Dal Lago and Martini's [19], where it is shown that evaluation in the call-by-value λ-calculus (corresponding to our $\multimap_{\mathtt{m}}$) and evaluation 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, however they employ practical values at the graphical level, exactly as our results prescribe.

2 Call-by-Name Analysis

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

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

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).

An *initial term* is a closed term (*i.e.* without free variables) 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 call-by-name (shortened CBN), evaluation contexts are defined by the following grammar:

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

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 $\langle t\rangle := t$ (resp. $\langle H'\rangle := H'$), $(Ht)\langle u\rangle := H\langle u\rangle t$ (resp. $(Ht)\langle H'\rangle := H\langle H'\rangle t$), and so on. *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 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 CBN, the rewriting relation is $\multimap := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$, 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 $t \mapsto_{\mathtt{e}} u$

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.

In the literature, $\multimap$ is known as *weak linear head reduction*. The 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.

Exponential vs Multiplicative Analysis. For CBN, the relationship between $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ is already well-known from the literature [8,11]. Given a derivation $d : t \multimap^* u$ let us note $|d|_e$ and $|d|_m$ the number of exponential and multiplicative steps, respectively. Then:

Theorem 1 (Quadratic Bound [8]). *Let $d : t \multimap^* u$ be a CBN derivation from an initial term t. Then $|d|_e = O(|d|_m^2)$ (and so $|d| = O(|d|_m^2)$).*

In [11] this result is generalized and its proof is axiomatized. In fact, it holds for any strategy having the two following abstract properties of $\multimap$ (using the notation of the theorem):

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

Their proof for $\multimap$ can be found in [8] or—in a more general form—in [11]. Then the bound can be proved easily.

Proof. Note that $\multimap_{\mathtt{m}}$ terminates, as the number of constructors decreases. The syntactic bound property gives termination of $\multimap_{\mathtt{e}}$. Then d has the shape:

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

By the syntactic bound property, we obtain $b_i \leq |r_i|_{[\,]}$. By the trace property we obtain $|r_i|_{[\,]} = \sum_{j=1}^{i} a_j$, and so $b_i \leq \sum_{j=1}^{i} a_j$. 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

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

The bound is tight, as it is reached for instance by $\delta\delta$ (where $\delta = \lambda x.(xx)$). In particular, its evaluation has subsequences of variable renamings of the form:

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

where it takes n renaming steps to obtain a multiplicative redex, that in turn generates a new sequence of $n+1$ renamings, and so on. In other words, these sequences meet the bound in the syntactic bound property.

Let us point out that the bound is reached also by some normalizing terms. Consider $\tau\tau n$ where $\tau = \lambda x.\lambda n.(n(\lambda y.y)(xx))$ and n is any Scott's numeral [20], defined by $[\![0]\!] = \lambda x.\lambda y.x$ and $[\![n+1]\!] = \lambda x.\lambda y.y[\![n]\!]$. Evaluating the term takes $(n+1)(n+4)/2$ exponential steps but only $4(n+1)$ multiplicative steps.

The trace and syntactic bound properties can be proved also for call-by-value and call-by-need variants of the calculus, obtaining a quadratic bound. But the next sections will show that for the variants of these strategies that employ practical values a finer analysis is possible, leading to a linear bound. These two results are new, and surprising in various ways:

1. *Variables*: for the linear bound it is crucial that values do not include variables. For instance, if variables are values $\delta\delta$ has exactly the same reductions in the three evaluation scheme considered, matching the quadratic bound. What is surprising is that it is enough to remove variables from values to decrease the asymptotic complexity of substitution.
2. *New Speed-Up*: the terms of the form $\tau\tau n$ mentioned before take the same number k of β-steps to evaluate to normal form in call-by-name and call-by-value, and yet their micro-step evaluation takes $O(k^2)$ steps in call-by-name and $O(k)$ steps in call-by-value.

3. *Linear Logic*: from a linear logic perspective the bound is quite unexpected. The exponentials (*i.e.* the substitutions), responsible for duplications, are expected to capture most of the computing time, while the multiplicatives are somehow 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 [8,11].

