



HAL
open science

Positive Focusing is Directly Useful

Beniamino Accattoli, Jui-Hsuan Wu

► **To cite this version:**

Beniamino Accattoli, Jui-Hsuan Wu. Positive Focusing is Directly Useful. Proceedings of MFPS XL, Jun 2024, Oxford, United Kingdom. 10.46298/entics.14758 . hal-04886376

HAL Id: hal-04886376

<https://inria.hal.science/hal-04886376v1>

Submitted on 14 Jan 2025

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.



Distributed under a Creative Commons Attribution 4.0 International License

Positive Focusing is Directly Useful

Beniamino Accattoli¹ Jui-Hsuan Wu²

Inria & LIX, Ecole Polytechnique, UMR 7161, Palaiseau, France

Abstract

Recently, Miller and Wu introduced the *positive λ -calculus*, a call-by-value λ -calculus with sharing obtained by assigning proof terms to the positively polarized focused proofs for minimal intuitionistic logic. The positive λ -calculus stands out among λ -calculi with sharing for a *compactness* property related to the sharing of variables. We show that—thanks to compactness—the positive calculus neatly captures the core of *useful sharing*, a technique for the study of reasonable time cost models.

Keywords: λ -calculus, sharing, focusing, reasonable cost models

1 Introduction

Andreoli’s focusing is a technique for restricting the space of proofs to a subset with good structural properties, called *focused proofs* [14]. Notably, it helps in the *proof search* approach to computation.

Focused proof systems have quickly become a widespread tool in proof theory and—via the Curry-Howard isomorphism—in the study of λ -calculi. They are used well beyond proof search, for instance, in connection to pattern matching by Zeilberger [41] and Krishnaswami [24], proof nets and expansion proofs by Chaudhuri et al. [18,17], synthetic connectives by Chaudhuri [16], abstract machines by Brock-Nannestad et al. [15], decidability of contextual equivalence for sum types by Scherer and Rémy [36] and Scherer [35], contextual equivalence by Rioux and Zdancewic [31], synthetic inference rules by Marin et al. [27], refinement types by Economou et al. [21], and certainly in even more studies. The present paper adds one more instance, studying a connection between focusing and sharing for λ -terms.

Positive Focusing and λ -Terms. In [28], Miller and Wu show how the focused intuitionistic proof system LJF by Liang and Miller [26] can be used to design term structures. In LJF, formulas are polarized. A key theorem of LJF is that different polarizations do not affect provability: if a formula is provable (resp. not provable) in LJ, then the formula is provable (resp. not provable) in LJF with *any* polarization. Different polarizations of a provable formula, however, induce focused proofs of very different shapes.

Via annotations of proofs with proof terms, Miller and Wu study the shape of polarized proofs of *minimal* intuitionistic logic, that is, for the logic of implications, which is the basic setting of the Curry-Howard isomorphism. They consider the two natural uniform polarizations having either all atomic formulas polarized negatively, or positively (non-atomic formulas are all negative, because implication is negative).

¹ Email: beniamino.accattoli@inria.fr

² Email: jwu@lix.polytechnique.fr

Two very different term structures arise. The negative polarity assignment yields the usual λ -calculus. The positive polarity assignment, instead, yields a quite different syntax, with a restricted form of application and accounting for sharing via a notion of explicit substitution.

The Positive λ -Calculus. Based on this positively-polarized syntax, Wu introduced the *positive λ -calculus* λ_{pos} [40], a calculus with explicit substitutions endowed with call-by-value (shortened to CbV) evaluation, because substituting applications (as in call-by-name) would break the shape of positive terms.

The positive λ -calculus is an unusual calculus and the aim of this paper is to show its relevance. At first sight, it is yet another sharing-based presentations of CbV evaluation. It stands out, however, for a peculiar treatment of variables in relation to sharing, here dubbed *compactness*, that to our knowledge is new. The main contribution of this paper is to show that compactness considerably simplifies the definition and the study of *useful sharing*, a form of shared evaluation first introduced by Accattoli and Dal Lago [4] to study reasonable time cost models for the λ -calculus.

In order to properly explain the novelty of the positive λ -calculus, we are now going to outline three concepts: sharing-based presentations of CbV evaluation, the sharing of variables, and useful sharing.

700 Sharing-Based Presentations of CbV Evaluation. In the literature, sharing is an overloaded word. The most basic form of sharing is *sub-term sharing*, which can be obtained by simply adding to the standard syntax of the λ -calculus a *let* $x = u$ in t construct, which shares u between all the occurrences of x in t . The use of *let*-expressions is pervasive in presentations of CbV λ -calculi, typically in Moggi [29,30]. In *let* $x = u$ in t , it is usually assumed that u shall be evaluated before t , but such an assumption is often dropped when studying sharing. Additionally, *let* $x = u$ in t is often rather more concisely noted as an explicit substitution $t[x \leftarrow u]$ (in this paper, meta-level substitution is noted $t\{x \leftarrow u\}$).

In a CbV setting, having both explicit substitutions $t[x \leftarrow u]$ and applications tu is somewhat redundant, as explicit substitutions can be used to constrain the shape of applications. It is possible, indeed, to have only values v for one or both sub-terms of applications tu , that is, they can be constrained to be of shape vu (or tv , or even vv'). The idea is to apply a transformation $\llbracket \cdot \rrbracket$ turning tu into $(x \llbracket u \rrbracket)[x \leftarrow \llbracket t \rrbracket]$ with x fresh (or into $(\llbracket t \rrbracket x)[x \leftarrow \llbracket u \rrbracket]$, or $(xy)[x \leftarrow \llbracket t \rrbracket][y \leftarrow \llbracket u \rrbracket]$ with y fresh). It is typical of CbV (rather than call-by-name), because, for the restriction to be stable by reduction, it should be forbidden to substitute applications. Instances of CbV calculi with restrained applications abound, two notable examples being the calculus of *A-normal forms* by Sabry and Felleisen [32,22] and the *fine-grained* CbV calculus by Levy et al. [25]. Applications are also restricted in Sestoft's study of call-by-need [37], or that of Walker's on substructural type systems [38].

We shall follow Accattoli et al. [2] and refer to these decompositions of applications as to *crumbled* calculi. By tweaking the rewriting rules, one can also further restrict the immediate sub-terms of applications to be variables (rather than values). This gives rise to at least nine different CbV calculi, one with ordinary application tu and eight crumbled variants vu , xu , tv' , vv' , xv' , ty , vy , and xy . One can also decide at least (each choice doubling the number of possible presentations):

- *Nested vs flattened ES*: whether explicit substitutions can be nested (as in $t[x \leftarrow u][y \leftarrow r]$) or have to be flattened (as in $t[x \leftarrow u][y \leftarrow r]$);
- *Granularity of substitutions*: whether evaluation is small-step (substitution acts on all occurrences of a variable at once) or micro-step (variable occurrences are replaced one at a time);
- *Variables vs values*: whether variables are values, and thus can trigger substitution redexes, or not, that is, only abstractions can be substituted.

None of these choices affects the expressivity of the calculus, but the various calculi have different properties and degrees of manageability. In particular, the proof that a crumbled calculus is as expressive as the full one depends on the choices, and the proof might be non-trivial, as we shall see.

According to this classification, Wu's positive λ -calculus is a crumbled calculus with applications of shape xy (the minimalistic one), flattened substitutions, micro-step, and where variables are not values. It distinguishes itself from all these calculi, however, because it also has a new *compactness* feature that solves a pervasive issue of sub-term sharing.

Sharing of Variables and Compactness. In λ -calculi with sharing, usually variables can be shared, that is $t[x \leftarrow y]$ is a valid term. This leads to the possibility of having *renaming chains*, such as:

$$t[x_1 \leftarrow x_2][x_2 \leftarrow x_3] \dots [x_{n-1} \leftarrow x_n] \quad (1)$$

These chains are an issue because they lead to both space and time inefficiencies. Optimizations to prevent their creation are adopted for instance by Sands et al. [34], Wand [39], Friedman et al. [23], and Sestoft [37]. The first systematic study of renaming chains is by Accattoli and Sacerdoti Coen [12], with respect to time, where they show that it is enough to remove variables from values to avoid time inefficiencies related to renaming chains. Recently, Accattoli et al. have shown that the dynamic removal of renaming chains is essential for the only known reasonable notion of logarithmic space in the λ -calculus [5].

The new feature of the positive λ -calculus λ_{pos} is that *it does not share variables*, that is, $t[x \leftarrow y]$ does *not* belong to the syntax of λ_{pos} . This fact removes the issue of renaming chains altogether, with no need to design optimizations to prevent their creations, or removing variables from values, because renaming chains simply *cannot be expressed*.

In fact, λ_{pos} pushes things one step further, forging a sort of syntactical duality with respect to sharing:

- Variables cannot appear in explicit substitutions and, additionally, are the only constructors beside explicit substitutions;
- Dually, abstractions and applications appear in explicit substitutions but *not* out of them.

That is, x is a positive term but, for instance, xy and $\lambda x.x$ are not positive terms, only $z[z \leftarrow xy]$ and $z[z \leftarrow \lambda x.x]$ (or more generally $u[z \leftarrow xy]$ and $u[z \leftarrow \lambda x.x]$, where u is a positive term) are positive terms. This removes the further issue of whether xy and $z[z \leftarrow xy]$ are distinct terms, which is relevant in the study of CbV program equivalences, see Accattoli et al. [6]. One might argue that a similar approach to applications is already present in calculi related to the sequent calculus (in the style of Curien and Herbelin [19]), but to our knowledge the extension of this approach to abstractions is new, and relevant for useful sharing, see Sect. 5.

We dub *compactness* the fact that variables are not shared and are the only terms out of ESs. With compact sub-term sharing, terms have shape $x_1[x_1 \leftarrow u_1] \dots [x_n \leftarrow u_n]$, where each of the u_1, \dots, u_n is an abstraction or an application (which do not belong to the grammar of terms).

Useful Sharing. Our aim here is promoting the compactness of λ_{pos} by showing its impact on *useful sharing*, a concept apparently unrelated to focusing. Useful sharing is a technique introduced by Accattoli and Dal Lago [4], and then refined in call-by-value by Accattoli et al. [3], to study reasonable time cost models for λ -calculi. It works at the level of micro-step evaluation, that is, of replacements of single variable occurrences. The basic idea is that (in CbV) one should replace a variable occurrence with a copy of a shared abstraction only when it is useful to create β -redexes, that is, in the following case:

$$(xt)[x \leftarrow \lambda y.u] \rightarrow ((\lambda y.u)t)[x \leftarrow \lambda y.u] \quad (2)$$

While one should avoid replacements that do not create β -redexes, i.e. are not useful, as the following one:

$$(tx)[x \leftarrow \lambda y.u] \rightarrow (t(\lambda y.u))[x \leftarrow \lambda y.u] \quad (3)$$

Avoiding non-useful replacements leads to considerable speed-ups, that can even be *exponential* for some terms, in the case of strong evaluation (that is, when evaluation goes under abstraction) [4,3].