3 Call-by-Value Analysis

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

$$V ::= \langle\cdot\rangle \mid Vt \mid L\langle v\rangle V \mid V[x{\leftarrow}t]$$

In CBV, it can be easily shown that a closed term either diverges or produces an answer (but this property will not play a role in our analysis), and moreover the definiens of substitutions are also answers.

Rewriting Rules. We re-define $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ as follows:

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

As for call-by-name, we silently work modulo α-equivalence and in $\mapsto_{\mathtt{e}}$ the context V does not capture x nor the free variables of v. We also still use the notation $\multimap := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$.

Let us revisit the $\delta\delta$ example of Sect. 2, used to show that the quadratic bound is strict for CBN. Using CBV and theoretical values one obtains the same evaluation sequence. Practical values, instead, give:

$$\begin{array}{lll}
\delta\delta \multimap_{\mathtt{m}} (x_1x_1)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} & \\
(\delta x_1)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} & \\
(\delta\delta)[x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} & \\
(x_2x_2)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} & \\
(\delta x_2)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} & \\
(\delta\delta)[x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{m}} & \\
(x_3x_3)[x_3{\leftarrow}\delta][x_2{\leftarrow}\delta][x_1{\leftarrow}\delta] & \multimap_{\mathtt{e}} & \dots
\end{array} \tag{2}$$

Where it is easily seen that for any $d : \delta\delta \multimap^* t$ we have the linear relationship $|d|_e \leq 2 \cdot |d|_m$. This fact suggests that any CBV derivation d verifies $|d|_e = O(|d|_m)$. Curiously, this is not true in general. In particular, in CBV

a chain of substitution steps can be arbitrarily longer than the number of previous multiplicative steps. Let us give an example. Let t^n stay 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

$$(\lambda x.x^n)I \multimap_{\mathtt{m}} x^n[x \leftarrow I] \multimap_{\mathtt{e}}^n I^n$$

So n substitution steps $\multimap_{\mathtt{e}}$ after just one multiplicative step $\multimap_{\mathtt{m}}$. It seems even worse than in CBN, while instead, globally, it is a faster mechanism, of a different nature (note that the steps in the sequence are independent, *i.e.* they are not generated by chains of substitutions occurring one in the other as in CBN). The idea is that the substituted values create or will create new multiplicative redexes, so that if we keep reducing the term we will match the substitution steps in excess (if evaluation terminates, as in the example) and obtain a linear relationship between the two. The point is that in CBV the linear bound holds only for evaluation to normal form, otherwise the gap between $|d|_e$ and $|d|_m$ can be arbitrarily big.

Exponential vs Multiplicative Analysis. We first need some easy invariants.

Lemma 1 (CBV Invariants). *Let t be initial and $d : t \multimap^* u$.*

1. Subterm*: every value in u is a value in t;*
2. Trace*: the number $|u|_{[\,]}$ of explicit substitutions in u is exactly $|d|_m$;*
3. Proper*: every substitution in u contains an answer.*

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

Let us provide an intuition for the forthcoming proof of the linear bound. An exponential step makes a new copy of a value that will be eventually *consumed* by a multiplicative step, unless the term is divergent. A multiplicative step *consumes* the value in its left subterm. Therefore it is possible to bound the number of exponential steps with the number of consumed values (that is the number of multiplicative steps) plus the number of values in the term, what we call the *value size* of the term.

Definition 1 (Value Size). *The* value size $|\cdot|_\lambda$ *of a term counts the number of values that are not inside another value. It is defined recursively as follows:* $|x|_\lambda = 0$, $|v|_\lambda = 1$, $|t[x \leftarrow u]|_\lambda = |t|_\lambda + |u|_\lambda$, $|tu|_\lambda = |t|_\lambda + |u|_\lambda$.

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

Lemma 2 (Main Invariant). *Let $d : t \multimap^n u$. Then $|d|_e \leq |d|_m + |u|_\lambda - |t|_\lambda$.*

Proof. By induction over n. Case $n = 0$ is obvious. Otherwise $t \multimap w$ and $e : w \multimap^{n-1} u$ and, by inductive hypothesis, $|e|_e \leq |e|_m + |u|_\lambda - |w|_\lambda$. Cases:

– *the first step is exponential.* Then

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

and $|w|_\lambda = |t|_\lambda + 1$. Thus

$$\begin{aligned} |d|_e = |e|_e + 1 &\leq_{i.h.} |e|_m + |u|_\lambda - |w|_\lambda + 1 \\ &= |d|_m + |u|_\lambda - (|t|_\lambda + 1) + 1 = |d|_m + |u|_\lambda - |t|_\lambda \end{aligned}$$

– *the first step is multiplicative.* Then

$$t = V\langle L\langle\lambda x.r\rangle L'\langle v\rangle\rangle \multimap_{\mathtt{m}} V\langle L\langle r[x{\leftarrow}L'\langle v\rangle]\rangle\rangle = w$$

and $|w|_\lambda = |t|_\lambda - 1 + |r|_\lambda$. Thus

$$\begin{aligned} |d|_e = |e|_e &\leq_{i.h.} |e|_m + |u|_\lambda - |w|_\lambda \\ &= |d|_m - 1 + |u|_\lambda - (|t|_\lambda - 1 + |r|_\lambda) \\ &= |d|_m + |u|_\lambda - |t|_\lambda - |r|_\lambda \leq |d|_m + |u|_\lambda - |t|_\lambda \quad \square \end{aligned}$$

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

Proof. By the proper invariant every substitution contains a value plus some substitutions, each one recursively having the same shape, so $|L\langle v\rangle|_\lambda = |L\langle v\rangle|_{[\,]} + 1$, where 1 accounts for the value v. By the trace invariant $|L\langle v\rangle|_{[\,]} = |d|_m$, and so $|L\langle v\rangle|_\lambda \leq |d|_m + 1$. Then the main invariant gives: $|d|_e \leq |d|_m + |L\langle v\rangle|_\lambda - |t|_\lambda \leq |d|_m + |L\langle v\rangle|_\lambda \leq |d|_m + |d|_m + 1 = 2 \cdot |d|_m + 1$. $\square$

Invariance of the CEK machine. Our result on CBV has an implicit by-product. In [19] it is shown that Plotkin's calculus, whose steps can be identified with our $\multimap_{\mathtt{m}}$ steps, is invariant, *i.e.* polynomially related to models like Turing machines or random access machines, see the introduction of [11] for a presentation of the topic. Then, our result implies that the CBV LSC is invariant. In [10] it is shown that the CEK abstract machine [21] is linearly related to the CBV LSC. Therefore, the CEK is invariant. Such a result—albeit expected—is new.

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

$$V ::= \langle\cdot\rangle \mid VL\langle v\rangle \mid tV \mid V[x{\leftarrow}t]$$

Our proof for the bound with practical values holds unchanged also for the right-to-left strategy. However, it is unclear how right-to-left CBV behaves with theoretical values, as the typical quadratic example for theoretical left-to-right CBV, given by $\delta\delta$, is linear when evaluated with theoretical right-to-left CBV:

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

Note indeed that this is essentially the same evaluation as in (2). We do not know if for theoretical right-to-left CBV $\multimap_{\mathtt{m}}$ and $\multimap_{\mathtt{e}}$ are linearly related. We believe so, but the two proof techniques developed in this paper do not apply.

4 Call-by-Need Analysis

For call-by-need (CBNeed), the analysis is different and technically more involved. At first sight, CBNeed is very similar to call-by-name: the length of substitution sequences is bounded by the number $|d|_m$ of multiplicative steps previously performed, and the bound is easily reached. There is however a fundamental difference. 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 call-by-value, 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). More precisely, the chains are bound by the number of *unevaluated substitutions* rather than by the number of preceding multiplicative steps, according to the following scheme, that can be seen as a simple form of amortized analysis:

1. every multiplicative step produces an *unevaluated substitution*;
2. the first time an unevaluated substitution substitutes somewhere it changes status and becomes *evaluated*;
3. chains of substitution steps are bound by the number of unevaluated substitutions, that is always $\leq |d|_m$ and only *globally* equal to $|d|_m$.

Our proof will use a calculus enriched with labels on substitutions, to explicitly trace unevaluated substitutions. The labels will have no effect on the dynamics of the calculus, and are only meant as an aid for the proof.