While the intuition behind useful sharing is easy to convey, the technical details are complex, because cases (2) and (3) are not the only possible ones, and various complications arise. A first simplified setting for useful sharing is given by crumbled λ -calculi where arguments can only be variables (that is, with applications of shape ty , vy , or xy), since then non-useful replacements such as those in (3) cannot be expressed and are then ruled out. This is already known and used by Accattoli et al. [3].

One of the complex aspects of useful sharing is related to renaming chains (as in (1)). They force the distinction between steps as in (2), which are *directly useful*, and steps over a renaming chain such as:

$$(x_1 t)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow x_n][x_n \leftarrow \lambda y.u] \rightarrow (x_1 t)[x_1 \leftarrow x_2] \dots [x_{n-1} \leftarrow \lambda y.u][x_n \leftarrow \lambda y.u] \quad (4)$$

These replacements are *indirectly useful*: they do not directly create a β -redex and yet they contribute to the *future* creation of β -redexes. Continuing with evaluation, indeed, $\lambda y.u$ shall replace the content of all the explicit substitutions in the chain and finally be substituted for x_1 , creating a β -redex. Indirectly useful steps cannot be avoided, otherwise some β -redexes are never created, and evaluation gets stuck.

Thanks to compactness, the positive λ -calculus λ_{pos} has no renaming chains and thus indirectly useful replacements are simply *ruled out*. Evaluation is not stuck in λ_{pos} , though, because indirectly useful steps somehow transform into directly useful steps in λ_{pos} , as explained in Sect. 6. Since λ_{pos} also has minimalistic applications of shape xy , non-useful replacements are ruled out as well. Therefore, λ_{pos} has only useful replacements, and only the *directly* useful ones (hence the title). Essentially, λ_{pos} captures the *core* of useful sharing, ruling out the technicalities.

Open CbV. Concretely, we develop our study with respect to Accattoli and Guerrieri’s framework of *Open CbV* [7], that is, we consider *weak evaluation* (i.e. not under abstraction) and *possibly open terms*. It is an intermediate setting between:

- (i) Weak evaluation and closed terms, where many subtleties of useful sharing are not observable, and
- (ii) Strong evaluation (which implies dealing with open terms), where further technicalities arise.

The same approach is followed by other works on useful sharing by Accattoli and co-authors [11,8,9]—see in particular the introduction of [9] for an extensive discussion of this choice.

Architecture of Our Study. After all these preliminary explanations, we can now explain our results. As a reference for a sharing-based λ -calculus for CbV, we take (a micro-step presentation of) Accattoli and Paolini’s *value substitution calculus* (VSC) [10], which is also the reference for the study of CbV reasonable time and useful sharing by Accattoli et al. [3]. The VSC has (possibly nested) explicit substitutions and ordinary applications of shape tu . We then define (our slight variant of) Wu’s positive λ -calculus λ_{pos} .

Intuitively, the paper is devoted to proving the equivalence of the Open λ_{pos} and the Open VSC. Often, the equivalence of two calculi is stated as a bisimulation property. For crumbled calculi ruling out some steps—as it is the case for the positive λ -calculus—this is not possible, because the ruled out steps cannot be simulated. A finer approach is needed.

We thus define a *core sub-calculus* of the Open VSC, which has only directly useful replacements, and prove a *factorization theorem* for the Open VSC, stating that every reduction sequence can be factored into a core one followed by a non-core one. Moreover, we provide a theorem ensuring that the Open VSC and the Core Open VSC are termination equivalent.

Next, we define a translation $\llbracket \cdot \rrbracket$ from the VSC to λ_{pos} and show that:

- (i) *Simulation*: it induces a simulation of the Core Open VSC by the Open λ_{pos} , and
- (ii) *Normal forms*: it sends core normal forms into appropriate normal forms of λ_{pos} .

From these properties, it follows that the Core Open VSC and the Open λ_{pos} are termination equivalent.

This proof technique is an original contribution of the paper. In particular, it is the first time that usefulness is justified via a factorization theorem.

Lastly, we show that the compactness of λ_{pos} induces a further relevant property: its open evaluation has the *diamond property*, a strong form of confluence, which is not true for the (micro-step) Open VSC.

Proofs. Proofs are in the technical report [13].

LANGUAGE		CONTEXTS	
TERMS	$t, u, r, q, p ::= v \mid tu \mid t[x \leftarrow u]$	SUBST.	$L, L' ::= \langle \cdot \rangle \mid L[x \leftarrow u]$
VALUES	$v, v' ::= x \mid \lambda x.t$	OPEN	$O, O' ::= \langle \cdot \rangle \mid Ot \mid tO \mid t[x \leftarrow O] \mid O[x \leftarrow u]$
ANSWERS	$a, a' ::= L\langle \lambda x.t \rangle$		
REDUCTION RULES			
MULTIPLICATIVE	$L\langle \lambda x.t \rangle u \mapsto_m L\langle t[x \leftarrow u] \rangle$		
EXPONENTIAL	$O\langle \langle x \rangle \rangle [x \leftarrow L\langle v \rangle] \mapsto_e L\langle O\langle \langle v \rangle \rangle [x \leftarrow v] \rangle$		CONTEXTUAL CLOSURE
GARBAGE COLL.	$t[x \leftarrow L\langle v \rangle] \mapsto_{gc} L\langle t \rangle \quad \text{if } x \notin \text{fv}(t)$		$\frac{t \mapsto_a t' \quad a \in \{\text{m}, \text{e}, \text{gc}\}}{O\langle t \rangle \mapsto_{oa} O\langle t' \rangle}$
NOTATION	$\rightarrow_{\text{ovsc}} := \rightarrow_{\text{om}} \cup \rightarrow_{\text{oe}} \cup \rightarrow_{\text{ogc}}$		

Fig. 1. The Open (Micro-Step) Value Substitution Calculus λ_{ovsc} .

2 Preliminaries: Rewriting Notions and Notations

Given a rewriting relation \rightarrow_r , we write $d: t \rightarrow_r^* u$ for a \rightarrow_r -reduction sequence from t to u , the length of which is noted $|d|$. Moreover, we use $|d|_a$ for the number of a -steps in d , for a sub-relation \rightarrow_a of \rightarrow_r .

A term t is *weakly r-normalizing* if $d: t \rightarrow_r^* u$ with u r -normal; and t is *strongly r-normalizing* if there are no diverging r -sequences from t , or, equivalently, if all its r -reducts are strongly r -normalizing.

Given two reductions \rightarrow_1 and \rightarrow_2 we use $\rightarrow_1 \rightarrow_2$ for their composition, defined as $t \rightarrow_1 \rightarrow_2 u$ if $t \rightarrow_1 r \rightarrow_2 u$ for some r . We also write, e.g., $\rightarrow_1 \rightarrow_2 \subseteq \rightarrow_3^*$ to state that $t \rightarrow_1 \rightarrow_2 u$ implies $t \rightarrow_3^* u$.

Diamond Property. According to Dal Lago and Martini [20], a relation \rightarrow_r is *diamond* if $u_1 r \leftarrow t \rightarrow_r u_2$ and $u_1 \neq u_2$ imply $u_1 \rightarrow_r r r \leftarrow u_2$ for some r . If \rightarrow_r is diamond then:

- (i) *Confluence:* \rightarrow_r is confluent, that is, $u_1 r \leftarrow t \rightarrow_r^* u_2$ implies $u_1 \rightarrow_r^* r r \leftarrow u_2$ for some r ;
- (ii) *Length invariance:* all r -reduction sequences to normal form with the same start term have the same length (i.e. if $d: t \rightarrow_r^* u$ and $d': t \rightarrow_r^* u$ with $u \rightarrow_r$ -normal then $|d| = |d'|$);
- (iii) *Uniform normalization:* t is weakly r -normalizing if and only if it is strongly r -normalizing.

Basically, the diamond property captures a more liberal form of determinism.

3 The (Micro-Step) Open Value Substitution Calculus

In this section, we present our system of reference for Open CbV, Accattoli and Paolini's *value substitution calculus* [10] (shortened to VSC). We shall adopt a micro-step presentation (explained below) of the open fragment, that is, of weak evaluation on possibly open terms. This variant is defined in Fig. 1 and shall be denoted with λ_{ovsc} . We also recall some of its basic properties, which shall either be used in the next section or used to compare λ_{ovsc} with the positive λ -calculus.

Terms. The VSC is a CbV λ -calculus extended with let-expressions, similarly to Moggi's CbV calculus [29,30]. We do however write a let-expression $\text{let } x = u \text{ in } t$ as a more compact *explicit substitution* $t[x \leftarrow u]$ (ES for short), which binds x in t . Moreover, our let/ES does not fix an order of evaluation between t and u , in contrast to many papers in the literature (e.g. Sabry and Wadler [33] or Levy et al. [25]) where u is evaluated first. Intuitively, $t[x \leftarrow u]$ is a construct for *sub-term sharing*, as for instance is evident when comparing tt and $(xx)[x \leftarrow t]$, where t is shared in the latter.

The set of free variables of a term t is denoted by $\text{fv}(t)$ and terms are identified up to α -renaming. We use $t\{x \leftarrow u\}$ for the capture-avoiding substitution of t for each free occurrence of x in t .

Contexts. We shall use many notions of *contexts*, *i.e.* terms with a hole noted $\langle \cdot \rangle$. The most general contexts used here are open contexts O , for which the hole cannot appear under abstraction and where the adjective *open* means *possibly open* (that is, possibly with occurrences of free variables), and not *necessarily open*. We shall also extensively use *substitution contexts* L , which are lists of ESs (L stands for *List*). Plugging a term t in a context O is noted $O\langle t \rangle$ and can capture variables, for instance $(\lambda x.\lambda y.\langle \cdot \rangle)\langle xy \rangle = \lambda x.\lambda y.xy$ (while $(\lambda x.\lambda y.z)\{z \leftarrow xy\} = \lambda x'.\lambda y'.xy$); we use $O\langle\langle t \rangle\rangle$ when we want to prevent it.

Substitution contexts are in particular used to define *answers*, which are abstractions surrounded by a list of substitutions. Answers shall play a key role in relating λ_{ovsc} with the positive λ -calculus in Sect. 7.

Reduction Rules. The reduction rules of VSC are slightly unusual as they use *contexts* both to allow one to reduce redexes located in sub-terms, which is standard, *and* to define the redexes themselves, which is less standard—this kind of rules is called *at a distance*. The rewriting rules at a distance of λ_{ovsc} are related to cut-elimination on proof nets, via the CbV translation $(A \Rightarrow B)^v = !(A^v \multimap B^v)$ of intuitionistic logic into linear logic, see Accattoli [1]. The linear logic connection is also the reason behind the *multiplicative* and *exponential* terminology for the rewriting rules.

The multiplicative root rule \mapsto_m turns a β -redex (at a distance, because of the substitution context L) into an ES. Note that there is no *by-value* restriction, rule \mapsto_m can fire also if the argument is not a value. The by-value restriction is part of rules \mapsto_e and \mapsto_{gc} which take care of the substitution process.

The VSC usually appears with a single *small-step* substitution / exponential rule, where small-step refers to the fact that it is based on meta-level substitution $t\{x \leftarrow v\}$, which replaces all the occurrences of the bound variable x at once. Here, we adopt a *micro-step* presentation: the exponential rule \mapsto_e replaces a *single* variable occurrence at a time, and there is an additional garbage collection rule \mapsto_{gc} for removing ESs bounding variables with no occurrences. These two rules can duplicate / erase only values. Both root rules are *at a distance* in that they involve a substitution context L , which is not duplicated / erased.

The rewriting relation $\rightarrow_{\text{ovsc}}$ of λ_{ovsc} is obtained by closing the root rewriting rules \mapsto_m , \mapsto_e , and \mapsto_{gc} by open contexts O and by taking the union of the obtained rules.

Confluence and Lack of Diamond. The VSC is confluent and the open small-step VSC even has the *diamond property* (defined in Sect. 2), see Accattoli and Paolini [10]. In the literature, there is no proof that the variant λ_{ovsc} used here is confluent, but this can be easily proved by adjusting the proof in [10]. We omit the details since the proof is absolutely standard. For the study in this paper, instead, it is worth pointing out that λ_{ovsc} is *not* diamond, as the following diagram shows:

$$\begin{array}{ccc}
 O\langle\langle x \rangle\rangle[x \leftarrow y][y \leftarrow v] & \xrightarrow{\text{oe}} & O\langle\langle x \rangle\rangle[x \leftarrow v][y \leftarrow v] \\
 \text{oe} \downarrow & & \downarrow \text{oe} \\
 O\langle\langle y \rangle\rangle[x \leftarrow y][y \leftarrow v] & \cdots \xrightarrow{\text{oe}} & O\langle\langle y \rangle\rangle[x \leftarrow v][y \leftarrow v] \cdots \xrightarrow{\text{oe}} & O\langle\langle v \rangle\rangle[x \leftarrow v][y \leftarrow v]
 \end{array}$$

Postponement of Garbage Collection. A typical property of micro-step λ -calculi at a distance is the possibility of postponing garbage collection (GC). In our setting, the postponement preserves both the number of non-GC steps (which is standard) and the number of GC steps (which is not always the case) because GC steps cannot be duplicated in call-by-value weak evaluation (since only values are duplicated, but GC steps cannot take place inside values because of weakness). As it is standard, the (global) postponement of GC is obtained by iterating a local form of postponement. Notation: $\rightarrow_{\text{o-gc}} := \rightarrow_{\text{om}} \cup \rightarrow_{\text{oe}}$.

Proposition 3.1 (Local postponement of garbage collection) *For $a \in \{\text{m}, \text{e}\}$, If $t \rightarrow_{\text{ogc}} \rightarrow_{\text{oa}} u$, then $t \rightarrow_{\text{oa}} \rightarrow_{\text{ogc}} u$.*

Proposition 3.2 (Postponement of garbage collection) *If $d : t \rightarrow_{\circ}^* u$, then there exist reduction sequences $e : t \rightarrow_{\text{o-gc}}^* u'$ and $f : u' \rightarrow_{\text{ogc}}^* u$ with $|e|_{\text{om}} = |d|_{\text{om}}$, $|e|_{\text{oe}} = |d|_{\text{oe}}$, and $|f| = |d|_{\text{ogc}}$.*

Local Termination. As it is often the case when β -reduction is decomposed into smaller rules, every single rule of λ_{ovsc} is strongly normalizing separately (it is only together that they may diverge, namely $\rightarrow_{\text{o-gc}}$ diverges on some terms).

TERMS	$t, u, r ::= x \mid t[x \leftarrow yz] \mid t[x \leftarrow \lambda y.u]$
EVALUATION CTXS	$E ::= \langle \cdot \rangle \mid E[x \leftarrow yz] \mid E[x \leftarrow \lambda y.t]$
ROOT REDUCTION RULES	
$E\langle t[x \leftarrow yz] \rangle [y \leftarrow \lambda w.E'\langle w' \rangle] \mapsto_{\text{eme}_+} E\langle E'\langle t\{x \leftarrow w'\} \{w \leftarrow z\} \rangle \rangle [y \leftarrow \lambda w.E'\langle w' \rangle]$ $t[x \leftarrow \lambda y.u] \mapsto_{\text{gc}_+} t \quad \text{if } x \notin \text{fv}(t)$	
NOTATION	$\rightarrow_{\text{o}_+} := \rightarrow_{\text{oeme}_+} \cup \rightarrow_{\text{ogc}_+}$
CTX CLOSURE	$\frac{t \mapsto_a t'}{E\langle t \rangle \rightarrow_{\text{oa}} E\langle t' \rangle} \quad a \in \{\text{eme}_+, \text{gc}_+\}$

Fig. 2. The open positive λ -calculus λ_{pos} .

Proposition 3.3 (Local termination) *Let $a \in \{\text{m}, \text{e}, \text{gc}\}$. Relation \rightarrow_{oa} is strongly normalizing. Moreover, $\rightarrow_{\text{oe}} \cup \rightarrow_{\text{ogc}}$ is strongly normalizing.*

Renaming Chains. In λ_{ovsc} , there can be renaming chains such as $t[x_1 \leftarrow x_2][x_2 \leftarrow x_3] \dots [x_{n-1} \leftarrow x_n]$. The issue with these chains is that some terms dynamically create longer and longer chains, slowing down the evaluation process. The simplest term on which the issue can be observed is the looping combinator Ω :

$$\begin{aligned}
\Omega = & \quad (\lambda x.xx)(\lambda x.xx) && \rightarrow_{\text{om}} (x_1x_1)[x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{oe}} ((\lambda x.xx)x_1)[x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{om}} (x_2x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{oe}} (x_1x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{oe}} ((\lambda x.xx)x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{om}} (x_3x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{oe}} (x_2x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{oe}} (x_1x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{oe}} ((\lambda x.xx)x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \dots
\end{aligned}$$

Note that after each \rightarrow_{om} step, evaluation does a sequence of \rightarrow_{oe} steps having length equal to the number of preceding \rightarrow_{om} steps. Easy calculations show that the number of \rightarrow_{oe} steps is then *quadratic* in the number of \rightarrow_{om} steps, for these sequences. This quadratic overhead was first pointed out and studied by Accattoli and Sacerdoti Coen [12]. They show that it is enough to remove variables from values (as it is done in most implementative studies, but usually without an explanation for this choice), in order to remove this issue, since evaluation then rather proceeds as follows:

$$\begin{aligned}
\Omega = & \quad (\lambda x.xx)(\lambda x.xx) && \rightarrow_{\text{om}} (x_1x_1)[x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{oe}} ((\lambda x.xx)x_1)[x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{om}} (x_2x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{oe}} (x_2x_2)[x_2 \leftarrow \lambda x.xx][x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{oe}} ((\lambda x.xx)x_2)[x_2 \leftarrow \lambda x.xx][x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{om}} (x_3x_3)[x_3 \leftarrow x_2][x_2 \leftarrow \lambda x.xx][x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{oe}} (x_3x_3)[x_3 \leftarrow \lambda x.xx][x_2 \leftarrow \lambda x.xx][x_1 \leftarrow \lambda x.xx] \\
& \rightarrow_{\text{oe}} ((\lambda x.xx)x_3)[x_3 \leftarrow \lambda x.xx][x_2 \leftarrow \lambda x.xx][x_1 \leftarrow \lambda x.xx] && \rightarrow_{\text{om}} \dots
\end{aligned}$$

And it is easily seen that the number of \rightarrow_{oe} steps is now *linear* in the number of \rightarrow_{om} steps. The positive λ -calculus of the next section shall subsume this approach, by forbidding altogether ESs containing a variable, thus also removing the ambiguity of whether variables are values or not.

4 The (Explicit) Open Positive λ -Calculus

Here, we present the open fragment λ_{opos} of Wu's positive λ -calculus λ_{pos} , and then slightly refine it into an *explicit* variant λ_{oxpos} . The definition of λ_{opos} is in Fig. 2. Terms of λ_{pos} are a subset of VSC terms.

Terms. In λ_{opos} , as in the λ -calculus, there are only three constructors, variables, applications, and

$$\boxed{
\begin{array}{c}
\frac{}{\Gamma, x : A \vdash x : A} \textit{var} \quad \frac{\Gamma, y : B \Rightarrow C, z : B, x : C \vdash t : A}{\Gamma, y : B \Rightarrow C, z : B \vdash t[x \leftarrow yz] : A} \textit{app} \\
\frac{\Gamma, y : B \vdash u : C \quad \Gamma, x : B \Rightarrow C \vdash t : A}{\Gamma \vdash t[x \leftarrow \lambda y.u] : A} \textit{abs}
\end{array}
}$$

Fig. 3. Assignment of simple types to positive λ -terms.

abstractions. There are however, various differences, namely:

- Applications are only between variables, that is, of the shape yz ;
- Applications and abstractions are always shared, that is, standalone applications and abstractions are not allowed in the grammar. They can only be introduced by the explicit substitution constructs $[x \leftarrow yz]$ and $[x \leftarrow \lambda y.u]$;
- Positive sharing is peculiar as positive terms are *not* shared in general, that is, $t[x \leftarrow u]$ is not a positive term. There are only two distinct sharing constructors for applications and abstractions, but no construct for sharing variables or applications/abstractions with top-level sharing;

Types. We are not going to work with types, and yet in Fig. 3 we show how simple types can be assigned to positive terms. The main point is to suggest that the *positive focusing* aspect, roughly, amounts to have only rules that act on the *left* of the deduction symbol \vdash , leaving the right hand type A unchanged. Note that this is dual to what happens in the standard system of simple types for the λ -calculus (*i.e.* natural deduction), which correspond to the negative polarity assignment, where rules only act on the *right* of \vdash .

Contexts. The notions of substitution contexts L and open contexts O of the λ_{ovsc} pleasantly collapse onto a single notion of *evaluation contexts* E in λ_{opos} , because the additional clauses for O in λ_{ovsc} (for the sub-terms of applications and inside ESs) do not make sense in λ_{opos} , due to the restricted shape of terms. As it is immediately seen from the grammar of positive terms, every positive term t can be written uniquely as $E\langle x \rangle$ for some x and E , with E possibly capturing x .