The CBNeed Calculus. For the sake of clarity, we start by introducing the calculus, and then we start over introducing its labeled version. 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 *hereditarily weak head contexts*. This new clause is how sharing will be implemented by the strategy $\multimap := \multimap_{\mathtt{m}} \cup \multimap_{\mathtt{e}}$ defined by:

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 $t \mapsto_{\mathtt{e}} u$

The multiplicative rule is taken from the CBN calculus. Therefore the definiens of a substitution is not necessarily an answer. The exponential rule come instead

from the CBV calculus, and requires arguments to be evaluated to answers before being substituted, reflecting the *by need* content of the strategy.

Now that the calculus is defined, let us evaluate again $\delta\delta$. Using CBNeed and theoretical values it would evaluate exactly in the same way as for CBN. Practical values, instead, give:

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

Where it is easily seen that for any $d : \delta\delta \multimap^* t$ we have $|d|_e \leq 2 \cdot |d|_m$. 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.

The labeled CBNeed Calculus. The labeled language is:

$$t, u, w, r ::= x \mid v \mid tu \mid t[x{\leftarrow}u]^{\circ} \mid t[x{\leftarrow}u]^{\bullet}; \qquad v ::= \lambda x.t;$$

A *white* substitution $t[x{\leftarrow}u]^{\circ}$ represents an unevaluated 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 $t[x{\leftarrow}u]^{*}$ 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:

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

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 $t \mapsto_{\mathtt{m}} u$
$N\langle x\rangle[x{\leftarrow}L\langle v\rangle]^{\circ} \mapsto_{\mathtt{e}\circ} L\langle N\langle v\rangle[x{\leftarrow}v]^{\bullet}\rangle$	$N\langle t\rangle \to_{e\circ} N\langle u\rangle$	iff $t \mapsto_{\mathtt{e}\circ} u$
$N\langle x\rangle[x{\leftarrow}L\langle v\rangle]^{\bullet} \mapsto_{\mathtt{e}\bullet} L\langle N\langle v\rangle[x{\leftarrow}v]^{\bullet}\rangle$	$N\langle t\rangle \to_{e\bullet} N\langle u\rangle$	iff $t \mapsto_{\mathtt{e}\bullet} u$

The rewriting relation is $\multimap := \multimap_{\mathtt{m}} \cup \to_{e\circ} \cup \to_{e\bullet}$. Let $\to_{e*}$ stay for $\to_{e\circ}$ or $\to_{e\bullet}$. A term is *black-proper* if every black substitution contains a value.

Lemma 3 (Invariants). *Let t be a λ-term and $d : t \multimap^* u$.*

1. Subterm*: every value in u is a value in t.*
2. Black-Proper*: u is black-proper.*

Proof. By induction on the length k of $t \multimap^k u$. □

Since the reduction rules only duplicate values, we obtain that every duplicated subterm along a $\multimap$-execution is a subterm of the initial term.

Multiplicative vs Exponential Analysis. Essentially, we prove two facts that refine the abstract properties providing the quadratic bound for CBN. We use $|t|_\circ$ for the number of white substitutions in t and $|d|_{\mathtt{e}\circ}$ for the number of $\to_{e\circ}$ steps in d.

Lemma 4 (White Trace). *Let t be initial and $d : t \multimap^* u$. 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 $t \multimap^{k-1} w \multimap u$ and let e be the derivation $t \multimap^{k-1} w$. By *i.h.*, $|w|_\circ = |e|_m - |e|_{\mathtt{e}\circ}$. 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 $|d|_m = |e|_m + 1$ and $|d|_{\mathtt{e}\circ} = |e|_{\mathtt{e}\circ}$, the statement holds.
   (b) $w \to_{e\circ} u$. By the subterm property (Lemma 3.1) the copied value has no substitution, so we have $|u|_\circ = |w|_\circ - 1$. Since $|d|_m = |e|_m$ and $|d|_{\mathtt{e}\circ} = |e|_{\mathtt{e}\circ} + 1$, the statement holds.
   (c) $w \to_{e\bullet} u$. By the subterm property the copied value has no substitution, so $|u|_\circ = |w|_\circ$. Since $|d|_m = |e|_m$ and $|d|_{\mathtt{e}\circ} = |e|_{\mathtt{e}\circ}$, the statement holds.