Rewriting Rules. There are two rewriting rules at a distance, based on open contexts and defined in Fig. 2. Rule $\rightarrow_{\text{ogc}_+}$ handles garbage collection and is standard. Rule $\rightarrow_{\text{ome}_+}$ is quite heavy, essentially it is a macro reduction step concatenating two steps of the VSC plus a meta-level substitution. Let us explain it (it is not necessary to grasp it completely, since right after we shall introduce a slight variant of the positive calculus that has simpler rewriting rules). Rule $\rightarrow_{\text{ome}_+}$ does three operations, and also adopts some notations to respect the constrained shape of terms:

- Useful exponential step:* it does the useful³ replacement of an applied occurrence of y , *i.e.* it acts on y on $E\langle t[x \leftarrow yz] \rangle$, by an abstraction $\lambda w.u$.
- Created multiplicative step:* since $t[x \leftarrow (\lambda w.u)z]$ is not a construct of λ_{pos} , the rule has to also do on-the-fly what in the VSC is a multiplicative step, which would create the ES $u[w \leftarrow z]$.
- Further meta-level substitution:* but since $u[w \leftarrow z]$ is not a construct of λ_{pos} either, the step also does on-the-fly the meta-level substitution associated to $[w \leftarrow z]$.
- Respecting the syntax:* to write the reduct, one needs to respect the constrained shape of positive terms, which requires to write u as $E'\langle w' \rangle$ and then do the re-arrangement of the term structure and the meta-level substitutions of variables specified by the rule in Fig. 2.

Working with such a rule is heavy. A first reason is that, quite simply, it involves a lot of symbols. Another reason is that, by concatenating steps of different nature of the VSC, λ_{opos} is *not* conservative over λ_{ovsc}

³ Useful steps shall be defined and studied in the next section. Here we rest on the intuitive description given in the introduction.

TERMS	$t, u, r ::= x \mid t[x \leftarrow yz] \mid t[x \leftarrow \lambda y.u] \mid [x \leftarrow (\lambda y.u)z]$
EVALUATION CTXS	$E ::= \langle \cdot \rangle \mid E[x \leftarrow yz] \mid E[x \leftarrow \lambda y.t] \mid E[x \leftarrow (\lambda y.t)z]$
ROOT REDUCTION RULES	
	$t[x \leftarrow (\lambda y.E\langle z \rangle)w] \mapsto_{m_+} E\langle t\{x \leftarrow z\} \rangle\{y \leftarrow w\}$
	$E\langle t[x \leftarrow yz] \rangle[y \leftarrow \lambda w.u] \mapsto_{e_+} E\langle t[x \leftarrow (\lambda w.u)z] \rangle[y \leftarrow \lambda w.u]$
	$t[x \leftarrow \lambda y.u] \mapsto_{gc_+} t \quad \text{if } x \notin \text{fv}(t)$
NOTATION	$\rightarrow_{ox_+} := \rightarrow_{om_+} \cup \rightarrow_{oe_+} \cup \rightarrow_{ogc_+} \quad \left \text{CTX CLOSURE } \frac{t \mapsto_a t'}{E\langle t \rangle \rightarrow_{oa} E\langle t' \rangle} \quad a \in \{m_+, e_+, gc_+\}$

Fig. 4. The open explicit positive λ -calculus λ_{oxpos} .

in the sense that it breaks one of its key property, namely local termination (stated by Prop. 3.3). Rule $\rightarrow_{\text{ome}_+}$ can indeed diverge by itself, as in the following representation of the looping term Ω in λ_{opos} :

$$x[x \leftarrow yy][y \leftarrow \lambda z.w[w \leftarrow zz]] \rightarrow_{\text{ome}_+} x[x \leftarrow yy][y \leftarrow \lambda z.w[w \leftarrow zz]] \rightarrow_{\text{ome}_+} \dots$$

We are then going to decompose $\rightarrow_{\text{ome}_+}$ into two rules, by adding a new constructor to the calculus.

The Explicit Positive λ -Calculus. The explicit positive calculus λ_{oxpos} is defined in Fig. 4. The idea is to extend λ_{opos} by adding the intermediate construct $t[x \leftarrow (\lambda w.u)z]$ which allows us to decompose rule \mapsto_{eme_+} in two, separating the first operation, the useful exponential step, now noted \mapsto_{e_+} , from the rest of the rule, now noted \mapsto_{m_+} . Rule \mapsto_{m_+} still corresponds to two actions (the multiplicative step of the VSC and the meta-level substitution of the variable), but it is one of the key points of λ_{opos} that variable substitutions disappear, so we shall not decompose the rule further. Intuitively, the new construct $t[x \leftarrow (\lambda w.u)z]$ plays a similar (but dual) role to the introduction of ESs in the λ -calculus to decompose β -redexes in two, as it is an explicit β -redex. This is why we refer to the obtained calculus as to the *explicit positive calculus*. One might also see it as switching the shape of (compact) applications of λ_{pos} from xy to vy .

Clearly, λ_{oxpos} simulates λ_{opos} : if $t \rightarrow_{\text{ome}_+} u$ then $t \rightarrow_{\text{oe}_+} \rightarrow_{\text{om}_+} u$.

Proposition 4.1 λ_{oxpos} simulates λ_{opos} .

Diamond. A difference between λ_{ovsc} and λ_{oxpos} is that the latter is diamond, as we now show, while the former is not (see Sect. 3). The proof uses the following basic lemma.

Lemma 4.2 (Stability under renamings) Let t and u be λ_{oxpos} terms. If $t \rightarrow_{\text{ox}_+} u$ then $t\{x \leftarrow y\} \rightarrow_{\text{ox}_+} u\{x \leftarrow y\}$ for any x and y .

Theorem 4.3 (Positive diamond) Relation $\rightarrow_{\text{ox}_+}$ is diamond.

Postponement of GC and Local Termination. In terms of properties, λ_{oxpos} is conservative with respect to λ_{ovsc} , as postponement of GC and local termination still hold. Notation: $\rightarrow_{\text{ox}_+ - \text{gc}} := \rightarrow_{\text{om}_+} \cup \rightarrow_{\text{oe}_+}$.

Proposition 4.4 (Local postponement of garbage collection) Let t and u be λ_{oxpos} terms and $a \in \{m_+, e_+\}$. If $t \rightarrow_{\text{ogc}_+} \rightarrow_{\text{oa}} u$, then $t \rightarrow_{\text{oa}} \rightarrow_{\text{ogc}_+} u$.

Proposition 4.5 (Postponement of garbage collection) Let t and u be λ_{oxpos} terms, $d : t \rightarrow_{\text{ox}_+}^* u$. Then there exist reduction sequences $e : t \rightarrow_{\text{ox}_+ - \text{gc}}^* u'$ and $f : u' \rightarrow_{\text{ogc}_+}^* u$ with $|e|_{\text{om}_+} = |d|_{\text{om}_+}$, $|e|_{\text{oe}_+} = |d|_{\text{oe}_+}$, and $|f| = |d|_{\text{ogc}_+}$.

Proposition 4.6 (Local termination) Let $a \in \{m_+, e_+, gc_+\}$. Relation \rightarrow_{oa} is strongly normalizing. Moreover, $\rightarrow_{\text{oe}_+} \cup \rightarrow_{\text{ogc}_+}$ is strongly normalizing.

EXP. ROOT RULE FOR ABSTRACTIONS	$O\langle\langle x \rangle\rangle[x \leftarrow L\langle\lambda y.t\rangle] \mapsto_{e_{\text{abs}}} L\langle O\langle\langle\lambda y.t\rangle\rangle[x \leftarrow \lambda y.t]\rangle$
EXP. ROOT RULE FOR VARIABLES	$O\langle\langle x \rangle\rangle[x \leftarrow L\langle y \rangle] \mapsto_{e_{\text{var}}} L\langle O\langle\langle y \rangle\rangle[x \leftarrow y]\rangle$
GC ROOT RULE FOR ABSTRACTIONS	$t[x \leftarrow L\langle\lambda y.u\rangle] \mapsto_{g_{\text{C}_{\text{abs}}}} L\langle t \rangle$ if $x \notin \text{fv}(t)$
GC ROOT RULE FOR VARIABLES	$t[x \leftarrow L\langle y \rangle] \mapsto_{g_{\text{C}_{\text{var}}}} L\langle t \rangle$ if $x \notin \text{fv}(t)$
CTX CLOSURE	$\frac{t \mapsto_a t'}{E\langle t \rangle \rightarrow_{\text{oa}} E\langle t' \rangle} \quad a \in \{e_{\text{abs}}, e_{\text{var}}, g_{\text{C}_{\text{abs}}}, g_{\text{C}_{\text{var}}}\}$

Fig. 5. Dissected rewriting rules for λ_{ovsc} .

Absence of Renaming Chains. At the end of Sect. 3, we discussed renaming chains and their dynamic creations by the looping term Ω . The following term is the representation of Ω in λ_{oxpos} , together with its evaluation, where evidently the number of $\rightarrow_{\text{oe}_+}$ steps is linear in the number of $\rightarrow_{\text{om}_+}$ steps:

$$\begin{aligned} & w[w \leftarrow (\lambda x.y[y \leftarrow xx])z][z \leftarrow \lambda x.y[y \leftarrow xx]] \rightarrow_{\text{om}_+} w[w \leftarrow zz][z \leftarrow \lambda x.y[y \leftarrow xx]] \\ \rightarrow_{\text{oe}_+} & w[w \leftarrow (\lambda x.y[y \leftarrow xx])z][z \leftarrow \lambda x.y[y \leftarrow xx]] \rightarrow_{\text{om}_+} w[w \leftarrow zz][z \leftarrow \lambda x.y[y \leftarrow xx]] \\ \rightarrow_{\text{oe}_+} & w[w \leftarrow (\lambda x.y[y \leftarrow xx])z][z \leftarrow \lambda x.y[y \leftarrow xx]] \dots \end{aligned}$$

5 Dissecting λ_{ovsc} : Variable and Useful Steps

In this section, we isolate various sub-reductions of λ_{ovsc} in order to relate λ_{ovsc} and λ_{oxpos} in the next section. Essentially, some steps of λ_{ovsc} cannot be expressed in λ_{oxpos} , and some are instead *absorbed*, that is, mapped to identities rather than being simulated. We then have to partition the rewriting rules of λ_{ovsc} into sub-rules as to identify the steps that cannot be expressed, those that are absorbed, and those that are simulated by λ_{oxpos} . In particular, the partition of steps leads us to discuss useful sharing.

Variable Substitutions. In λ_{oxpos} , there is no way to represent an ES $t[x \leftarrow y]$ containing a variable, and there is also no way of simulating a variable exponential step such as $O\langle\langle x \rangle\rangle[x \leftarrow y] \rightarrow_e O\langle\langle y \rangle\rangle[x \leftarrow y]$ or a variable GC step $t[x \leftarrow y] \rightarrow_e t$ with $x \notin \text{fv}(t)$. These steps shall be *absorbed* by our translation from λ_{ovsc} to λ_{oxpos} , that is, they shall be mapped to identities. To properly state this fact later on, we split now the root exponential rule \mapsto_e in two rules $\mapsto_{e_{\text{abs}}}$ and $\mapsto_{e_{\text{var}}}$, depending on whether the replacing value is an abstraction or a variable, and similarly for GC. The split rules are defined in Fig. 5.