□

By means of an omitted lemma (Lemma 6, page 13, in the appendix) we obtain the following bounds on substitution sequences.

Lemma 5. *Let t be an initial term and $t \multimap^* u$.*

1. Black Constant Bound*: If $u \to_{e*} \to_{e*} w$ then the second step is not black.*
2. White Syntactic Bound*: If $u \to_{e\circ}^k w$ then $k \leq |u|_\circ$.*

The first point states that sequences of $\to_{e\bullet}$ 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 bound for CBN (see Sect. 2).

Proof. The first point is given by the omitted Lemma 6.4. The second point is by induction on k. If $k = 0$ the statement trivially holds. If $u \to_{e\circ} r \to_{e\circ}^{k-1} w$ by the subterm property (Lemma 3.1) 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 $k \leq |u|_\circ$. □

Theorem 2 (Linear Bound for CBNeed). *Let t be initial and $d : t \multimap^* u$. Then $|d|_e \leq 2 \cdot |d|_m$.*

Proof. Given that $\multimap_{\mathtt{m}}$ is evidently terminating, and according to Lemma 5, d writes uniquely as (where $\to_{e\bullet}^{(1)}$ means 0 or 1 steps of $\to_{e\bullet}$):

$$t = t_1 \multimap_{\mathtt{m}}^{a_1} w_1 \to_{e\bullet}^{(1)} u_1 \to_{e\circ}^{b_1} t_2 \dots t_k \multimap_{\mathtt{m}}^{a_k} w_k \to_{e\bullet}^{(1)} u_k \to_{e\circ}^{b_k} u$$

Clearly $|d|_{\mathtt{e}\bullet} \leq |d|_m$. Since $|d|_e = |d|_{\mathtt{e}\circ} + |d|_{\mathtt{e}\bullet}$, we are left to show that $|d|_{\mathtt{e}\circ} \leq |d|_m$. Let $d_i : t \multimap^* w_i$ 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 $|d_i|_{\mathtt{e}\circ} = \sum_{j=0}^{i-1} b_j$ for $i \in \{1, \ldots, k\}$. Now we can easily estimate the generic term b_i and conclude:

$$b_i \leq_{Lemma\ 5} |u_i|_\circ =_{Lemma\ 4} |d_i|_m - |d_i|_{\mathtt{e}\circ} = |d_i|_m - \textstyle\sum_{j=0}^{i-1} b_j$$

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

On the Need of Labels. In fact, labels are not strictly necessary. It is possible to prove a linear relationship on the original CBNeed calculus, and the proof, along the same lines, is also slightly simpler (the role of white substitutions is played by those substitution whose content is a term of the form $L\langle x \rangle$). The price to pay however is that such an alternative analysis provides only a laxer—despite always linear—bound, as the multiplicative constant is higher (3 instead of 2). We preferred to use labels because the analysis they provide is *tight*, as it is shown by the $\delta\delta$ example, that reaches the bound given by Theorem 2.

Let us conclude with a comment. The call-by-need LSC can be seen as a variant of Chang and Felleisen's calculus [15], 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.

Acknowledgements

To Pablo Barenbaum, for discussions and help with some technical details.

References

1. Plotkin, G.D.: Call-by-name, call-by-value and the lambda-calculus. Theor. Comput. Sci. **1**(2) (1975) 125–159
2. Landin, P.J.: The Mechanical Evaluation of Expressions. The Computer Journal **6**(4) (January 1964) 308–320
3. Crank, E., Felleisen, M.: Parameter-passing and the lambda calculus. In: POPL. (1991) 233–244
4. Ronchi Della Rocca, S., Paolini, L.: The Parametric λ-Calculus. Springer Berlin Heidelberg (2004)
5. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge, MA, USA (2002)
6. Pfenning, F., Simmons, R.J.: Substructural operational semantics as ordered logic programming. In: LICS. (2009) 101–110
7. Accattoli, B.: An abstract factorization theorem for explicit substitutions. In: RTA. (2012) 6–21
8. Accattoli, B., Dal Lago, U.: On the invariance of the unitary cost model for head reduction. In: RTA. (2012) 22–37