Of two sub-rules $\mapsto_{e_{\text{abs}}}$ and $\mapsto_{g_{\text{C}_{\text{abs}}}}$ that are not absorbed, $\mapsto_{g_{\text{C}_{\text{abs}}}}$ shall be closed by all open context and simply factored-out via the postponement of GC (Prop. 3.2). The other sub-rule $\mapsto_{e_{\text{abs}}}$ is where usefulness plays a role, discussed next.

Useful Steps, First Intuitions. Another difference between the two calculi is that in λ_{oxpos} exponential steps can only be directly *useful*. Let us overview usefulness in a bit more detail than in the introduction. Replacements of variable occurrences out of ESs by abstractions amount to three *direct* cases, in the λ_{ovsc} :

$$\begin{aligned} \text{DIRECTLY USEFUL} & \quad (xt)[x \leftarrow \lambda y.u] \mapsto_{e_{\text{abs}}} ((\lambda y.u)t)[x \leftarrow \lambda y.u] \\ \text{DIRECTLY NON-USEFUL 1} & \quad (tx)[x \leftarrow \lambda y.u] \mapsto_{e_{\text{abs}}} (t(\lambda y.u))[x \leftarrow \lambda y.u] \\ \text{DIRECTLY NON-USEFUL 2} & \quad x[x \leftarrow \lambda y.u] \mapsto_{e_{\text{abs}}} (\lambda y.u)[x \leftarrow \lambda y.u] \end{aligned} \tag{5}$$

The terminology *useful* refers to the fact that the exponential step *creates* a new multiplicative redex, namely $(\lambda y.u)t$, and it is used to contrast with *non-useful*, or *useless* exponential steps (we shall prefer *non-useful* in this paper, as to denote them concisely with the letter n , given that *useful* and *useless* both start with u) that instead do not create multiplicative steps.

The reason why one considers usefulness with respect to multiplicative steps is that the number of multiplicative steps is a reasonable cost model in CbV [11,8,3], and that this fact is proved by an evaluation strategy that crucially avoids non-useful substitution steps, since non-useful substitution steps can at times add an exponential overhead, breaking the reasonability of the cost model.

In λ_{oxpos} , directly useful steps can be simulated, while the two kinds of direct non-useful step cannot be expressed. The first kind because of the shape of positive applications, which can only have a variable as an argument. The second kind because abstractions cannot appear out of ESs in λ_{oxpos} (this is the relevance of the second aspect of the compactness of $\lambda_{\text{pos}}/\lambda_{\text{oxpos}}$ mentioned in the introduction).

There are (at least) two technical difficulties in defining the useful steps of λ_{ovsc} precisely. Before diving into them, we provide a *disclaimer*. Useful sharing is complex and takes different shapes in different settings (i.e. call-by-name/value/need). For this reason, the only two works distinguishing into useful and non-useful steps, one in call-by-name [4] and one in CbV [11], rest on slightly different definitions concerning indirectly (non-)useful steps (defined below). On purpose, later papers avoid these definitions, despite implementing useful sharing [8,3,9]. Here, we are going to define the two kinds of steps, but our definitions (and terminology) shall—once more—be slightly different from [4,11]. The only thing on which all these works agree are the notions of direct useful steps, which is what λ_{oxpos} captures.

Difficulty 1: Indirect Useful Steps. In λ_{ovsc} , there are also *indirect* cases to consider, given by when there is a renaming chain connecting the acting ESs and the end variable occurrence. Consider for instance the following three indirect cases, mimicking the direct ones above via a renaming chain of length 1:

$$\begin{array}{lll}
\text{INDIRECTLY USEFUL} & (xt)[x\leftarrow z][z\leftarrow \lambda y.u] \mapsto_{\text{e}_{\text{abs}}} (xt)[x\leftarrow \lambda y.u][z\leftarrow \lambda y.u] \\
\text{INDIRECTLY NON-USEFUL 1} & (tx)[x\leftarrow z][z\leftarrow \lambda y.u] \mapsto_{\text{e}_{\text{abs}}} (tx)[x\leftarrow \lambda y.u][z\leftarrow \lambda y.u] \\
\text{INDIRECTLY NON-USEFUL 2} & x[x\leftarrow z][z\leftarrow \lambda y.u] \mapsto_{\text{e}_{\text{abs}}} x[x\leftarrow \lambda y.u][z\leftarrow \lambda y.u]
\end{array} \tag{6}$$

The first step does not create a multiplicative redex, but it is usually considered as useful because it contributes anyway to the *future* creation of the multiplicative redex that shall happen after that step. Similarly, the other two cases are usually considered as non-useful. This indirect aspect is quite difficult to work with. In λ_{oxpos} , the indirect phenomenon disappears, because ESs such as $[x\leftarrow z]$, which are the cause of the indirection, do not exist. This is a considerable improvement.

It remains, however, the issue of relating indirectly useful steps in λ_{ovsc} with (directly useful) steps in λ_{oxpos} . We somewhat circumvent this issue by departing from the literature and considering indirect useful steps *as non-useful*. Therefore, *all the indirect cases above are non-useful*, in this paper. This is not cheating, as we shall explain, it is related to the core factorization theorem of the next section.

For the sake of completeness, one should also consider the replacements in which the end variable is inside an ES. The first two cases of both (5) and (6) are unaffected. The third one instead amounts to extend a chain, and then can become any of the three cases in (6).

Difficulty 2: Contextual Closure. The other difficulty is that the directly useful steps of λ_{ovsc} cannot be defined at the root level and then closed by evaluation contexts, since sometimes the useful aspect is contributed by the evaluation context itself. Consider the following root step:

$$x[x\leftarrow \lambda y.u] \mapsto_{\text{e}_{\text{abs}}} (\lambda y.u)[x\leftarrow \lambda y.u]$$

As a root step it is not useful. But when plugged in the evaluation context $O = \langle \cdot \rangle t$, it gives rise to the following useful step, since now the steps creates a multiplicative redex in the reduct, because of the definition at a distance of these redexes:

$$x[x\leftarrow \lambda y.u]t \mapsto_{\text{e}_{\text{abs}}} (\lambda y.u)[x\leftarrow \lambda y.u]t$$

Therefore, usefulness of a $\rightarrow_{\text{oe}_{\text{abs}}}$ step depends also on the evaluation contexts surrounding the root step.

Useful Contexts and Steps. Useful exponential steps are defined via useful contexts by putting together

USEFUL EXP. RULE	$O_1 \langle O_2 \langle x \rangle [x \leftarrow L \langle \lambda y.t \rangle] \rangle \rightarrow_{\text{oeu}} O_1 \langle L \langle O_2 \langle \lambda y.t \rangle [x \leftarrow \lambda y.t] \rangle \rangle$ if $\text{usef}(O_1 \langle O_2 \rangle)$
NON-USEFUL EXP. RULE	$O_1 \langle O_2 \langle x \rangle [x \leftarrow L \langle \lambda y.t \rangle] \rangle \rightarrow_{\text{oenu}} O_1 \langle L \langle O_2 \langle \lambda y.t \rangle [x \leftarrow \lambda y.t] \rangle \rangle$ if $\text{nusef}(O_1 \langle O_2 \rangle)$

Fig. 6. Definition of useful and non-useful exponential variants of $\rightarrow_{\text{oeabs}}$, based on Lemma 5.4.1.

the context used to isolate the replaced variable and the surrounding evaluation context. We also define non-useful contexts N , whose first two clauses cover the two direct cases in (5) and whose third clause cover the three indirect cases in (6).

Definition 5.1 Useful and non-useful contexts of λ_{ovsc} are defined as follows:

$$\text{USEFUL CTXS } U ::= O \langle Lt \rangle \quad \text{NON-USEFUL CTXS } N ::= L \mid O \langle tL \rangle \mid O \langle t[x \leftarrow L] \rangle$$

When taking into account the evaluation context, an $\rightarrow_{\text{oeabs}}$ step has the following shape:

$$O_1 \langle O_2 \langle x \rangle [x \leftarrow L \langle \lambda y.t \rangle] \rangle \rightarrow_{\text{oeabs}} O_1 \langle L \langle O_2 \langle \lambda y.t \rangle [x \leftarrow \lambda y.t] \rangle \rangle$$

Such a $\rightarrow_{\text{oeabs}}$ step is *useful* if $O_1 \langle O_2 [x \leftarrow L \langle \lambda y.t \rangle] \rangle$ is a useful context, and *non-useful* otherwise.

Properties of Useful Contexts. In symbols, we use predicates usef and nusef : $\text{usef}(O)$ means that O is useful while $\text{nusef}(O)$ means that O is non-useful. Similarly, we use the predicate sub (resp. nsub) for substitution contexts (resp. non-substitution contexts).

The following immediate lemma states how useful contexts depend on their sub-contexts, stressing that the only subtle case is the first one.

Lemma 5.2 (Useful sub-contexts)

- (i) $\text{usef}(Ot) \Leftrightarrow \text{usef}(O) \vee \text{sub}(O)$.
- (ii) $\text{usef}(tO) \Leftrightarrow \text{usef}(O)$.
- (iii) $\text{usef}(O[x \leftarrow t]) \Leftrightarrow \text{usef}(O)$.
- (iv) $\text{usef}(t[x \leftarrow O]) \Leftrightarrow \text{usef}(O)$.

As a sanity check, we show that the definitions of useful and non-useful contexts provide a partition of open contexts. The proof is an easy induction on the open context O , the only non immediate case of which is the one for $O = O't$, as it can be guessed by Lemma 5.2.

Lemma 5.3 (Useful partitions open) *A VSC open context O is either useful or non-useful.*

The next easy lemma collects a few properties that are helpful in proofs. The first point says that in a $\rightarrow_{\text{oeabs}}$ step the usefulness of the context does not depend on the acting substitution $[x \leftarrow L \langle \lambda y.t \rangle]$ but only on the composition of the inner and outer open contexts O_1 and O_2 . The definition of (non-)useful steps is then re-stated in Fig. 6.

Lemma 5.4 (Context plugging and usefulness)

- (i) $\text{usef}(O_1 \langle L \langle O_2 \rangle \rangle) \Leftrightarrow \text{usef}(O_1 \langle O_2 \rangle)$.
- (ii) $\text{usef}(O_1 \langle O_2 \rangle) \Leftrightarrow \text{usef}(O_2) \vee (\text{sub}(O_2) \wedge \text{usef}(O_1))$.
- (iii) $\text{nusef}(O_1 \langle O_2 \rangle) \Leftrightarrow \text{nusef}(O_2) \wedge (\text{sub}(O_2) \Rightarrow \text{nusef}(O_1))$.