9. Accattoli, B., Bonelli, E., Kesner, D., Lombardi, C.: A nonstandard standardization theorem. In: POPL. (2014) 659–670
10. Accattoli, B., Barenbaum, P., Mazza, D.: Distilling abstract machines. Accepted to ICFP 2014 (2014)
11. Accattoli, B., Dal Lago, U.: Beta Reduction is Invariant, Indeed. Accepted to LICS/CSL 2014 (2014)
12. Launchbury, J.: A natural semantics for lazy evaluation. In: POPL. (1993) 144–154
13. Ariola, Z.M., Felleisen, M.: The call-by-need lambda calculus. J. Funct. Program. **7**(3) (1997) 265–301
14. Maraist, J., Odersky, M., Wadler, P.: The call-by-need lambda calculus. J. Funct. Program. **8**(3) (1998) 275–317
15. Chang, S., Felleisen, M.: The call-by-need lambda calculus, revisited. In: ESOP. (2012) 128–147
16. Danvy, O., Zerny, I.: A synthetic operational account of call-by-need evaluation. In: PPDP. (2013) 97–108
17. Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
18. Accattoli, B.: Proof nets and the call-by-value lambda-calculus. In: LSFA. (2012) 11–26
19. Dal Lago, U., Martini, S.: On Constructor Rewrite Systems and the Lambda-Calculus. In: ICALP (2). (2009) 163–174
20. Wadsworth, C.P.: 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 (1980)
21. Felleisen, M., Friedman, D.P.: Control operators, the SECD-machine, and the lambda-calculus. In: 3rd Working Conference on the Formal Description of Programming Concepts. (August 1986)

Proofs Appendix

Lemma 6. *Let N be a call-by-need context.*

1. *If $N\langle x\rangle = M\langle y\rangle$ with x not bound by N and y not bound by M, then $N = M$ and $x = y$.*
2. *A term of the form $N\langle x\rangle$, with x not bound by N, is not of the form $M\langle v\rangle$.*
3. *Suppose $N\langle x_0\rangle$ is black-proper for some variable x_0, and $N\langle v\rangle \to_{e*} t$. Then the step is not $\to_{e\bullet}$.*
4. *If $t \to_{e*}\to_{e*} u$ then the second step is not black.*

Proof.

1. By induction on N.
   (a) *Empty context, i.e.* $N = \langle\cdot\rangle$: then M must be $\langle\cdot\rangle$ and $x = y$.
   (b) *Left of an application, i.e.* $N = N'\,t$: suppose $N'\langle x\rangle\, t$ is of the form $M\langle y\rangle$. Then $N'\langle x\rangle$ must be of the form $M'\langle y\rangle$, with $M = M'\,t$, and we conclude by *i.h.*.
   (c) *Left of a white or black substitution, i.e.* $N = N'[z{\leftarrow}t]^*$: suppose the $N\langle x\rangle = N'\langle x\rangle[z{\leftarrow}t]^*$ is also of the form $M\langle y\rangle$. There are two possibilities:
      i. *The hole of M is on the left of the substitution.* That is, $M = M'[z{\leftarrow}t]^*$ and $N'\langle x\rangle = M'\langle y\rangle$. We conclude by *i.h.*.

ii. *The hole of M is inside the substitution.* Then it must be that $M = M'\langle z\rangle[z{\leftarrow}M'']^*$. It follows $N'\langle x\rangle = M'\langle z\rangle$, with z not bound by M'. By *i.h.*, we conclude $x = z$, which is absurd since x is not bound by N. Hence this case is impossible.