6 Core Factorization, or Postponing Non-Useful Steps

Since non-useful steps cannot be simulated by the positive calculus, the translation from λ_{ovsc} to λ_{oxpos} of the next section cannot induce a bisimulation—we need a finer approach. The idea is that any reduction sequence in λ_{ovsc} can be factored into a core part (that includes useful steps, that is, \rightarrow_{oeu}) and a non-useful part, and that the evaluation of a term terminates if and only if its core evaluation terminates. In

this section, we prove these two facts. In the next one, we shall give a translation from λ_{ovsc} and λ_{oxpos} inducing a simulation between the core part of λ_{ovsc} and λ_{oxpos} , and a termination equivalence result.

Core Reduction. The (open) core reduction $\rightarrow_{\text{ocore}}$ of λ_{ovsc} is defined as $\rightarrow_{\text{ocore}} := \rightarrow_{\text{om}} \cup \rightarrow_{\text{oeu}} \cup \rightarrow_{\text{oevar}}$, and we dub *Core* λ_{ovsc} the set of VSC terms endowed with $\rightarrow_{\text{ocore}}$. Beyond multiplicative and useful exponential steps, which shall be simulated by λ_{oxpos} , it includes also $\rightarrow_{\text{oevar}}$ steps, which—on purpose—we have *not* classified as useful or non-useful (in another departure from the literature) and which are going to be absorbed. As we explain next, they are crucial for our core factorization theorem.

Non-Useful Postponement. Factorization can be seen as a postponement property, in this case of non-useful steps. Similarly to how we dealt with GC, we first give a local form of postponement of $\rightarrow_{\text{oe nu}}$ steps. The proof of local postponement is simple but more involved than for GC steps, since it requires to check all the (many!) possible cases for a core step following a $\rightarrow_{\text{oe nu}}$ step, which are quite technical to list given the many contexts involved in the definition of (non-)useful rewriting steps. There is also a case of tricky local postponement diagram, where the swap postponing $\rightarrow_{\text{oe nu}}$ requires to do *two* core steps (this case is taken into account in Prop. 6.1.(ii) below):

$$\begin{array}{ccc}
 (xt)[x \leftarrow y][y \leftarrow \lambda z.u] & \xrightarrow{\text{oe nu}} & (xt)[x \leftarrow \lambda z.u][y \leftarrow \lambda z.u] \\
 \text{oe var} \downarrow & & \downarrow \text{oe u} \\
 (yt)[x \leftarrow y][y \leftarrow \lambda z.u] & \xrightarrow{\text{oe u}} & ((\lambda z.u)t)[x \leftarrow y][y \leftarrow \lambda z.u] \xrightarrow{\text{oe nu}} & ((\lambda z.u)t)[x \leftarrow \lambda z.u][y \leftarrow \lambda z.u]
 \end{array}$$

This diagram actually justifies both our avoidance of indirect useful steps in the λ_{ovsc} and the fact that $\rightarrow_{\text{oe var}}$ steps are not labeled as useful/non-useful. Note that the solid lines are exactly what would usually be an indirectly useful step followed by a directly useful one, and that for us they are instead a non-useful step followed by a (directly) useful one. The point is that the sequence can always be re-arranged as shown by the diagram, using $\rightarrow_{\text{oe var}}$ as to turn the replacement on $[x \leftarrow y]$ into a directly non-useful one.

Proposition 6.1 (Local postponement of $\rightarrow_{\text{oe nu}}$) *Let t and u be VSC terms. If $t \rightarrow_{\text{oe nu}} \rightarrow_{\text{ocore}} u$ then $t \rightarrow_{\text{ocore}} \rightarrow_{\text{oe nu}} u$ or $t \rightarrow_{\text{ocore}} \rightarrow_{\text{ocore}} \rightarrow_{\text{oe nu}} u$. More precisely:*

- (i) $\rightarrow_{\text{oe nu}} \rightarrow_{\text{om}} \subseteq \rightarrow_{\text{om}} \rightarrow_{\text{oe nu}}$;
- (ii) $\rightarrow_{\text{oe nu}} \rightarrow_{\text{oe u}} \subseteq \rightarrow_{\text{oe u}} \rightarrow_{\text{oe nu}} \cup \rightarrow_{\text{oe var}} \rightarrow_{\text{oe u}} \rightarrow_{\text{oe nu}}$;
- (iii) $\rightarrow_{\text{oe nu}} \rightarrow_{\text{oe var}} \subseteq \rightarrow_{\text{oe var}} \rightarrow_{\text{oe nu}}$.

Obtaining the global postponement property from the local one is easy because, although the number of core steps can grow with local swaps, the number of non-useful steps is preserved, and can be easily exploited for the induction lifting the local diagrams to the global property.

Theorem 6.2 (Core Factorization / Postponement of non-useful steps) *Let t and u be VSC terms. If $d : t \rightarrow_{\text{o-gc}}^* u$, then $e : t \rightarrow_{\text{ocore}}^* \rightarrow_{\text{oe nu}}^* u$ with $|e|_{\text{om}} = |d|_{\text{om}}$.*

Lastly, we use the postponement property to prove that the core sub-system of λ_{ovsc} is termination-equivalent to the whole of λ_{ovsc} , justifying the core terminology.

Theorem 6.3 (Termination equivalence of λ_{ovsc} and Core λ_{ovsc})

- (i) t has a diverging $\rightarrow_{\text{ovsc}}$ sequence if and only if t has a diverging $\rightarrow_{\text{ocore}}$ sequence;
- (ii) t is $\rightarrow_{\text{ovsc}}$ -weakly normalizing if and only if t is $\rightarrow_{\text{ocore}}$ -weakly normalizing.

7 Translating λ_{ovsc} to λ_{oxpos} and Simulating Core Steps

In this section, we define a translation $\llbracket \cdot \rrbracket$ from λ_{ovsc} to λ_{oxpos} and show that it induces a simulation of the core reduction $\rightarrow_{\text{ocore}}$ of λ_{ovsc} by λ_{oxpos} . There are various delicate points, concerning both the definition and the simulation, discussed all along this section.

TRANSLATION OF SUBSTITUTION CONTEXTS	
$\llbracket \langle \cdot \rangle \rrbracket := (\langle \cdot \rangle, \cdot)$	$\llbracket L[x \leftarrow t] \rrbracket := (E' \langle E \{x \leftarrow y\} \rangle, \sigma \{x \leftarrow y\})$ where $\llbracket L \rrbracket = (E, \sigma)$ and $\llbracket t \rrbracket = E' \langle y \rangle$
TRANSLATION OF TERMS	
$\llbracket x \rrbracket := x$	
$\llbracket \lambda x.t \rrbracket := y[y \leftarrow \lambda x. \llbracket t \rrbracket]$	
$\llbracket t[x \leftarrow u] \rrbracket := E \langle \llbracket t \rrbracket \{x \leftarrow y\} \rangle$	where $\llbracket u \rrbracket = E \langle y \rangle$
$\llbracket L \langle \lambda x.t \rangle u \rrbracket := E \langle E' \langle y[y \leftarrow (\lambda x. \llbracket t \rrbracket \sigma) z] \rangle \rangle$	where $\llbracket L \rrbracket = (E, \sigma)$ and $\llbracket u \rrbracket = E' \langle z \rangle$
$\llbracket tu \rrbracket := E \langle E' \langle y[y \leftarrow xz] \rangle \rangle$	where $\llbracket t \rrbracket = E \langle x \rangle$ and $\llbracket u \rrbracket = E' \langle z \rangle$, if t is not an answer

Fig. 7. The translation from λ_{ovsc} to λ_{oxpos} .

Subtlety 1: Absorption of Variables. Since ESs in λ_{ovsc} can contain variables (as e.g. in $t[x \leftarrow y]$) but ESs in λ_{oxpos} cannot, the translation $\llbracket \cdot \rrbracket$ that we shall define turns these ESs of λ_{ovsc} into meta-level substitutions of λ_{oxpos} . For instance, we shall have $\llbracket t[x \leftarrow y] \rrbracket = \llbracket t \rrbracket \{x \leftarrow y\}$.

Subtlety 2: Answers. It is natural to define $\llbracket \cdot \rrbracket$ as introducing sharing points for every non-variable sub-term (as in Accattoli et al. [3]), which gives the following definition (where the meta-level substitution $\{x \leftarrow y\}$ in the last case is due to the absorption of variables):

$$\begin{array}{l|l} \llbracket x \rrbracket := x & \llbracket tu \rrbracket := E \langle E' \langle x \rangle [x \leftarrow yz] \rangle \text{ where } \llbracket t \rrbracket := E \langle y \rangle \text{ and } \llbracket u \rrbracket = E' \langle z \rangle \\ \llbracket \lambda x.t \rrbracket := y[y \leftarrow \lambda x. \llbracket t \rrbracket] & \llbracket t[x \leftarrow u] \rrbracket := E \langle \llbracket t \rrbracket \{x \leftarrow y\} \rangle \text{ where } \llbracket u \rrbracket = E \langle y \rangle \end{array}$$

Unfortunately, such a definition does not induce a simulation. For instance, consider the following e_u -step:

$$t := (xx)[x \leftarrow \lambda y.u] \rightarrow_{\text{oe}_u} ((\lambda y.u)x)[x \leftarrow \lambda y.u] =: t'$$

Using the above definition of $\llbracket \cdot \rrbracket$, one would need to have:

$$\llbracket t \rrbracket = z[z \leftarrow xx][x \leftarrow \lambda y. \llbracket u \rrbracket] \rightarrow_{\text{ox}_+}^* z[z \leftarrow wx][w \leftarrow \lambda y. \llbracket u \rrbracket][x \leftarrow \lambda y. \llbracket u \rrbracket] = \llbracket t' \rrbracket$$

Note, however, that such a *duplication of ESs* cannot be performed in λ_{oxpos} . Therefore, we rather adopt a modified translation that behaves differently on applied abstractions—more precisely, on applied *answers* (answers are defined in Fig. 1, page 5). The new translation is defined in Fig. 7. In fact, the translation $\llbracket t \rrbracket$ of terms is defined by mutual induction with the translation $\llbracket L \rrbracket$ of substitution contexts, which is used in the case of applied answers. Because of the absorption of variables, the translation $\llbracket L \rrbracket$ of substitution contexts L is not simply an evaluation context of λ_{oxpos} but a *pair* of an evaluation context E and a renaming σ , that is, a meta-level substitution of variables for variables. For instance, $\llbracket \langle \cdot \rangle [x \leftarrow \lambda y.t][w \leftarrow z] \rrbracket = (E, \sigma)$ with $E := \langle \cdot \rangle [x \leftarrow \lambda y. \llbracket t \rrbracket]$ and $\sigma := \{w \leftarrow z\}$.

The next lemma shows that such a translation of substitution contexts is compositional. It is proved by a straightforward induction on L .