(d) *Inside a white or black substitution, i.e.* $N = N'\langle z\rangle[z{\leftarrow}N'']^*$: suppose that $N\langle x\rangle = N'\langle z\rangle[z{\leftarrow}N''\langle x\rangle]^*$ is also of the form $M\langle y\rangle$. As in the previous case, there are two possibilities:

i. *The hole of M is on the left of the substitution.* That is, $M = M'[z{\leftarrow}t]^*$. In particular, we must have $N'\langle z\rangle = M'\langle y\rangle$ with z now not bound by N'. By *i.h.*, we conclude $y = z$ which is absurd. Hence this case is impossible.

ii. *The hole of M is inside the substitution.* That is, $M = M'\langle z\rangle[z{\leftarrow}M'']^*$. Hence we have that $N'\langle z\rangle[z{\leftarrow}N''\langle x\rangle]^* = M'\langle z\rangle[z{\leftarrow}M''\langle y\rangle]^*$. On one hand, this implies $N'\langle z\rangle = M'\langle z\rangle$ with z now free on both sides, which by *i.h.* gives us $N' = M'$. On the other, we obtain that $N''\langle x\rangle = M''\langle y\rangle$, that lets us conclude by resorting again to the *i.h.*.

2. By induction on N.

(a) *Empty context, i.e.* $N = \langle\cdot\rangle$: a variable cannot be of the form $M\langle v\rangle$.

(b) *Left of an application, i.e.* $N = N'\,t$: suppose $N\langle x\rangle = N'\langle x\rangle\,t$ is also of the form $M\langle v\rangle$. Then $N'\langle x\rangle$ must be of the form $M'\langle v\rangle$, which is impossible by *i.h.*.

(c) *Left of a white or black substitution, i.e.* $N = N'[z{\leftarrow}t]^*$: suppose that $N\langle x\rangle = N'\langle x\rangle[z{\leftarrow}t]^*$ is also of the form $M\langle v\rangle$. There are two possibilities:

i. *The hole of M is on the left of the substitution.* That is, $M = M'[z{\leftarrow}t]^*$ and $N'\langle x\rangle = M'\langle v\rangle$. This is impossible by *i.h.*.

ii. *The hole of M is inside the substitution.* That is, $M = M'\langle z\rangle[z{\leftarrow}M'']^*$. It follows that $N'\langle x\rangle = M'\langle z\rangle$, with z not bound by M'. By point 1 of this lemma, we conclude $x = z$, which is absurd since x is not bound by N. Hence this case is impossible.

(d) *Inside a white or black substitution, i.e.* $N = N'\langle z\rangle[z{\leftarrow}N'']^*$: suppose that $N\langle x\rangle = N'\langle z\rangle[z{\leftarrow}N''\langle x\rangle]^*$ is also of the form $M\langle v\rangle$. There are two possibilities:

i. *The hole of M is on the left of the substitution.* That is, $M = M'[z{\leftarrow}t]^*$. In particular, we must have $N'\langle z\rangle = M'\langle v\rangle$ with z now not bound by N'. By *i.h.*, this is impossible.

ii. *The hole of M is inside the substitution.* That is, $M = M'\langle z\rangle[z{\leftarrow}M'']^*$. Hence we have that $N'\langle z\rangle[z{\leftarrow}N''\langle x\rangle]^* = M'\langle z\rangle[z{\leftarrow}M''\langle v\rangle]^*$. From this we obtain that $N''\langle x\rangle = M''\langle v\rangle$ that is impossible by *i.h.*.

3. By induction on N.

(a) *Empty context, i.e.* $N = \langle\cdot\rangle$: trivial, since v is a normal form.

(b) *Left of an application, i.e.* $N = N'\,u$: any $\to_{e\bullet}$ redex in $N'\langle v\rangle\,u$ must be internal to $N'\langle v\rangle$, and we conclude this is impossible by *i.h.*.

(c) *Left of a white or black substitution, i.e.* $N = N'[x{\leftarrow}u]^*$: so $N\langle v\rangle$ is a substitution $N'\langle v\rangle[x{\leftarrow}u]^*$. There are three possibilities for a $\to_{e\bullet}$ step:

i. *The $\to_{e\bullet}$ step takes place on the left of the substitution, i.e.* internal to $N'\langle v\rangle$. This is impossible by *i.h.*.
ii. *The $\to_{e\bullet}$ step takes place inside the substitution.* It must then be that $N'\langle v\rangle$ is of the form $M'\langle x\rangle$. By point 2 of this lemma, this is impossible.
iii. *The $\to_{e\bullet}$ step is at the root.* In this case the substitution is black. Suppose $N'\langle v\rangle[x\leftarrow u]^\bullet \mapsto_{\mathtt{e}\bullet} t$. Then $N'\langle v\rangle$ must be of the form $M'\langle x\rangle$, which is impossible by point 2 of this lemma.

(d) *Inside a white or black substitution, i.e.* $N = N'\langle x\rangle[x\leftarrow N'']^*$: so $N\langle v\rangle$ is $N'\langle x\rangle[x\leftarrow N''\langle v\rangle]^*$. There are three possibilities for a $\to_{e\bullet}$ step:
i. *The $\to_{e\bullet}$ step takes place on the left of the substitution. i.e.* internal to $N'\langle x\rangle$. This means $N'\langle x\rangle$ can be written as $M\langle M'\langle y\rangle[y\leftarrow u]^\bullet\rangle$. Since the term is black-proper, u must be a value v'. Note also that $M'' := M\langle M'\langle y\rangle[y\leftarrow\langle\cdot\rangle]^\bullet\rangle$ is a call-by-need context. Then $N'\langle x\rangle$ can be written as of the form $M''\langle v'\rangle$. This is impossible by point 2 of this lemma.
ii. *The $\to_{e\bullet}$ step takes place inside the substitution. i.e.* internal to $N''\langle v\rangle$. This is impossible by *i.h.*.
iii. *The $\to_{e\bullet}$ step is at the root.* In this situation the substitution is black. Since $N'\langle x\rangle[x\leftarrow N''\langle v\rangle]^\bullet$ is black-proper by hypothesis, we know $N''\langle v\rangle$ must be a value, which implies that $N'' = \langle\cdot\rangle$. By hypothesis we also know that $N\langle x_0\rangle = N'\langle x\rangle[x\leftarrow N''\langle x_0\rangle]^\bullet = N'\langle x\rangle[x\leftarrow x_0]^\bullet$ is black-proper for some variable x_0. This is absurd, as the term is supposed to be black-proper but the black substitution contains a variable. Hence this case is impossible.

4. Let $t = N'\langle N\langle x\rangle[x\leftarrow L\langle v\rangle]^*\rangle \to_{e*} N'\langle L\langle N\langle v\rangle[x\leftarrow v]^\bullet\rangle\rangle \to_{e*} u$. By induction on N'. Cases:
(a) *Empty context, i.e.* $N' = \langle\cdot\rangle$ and
$$t = N\langle x\rangle[x\leftarrow L\langle v\rangle]^* \to_{e*} L\langle N\langle v\rangle[x\leftarrow v]^\bullet\rangle \to_{e*} u$$
It is easily seen that $L\langle N\langle v\rangle[x\leftarrow v]^\bullet\rangle \to_{e*} u$ because there exists w s.t. $N\langle v\rangle[x\leftarrow v]^\bullet \to_{e*} w$ and $L\langle w\rangle = u$ (variables bound by L can only occur in v and evaluation contexts do not go under abstractions). Then we are in the hypotheses of Point 3, that allows to conclude.
(b) *Left of an application, i.e.* $N' = N''\,u$: then any $\to_{e*}$ redex in $N''\langle L\langle N\langle v\rangle[x\leftarrow v]^\bullet\rangle\rangle\,u$ is internal to the left subterm, and we conclude using the *i.h.*.
(c) *Left of a white or black substitution, i.e.* $N' = N''[y\leftarrow w]^*$. Note that the second step cannot be an action of the substitution $[y\leftarrow w]^*$, because its left term is $N''\langle L\langle N\langle v\rangle[x\leftarrow v]^\bullet\rangle\rangle$ — *i.e.* a value in a CBNeed context — and by Point 2 it cannot be of the form $N'''\langle y\rangle$. Then the second step takes place in the left subterm and we conclude by the *i.h.*.
(d) *Inside a white or black substitution, i.e.* $N' = N''\langle x\rangle[x\leftarrow N''']^*$. Note that $[x\leftarrow N''']^*$ is necessarily white, as reduction took place inside it. If the second substitution step takes place inside N''' we conclude by the *i.h.*. Otherwise, the step is an action of $[x\leftarrow N''']^\circ$, that is a white step.