Lemma 7.1 *Let $L \langle t \rangle$ be a VSC term and $\llbracket L \rrbracket = (E, \sigma)$. Then $\llbracket L \langle t \rangle \rrbracket = E \langle \llbracket t \rrbracket \sigma \rangle$.*

Simulation. Core reduction $\rightarrow_{\text{ocore}}$ is made out of three kinds of steps, namely \rightarrow_{om} , $\rightarrow_{\text{oe}_u}$, and $\rightarrow_{\text{oe}_{\text{var}}}$. Given the special role of answers in the definition of $\llbracket \cdot \rrbracket$, the proof of the simulation becomes tricky when core steps can turn an applied non-answer into an applied answer. This can happen with \rightarrow_{om} and $\rightarrow_{\text{oe}_u}$ steps, which are then discussed in detail in the next paragraphs. Rule $\rightarrow_{\text{oe}_{\text{var}}}$, instead, does not alter whether sub-terms are answers, and so the proof that $\rightarrow_{\text{oe}_{\text{var}}}$ steps are absorbed is smooth.

USEFUL EXP. ROOT RULE 1	$U\langle\langle x \rangle\rangle[x \leftarrow L\langle\lambda y.t\rangle] \mapsto_{e_{u_1}} L\langle U\langle\langle\lambda y.t\rangle\rangle[x \leftarrow \lambda y.t]\rangle$
USEFUL EXP. ROOT RULE 2	$L_1\langle L_2\langle\langle x \rangle\rangle[x \leftarrow L_3\langle\lambda y.t\rangle]\rangle u \mapsto_{e_{u_2}} L_1\langle L_3\langle L_2\langle\langle\lambda y.t\rangle\rangle[x \leftarrow \lambda y.t]\rangle u$
CTX CLOSURE	$\frac{t \mapsto_a t'}{O\langle t \rangle \rightarrow_{oa} O\langle t' \rangle} \quad a \in \{e_{u_1}, e_{u_2}\}$

Fig. 8. Two root rules for \rightarrow_{oe_u} .

Lemma 7.2 (Absorption of variable exponentials) *Let t and u be VSC terms. If $t \rightarrow_{oe_{var}} u$ then $\llbracket t \rrbracket = \llbracket u \rrbracket$.*

Subtlety 3: Simulation of Multiplicative Steps. Root multiplicative steps are simulated smoothly.

Lemma 7.3 (Simulation of root multiplicative steps) *Let t and u be VSC terms. If $t \mapsto_m u$ then $\llbracket t \rrbracket \rightarrow_{om_+} \llbracket u \rrbracket$.*

A complication arises for the contextual closure of multiplicative steps, because in a root step $L\langle\lambda x.t\rangle u \mapsto_m L\langle t[x \leftarrow u]\rangle$ the redex is not an answer but the reduct might be one, if t is an abstraction. Thus, if the root step is applied to a further argument r , the reduction turns an applied non-answer into an applied answer, changing the clause of the translation that is used for the application to r . This phenomenon is handled by doing two additional rewriting steps in λ_{oxpos} . The simplest case is the following one, where $t = y$, $u = z$, $r = w$, and $L = \langle \cdot \rangle$ and x', y', z' are variables introduced by the translation:

$$\begin{array}{ccc}
 (\lambda x.\lambda y.y)zw & \xrightarrow{\text{om}} & (\lambda y.y)[x \leftarrow z]w \\
 \begin{array}{c} \llbracket \cdot \rrbracket \\ \vdots \\ \downarrow \end{array} & & \begin{array}{c} \downarrow \\ \llbracket \cdot \rrbracket \end{array} \\
 x'[x' \leftarrow y'w][y' \leftarrow (\lambda x.z'[z' \leftarrow \lambda y.y])z] & \xrightarrow{\text{om}_+} x'[x' \leftarrow z'w][z' \leftarrow \lambda y.y] \xrightarrow{\text{oe}_+} x'[x' \leftarrow (\lambda y.y)w][z' \leftarrow \lambda y.y] \xrightarrow{\text{ogc}_+} x'[x' \leftarrow (\lambda y.y)w] &
 \end{array}$$

In general, we have the following simulation of multiplicative steps, where the first case isolates exactly when applied non-answers are turned into applied answers.

Proposition 7.4 (Simulation of \rightarrow_{om} steps) *Let t and u be VSC terms and $t \mapsto_m u$.*

- (i) *If u is an answer and $\text{usef}(O)$ then $\llbracket O\langle t \rangle \rrbracket \rightarrow_{om_+} \rightarrow_{oe_+} \rightarrow_{ogc_+} \llbracket O\langle u \rangle \rrbracket$;*
- (ii) *Otherwise, $\llbracket O\langle t \rangle \rrbracket \rightarrow_{om_+} \llbracket O\langle u \rangle \rrbracket$.*

Subtlety 4: Simulation of Useful Exponential Steps. Exponential steps can turn applied non-answers into applied answers too: these cases actually are the very definition of useful exponential steps. In contrast to the multiplicative case, the simulation does not need extra steps, it simply maps one \rightarrow_{oe_u} step in λ_{ovsc} to one \rightarrow_{oe_+} step in λ_{oxpos} . What is tricky in this case is that the definition of useful steps needs the surrounding open context of the root step, as explained in Sect. 5, so it does not seem possible to prove the simulation for a root case and then extending the property with an induction on the context closure.

To get around this issue, in Fig. 8 we give an alternative definition of \rightarrow_{oe_u} resting on *two* root rules, where the second rule captures the cases when the argument of the created redex is provided by the context. With these two root rules, useful exponential steps can then be defined via a closure by *any* open context, thus bypassing the global aspect of the definition that we gave in Sect. 5. The reader probably wonders why we did not do it already in Sect. 5. The reason is that the global definition is preferable for proving local postponement (Prop. 6.1). The alternative definition is justified by the following lemma.

Lemma 7.5 (Alternative presentation of useful steps) $\rightarrow_{oe_u} = \rightarrow_{oe_{u_1}} \cup \rightarrow_{oe_{u_2}}$.

The simulation is then proved smoothly via the alternative definition.

Proposition 7.6 (Simulation of useful exponential steps) *Let t and u be VSC terms. If $t \rightarrow_{\text{oe}_{u_1}} u$ or $t \rightarrow_{\text{oe}_{u_2}} u$ then $\llbracket t \rrbracket \rightarrow_{\text{oe}_+} \llbracket u \rrbracket$.*

Summing Up. We can now put together the simulations of single core steps, and also iterate over reduction sequences. Since at times one source multiplicative step is simulated by more than one target step, the simulation does not preserve evaluation lengths. An important point to note, however, is that the number of multiplicative steps—which is the cost model of λ_{ovsc} —is preserved. More generally, the increment in length is only linear.

Theorem 7.7 (Simulation of core sequences) *Let $d : t \rightarrow_{\text{ocore}}^* t'$ be a reduction sequence in λ_{ovsc} . Then there exists $e : \llbracket t \rrbracket \rightarrow_{\text{ox}_+}^* \llbracket t' \rrbracket$ in λ_{oxpos} such that $|e|_{\text{om}_+} = |d|_{\text{om}}$ and $|d|_{\text{om}, \text{oe}_u} \leq |e| \leq 3 \cdot |d|$.*

8 Core Normal Forms and Termination Equivalence

In this section, we show that the translation $\llbracket \cdot \rrbracket$ preserves and reflects termination. Reflection is a consequence of the simulation theorem: if $\llbracket t \rrbracket$ terminates then t cannot diverge, because $\llbracket t \rrbracket$ can simulate it. Preservation instead is proved by showing that $\rightarrow_{\text{ocore}}$ normal forms are mapped to (non-erasing) positive normal forms, *i.e.*, $\rightarrow_{\text{ox}_+ \text{-gc}}$ -normal forms, which is proved via a characterization of core normal forms.

Characterization of Core Normal Forms. For the technical characterization of core normal forms, we need a few auxiliary definitions. We start by defining two sets of variables for terms.

Definition 8.1 The set $\text{ofv}(t)$ of *open free variables* of a VSC term t is the set of variables of t having occurrences out of all abstractions, formally defined as follows:

$$\begin{aligned} \text{ofv}(x) &:= \{x\} & \text{ofv}(tu) &:= \text{ofv}(t) \cup \text{ofv}(u) \\ \text{ofv}(\lambda x.t) &:= \emptyset & \text{ofv}(t[x \leftarrow u]) &:= (\text{ofv}(t) \setminus \{x\}) \cup \text{ofv}(u) \end{aligned}$$

The set $\text{aofv}(t)$ of *applied open free variables* of a VSC term t is the set of variables of t having applied occurrences out of all abstractions, formally defined as follows:

$$\begin{array}{l} \text{aofv}(x) = \text{aofv}(\lambda x.t) := \emptyset \\ \text{aofv}(t[x \leftarrow u]) := (\text{aofv}(t) \setminus \{x\}) \cup \text{aofv}(u) \end{array} \quad \left| \quad \begin{array}{ll} \text{aofv}(tu) & := \text{aofv}(t) \cup \text{aofv}(u) \cup \{x\} \text{ if } t = L\langle\langle x \rangle\rangle \\ \text{aofv}(tu) & := \text{aofv}(t) \cup \text{aofv}(u) \text{ otherwise} \end{array} \right.$$

We also need a weakened notion of answer.

Definition 8.2 An *almost answer* is an answer or a VSC term of the form $L\langle L'\langle x \rangle[x \leftarrow t] \rangle$ where t is an answer.

Finally, we can provide a characterization of core normal forms, based on the following grammar.

GRAMMAR OF CORE NORMAL TERMS

$$\begin{aligned} n &= v \\ &| nn' \quad \text{with } n \text{ not an almost answer} \\ &| n[x \leftarrow n'] \text{ with } n' = L\langle \lambda y.t \rangle \text{ and } x \notin \text{aofv}(n) \\ &| n[x \leftarrow n'] \text{ with } n' = L\langle y \rangle \text{ and } x \notin \text{ofv}(n) \\ &| n[x \leftarrow n'] \text{ with } n' = L\langle tu \rangle \end{aligned}$$

Proposition 8.3 (Characterization of core normal forms) *Let t be a VSC term. t is $\rightarrow_{\text{ocore}}$ -normal if and only if it is a n term.*

Via a few technical lemmas (in the tech report [13]), we obtain the following preservation property.

Proposition 8.4 (Preservation of core normal forms) *Let t be a VSC term. If t is $\rightarrow_{\text{ocore-normal}}$ then $\llbracket t \rrbracket$ is $\rightarrow_{\text{ox}_+ \text{-gc-normal}}$.*

Theorem 8.5 (Termination equivalence of Core λ_{ovsc} and λ_{oxpos}) *Let t be a VSC term.*

- (i) t has a diverging $\rightarrow_{\text{ocore}}$ sequence if and only if $\llbracket t \rrbracket$ has a diverging $\rightarrow_{\text{ox}_+}$ sequence.
- (ii) t is $\rightarrow_{\text{ocore}}$ -weakly normalizing if and only if $\llbracket t \rrbracket$ is $\rightarrow_{\text{ox}_+}$ -weakly normalizing.

As a corollary, we can prove *uniform normalization* (see Sect. 2) for λ_{ovsc} and Core λ_{ovsc} . Uniform normalization follows immediately from the diamond property, thus it holds for λ_{oxpos} (Thm. 4.3). Concerning λ_{ovsc} and Core λ_{ovsc} , the proof is instead not immediate, because they are not diamond (see the diagram at page 6). But we can obtain it by lifting the one of λ_{oxpos} via the proved termination equivalences.

Corollary 8.6 λ_{ovsc} and Core λ_{ovsc} are uniformly normalizing.

9 Conclusions

This paper studies Wu’s positive λ -calculus λ_{pos} , which at first sight looks simply as yet another call-by-value λ -calculus with sharing. It has, however, a new feature that distinguishes it among similar calculi, called here *compactness* and concerning the treatment of variables.

Our main contribution is showing that compactness allows one to elegantly capture the essence of useful sharing (in an open setting), circumventing the main technicalities of this notion. What is remarkable is that λ_{pos} has not arisen as an incremental refinement of useful sharing, but as an outcome of the completely unrelated study of focalization for minimal intuitionistic logic by Miller and Wu [28].

We believe that the positive λ -calculus is a sharp tool deserving to be studied further, in particular with respect to program transformations and optimizations, and also endowed with call-by-need evaluation.

An aspect that we have left for future work is the efficient implementation of the meta-level renamings involved in the multiplicative rewriting rule, which can be computationally costly if done without care. We expect it to be doable efficiently via an appropriate abstract machine.

References

- [1] Accattoli, B., *Proof nets and the call-by-value λ -calculus*, Theor. Comput. Sci. **606**, pages 2–24 (2015). <https://doi.org/10.1016/j.tcs.2015.08.006>
- [2] Accattoli, B., A. Condoluci, G. Guerrieri and C. Sacerdoti Coen, *Crumbing abstract machines*, in: E. Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15, ACM (2019). <https://doi.org/10.1145/3354166.3354169>
- [3] Accattoli, B., A. Condoluci and C. Sacerdoti Coen, *Strong call-by-value is reasonable, implisively*, in: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14, IEEE (2021). <https://doi.org/10.1109/LICS52264.2021.9470630>
- [4] Accattoli, B. and U. Dal Lago, *(leftmost-outermost) beta reduction is invariant, indeed*, Log. Methods Comput. Sci. **12** (2016). [https://doi.org/10.2168/LMCS-12\(1:4\)2016](https://doi.org/10.2168/LMCS-12(1:4)2016)
- [5] Accattoli, B., U. Dal Lago and G. Vanoni, *Reasonable space for the λ -calculus, logarithmically*, in: C. Baier and D. Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 47:1–47:13, ACM (2022). <https://doi.org/10.1145/3531130.3533362>
- [6] Accattoli, B., C. Faggian and A. Lancelot, *Normal form bisimulations by value*, CoRR **abs/2303.08161** (2023). <https://doi.org/10.48550/arXiv.2303.08161>

- [7] Accattoli, B. and G. Guerrieri, *Open call-by-value*, in: A. Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226 (2016).
https://doi.org/10.1007/978-3-319-47958-3_12
- [8] Accattoli, B. and G. Guerrieri, *Abstract machines for open call-by-value*, *Sci. Comput. Program.* **184** (2019).
<https://doi.org/10.1016/J.SCICO.2019.03.002>
- [9] Accattoli, B. and M. Leberle, *Useful open call-by-need*, in: F. Manea and A. Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 4:1–4:21, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022).
<https://doi.org/10.4230/LIPICS.CSL.2022.4>
- [10] Accattoli, B. and L. Paolini, *Call-by-value solvability, revisited*, in: T. Schrijvers and P. Thiemann, editors, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16, Springer (2012).
https://doi.org/10.1007/978-3-642-29822-6_4
- [11] Accattoli, B. and C. Sacerdoti Coen, *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*, pages 141–155, IEEE Computer Society (2015).
<https://doi.org/10.1109/LICS.2015.23>
- [12] Accattoli, B. and C. Sacerdoti Coen, *On the value of variables*, *Information and Computation* **255**, pages 224–242 (2017).
<https://doi.org/10.1016/j.ic.2017.01.003>
- [13] Accattoli, B. and J.-H. Wu, *Positive Focusing is Directly Useful* (2024). Long version with proofs.
<https://hal.science/hal-04606194>
- [14] Andreoli, J., *Logic programming with focusing proofs in linear logic*, *J. Log. Comput.* **2**, pages 297–347 (1992).
<https://doi.org/10.1093/LOGCOM/2.3.297>
- [15] Brock-Nannestad, T., N. Guenot and D. Gustafsson, *Computation in focused intuitionistic logic*, in: M. Falaschi and E. Albert, editors, *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14-16, 2015*, pages 43–54, ACM (2015).
<https://doi.org/10.1145/2790449.2790528>
- [16] Chaudhuri, K., *Focusing strategies in the sequent calculus of synthetic connectives*, in: I. Cervesato, H. Veith and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 467–481, Springer (2008).
https://doi.org/10.1007/978-3-540-89439-1_33
- [17] Chaudhuri, K., S. Hetzl and D. Miller, *A multi-focused proof system isomorphic to expansion proofs*, *J. Log. Comput.* **26**, pages 577–603 (2016).
<https://doi.org/10.1093/LOGCOM/EXU030>
- [18] Chaudhuri, K., D. Miller and A. Saurin, *Canonical sequent proofs via multi-focusing*, in: G. Ausiello, J. Karhumäki, G. Mauri and C. L. Ong, editors, *Fifth IFIP International Conference On Theoretical Computer Science - TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, September 7-10, 2008, Milano, Italy*, volume 273 of *IFIP*, pages 383–396, Springer (2008).
https://doi.org/10.1007/978-0-387-09680-3_26
- [19] Curien, P. and H. Herbelin, *The duality of computation*, in: M. Odersky and P. Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, pages 233–243, ACM (2000).
<https://doi.org/10.1145/351240.351262>
- [20] Dal Lago, U. and S. Martini, *The weak lambda calculus as a reasonable machine*, *Theor. Comput. Sci.* **398**, pages 32–50 (2008).
<https://doi.org/10.1016/J.TCS.2008.01.044>
- [21] Economou, D. J., N. Krishnaswami and J. Dunfield, *Focusing on refinement typing*, *ACM Trans. Program. Lang. Syst.* **45**, pages 22:1–22:62 (2023).
<https://doi.org/10.1145/3610408>
- [22] Flanagan, C., A. Sabry, B. F. Duba and M. Felleisen, *The essence of compiling with continuations*, in: R. Cartwright, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, pages 237–247, ACM (1993).
<https://doi.org/10.1145/155090.155113>

- [23] Friedman, D. P., A. Ghuloum, J. G. Siek and O. L. Winebarger, *Improving the lazy krivine machine*, High. Order Symb. Comput. **20**, pages 271–293 (2007).
<https://doi.org/10.1007/S10990-007-9014-0>
- [24] Krishnaswami, N. R., *Focusing on pattern matching*, in: Z. Shao and B. C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 366–378, ACM (2009).
<https://doi.org/10.1145/1480881.1480927>
- [25] Levy, P. B., J. Power and H. Thielecke, *Modelling environments in call-by-value programming languages*, Inf. Comput. **185**, pages 182–210 (2003).
[https://doi.org/10.1016/S0890-5401\(03\)00088-9](https://doi.org/10.1016/S0890-5401(03)00088-9)
- [26] Liang, C. C. and D. Miller, *Focusing and polarization in linear, intuitionistic, and classical logics*, Theor. Comput. Sci. **410**, pages 4747–4768 (2009).
<https://doi.org/10.1016/J.TCS.2009.07.041>
- [27] Marin, S., D. Miller, E. Pimentel and M. Volpe, *From axioms to synthetic inference rules via focusing*, Ann. Pure Appl. Log. **173**, page 103091 (2022).
<https://doi.org/10.1016/J.APAL.2022.103091>
- [28] Miller, D. and J.-H. Wu, *A positive perspective on term representation (invited talk)*, in: B. Klin and E. Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPICs*, pages 3:1–3:21, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023).
<https://doi.org/10.4230/LIPICs.CSL.2023.3>
- [29] Moggi, E., *Computational λ -Calculus and Monads*, LFCS report ECS-LFCS-88-66, University of Edinburgh (1988).
<http://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-66/ECS-LFCS-88-66.pdf>
- [30] Moggi, E., *Computational lambda-calculus and monads*, in: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23, IEEE Computer Society (1989).
<https://doi.org/10.1109/LICS.1989.39155>
- [31] Rioux, N. and S. Zdancewic, *Computation focusing*, Proc. ACM Program. Lang. (ICFP) **4**, pages 95:1–95:27 (2020).
<https://doi.org/10.1145/3408977>
- [32] Sabry, A. and M. Felleisen, *Reasoning about programs in continuation-passing style*, in: J. L. White, editor, *Proceedings of the Conference on Lisp and Functional Programming, LFP 1992, San Francisco, California, USA, 22-24 June 1992*, pages 288–298, ACM (1992).
<https://doi.org/10.1145/141471.141563>
- [33] Sabry, A. and P. Wadler, *A Reflection on Call-by-Value*, ACM Trans. Program. Lang. Syst. **19**, pages 916–941 (1997).
<https://doi.org/10.1145/267959.269968>
- [34] Sands, D., J. Gustavsson and A. Moran, *Lambda Calculi and Linear Speedups*, in: *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, pages 60–84 (2002).
https://doi.org/10.1007/3-540-36377-7_4
- [35] Scherer, G., *Deciding equivalence with sums and the empty type*, in: G. Castagna and A. D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 374–386, ACM (2017).
<https://doi.org/10.1145/3009837.3009901>
- [36] Scherer, G. and D. Rémy, *Which simple types have a unique inhabitant?*, in: K. Fisher and J. H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 243–255, ACM (2015).
<https://doi.org/10.1145/2784731.2784757>
- [37] Sestoft, P., *Deriving a lazy abstract machine*, J. Funct. Program. **7**, pages 231–264 (1997).
<https://doi.org/10.1017/S0956796897002712>
- [38] Walker, D., *Substructural Type Systems*, in: *Advanced Topics in Types and Programming Languages*, The MIT Press (2004), ISBN 9780262281591. https://direct.mit.edu/book/chapter-pdf/186357/9780262281591_caa.pdf.
<https://doi.org/10.7551/mitpress/1104.003.0003>
- [39] Wand, M., *On the correctness of the krivine machine*, High. Order Symb. Comput. **20**, pages 231–235 (2007).
<https://doi.org/10.1007/S10990-007-9019-8>

- [40] Wu, J.-H., *Proofs as terms, terms as graphs*, in: C. Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 91–111, Springer (2023).
https://doi.org/10.1007/978-981-99-8311-7_5
- [41] Zeilberger, N., *Focusing and higher-order abstract syntax*, in: G. C. Necula and P. Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 359–369, ACM (2008).
<https://doi.org/10.1145/1328438.1328482>