Logical Methods in Computer Science
Vol. 12(2:12)2016, pp. 1–43
www.lmcs-online.org

Submitted Nov. 19, 2015
Published Jun. 28, 2016

NO SOLVABLE LAMBDA-VALUE TERM LEFT BEHIND

ÁLVARO GARCÍA-PÉREZ [a] AND PABLO NOGUEIRA [b]

[a] Reykjavík University, Iceland
e-mail address: alvarog@ru.is

[b] IMDEA Software Institute, Madrid, Spain
e-mail address: pablo.nogueira@imdea.org

Abstract. In the lambda calculus a term is solvable iff it is operationally relevant. Solvable terms are a superset of the terms that convert to a final result called normal form. Unsolvable terms are operationally irrelevant and can be equated without loss of consistency. There is a definition of solvability for the lambda-value calculus, called v-solvability, but it is not synonymous with operational relevance, some lambda-value normal forms are unsolvable, and unsolvables cannot be consistently equated. We provide a definition of solvability for the lambda-value calculus that does capture operational relevance and such that a consistent proof-theory can be constructed where unsolvables are equated attending to the number of arguments they take (their 'order' in the jargon). The intuition is that in lambda-value the different sequentialisations of a computation can be distinguished operationally. We prove a version of the Genericity Lemma stating that unsolvable terms are generic and can be replaced by arbitrary terms of equal or greater order.

1. Introduction

Call-by-value is a common evaluation strategy of many functional programming languages, whether full-fledged or fragments of proof assistants. Such languages and their evaluation strategies can be formalised operationally in terms of an underlying lambda calculus and its reduction strategies. As shown in [Plo75], the classic lambda calculus λK [Bar84] is inadequate to formalise call-by-value evaluation as defined by Landin's SECD abstract machine. The adequate calculus is the lambda-value calculus λ_V. The pure (and untyped) version [RP04] is the core that remains after stripping away built-in primitives whose main purpose is to facilitate the encoding of programs as terms of the calculus. Hereafter we write λ_V for the pure version.

2012 ACM CCS: [**Theory of computation**]: Models of computation—Computability—Lambda calculus; Semantics and Reasoning—Operational semantics; Logic—Proof theory.

Key words and phrases: Lambda-value calculus, solvability, operational relevance, effective use, consistent lambda-theories.

This research has been partially funded by the Spanish Ministry of Economy and Competitiveness through project STRONGSOFT TIN2012-39391-C04-02, by the Regional Government of Madrid through programme N-GREENS SOFTWARE S2013/ICE-2731, and by the Icelandic Research Fund through project NOSOS 141558-053.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-12(2:12)2016

Unfortunately, the lambda-value calculus, and by extension its pure version, are considered defective on several fronts for formalising call-by-value evaluation at large, and many alternative calculi have been proposed with various aims, *e.g.* [FF86, HZ09, Mog91, EHR91, AK10, AK12, AP12].

We do not wish to propose yet another calculus. These proposals vary the calculus to fit an intended call-by-value model, but this is one of the choices for investigations on full abstraction. The other is to vary the model to fit the intended calculus [Cur07, p.1]. The questions are: What does λ_V model? Is its import larger than call-by-value evaluation under SECD? To answer these questions and avoid 'the mismatch between theory [the calculus] and practice [the model]' [Abr90, p.2] we have to first address the open problem of whether λ_V has a 'standard theory'. A central piece of a standard theory is the notion of solvability which is synonymous with operational relevance. Let us elaborate these ideas first and discuss their utility further below.

Recall that a lambda calculus consists of a set of terms and of proof-theories for conversion and reduction of terms. Conversion formalises intensional (computational) equality and reduction formalises directed computation. A term converts/reduces to another term (both terms are in a conversion/reduction relation) iff this fact can be derived in the conversion/reduction proof-theory (Section 2 illustrates). The relations must be confluent for the proof-theory to be consistent. In the calculus the reduction relation is full-reducing and 'goes under lambda'. It is possible to reason algebraically at any scope where free variables (which stand for unknown operands in that scope) occur. Operational equivalence can be established for 'arbitrary terms, not necessarily closed nor of observable type' [Cur07, p.3].

Solvability is a basic concept in lambda calculus. It appears 18 pages after the definition of terms in the standard reference [Bar84] (terms are defined on page 23 and solvability on page 41). Solvability was first studied in [Bar71, Bar72, Wad76] and stems from the realisation that not all diverging terms (*i.e.* terms whose reduction does not terminate) are operationally irrelevant (*i.e.* meaningless, useless, of no practical use, etc.) For a start, not all of them are equal. An inconsistent proof-theory results from extending the conversion proof-theory with equations between all diverging terms. Indeed, some diverging terms can be applied to suitable operands such that the application converges to a definite final result of the calculus (a 'normal form' in the jargon). For other diverging terms the application diverges no matter to how many or to which operands they are applied. Solvable terms are therefore terms from which a normal form can be obtained when used as functions. The name 'solvable' stems from their characterisation as solutions to a conversion. By definition, terms that directly convert to a normal form are solvable.

In contrast, unsolvable terms are the terms that are operationally irrelevant. A consistent proof-theory results from extending the conversion proof-theory with equations between all unsolvables. This consistent extension is satisfied by well-known models where unsolvables correspond to the least-defined element of the model. Any further extension that includes the equations between unsolvables and is consistent is called *sensible* in the jargon. Finally, solvable terms can be characterised operationally: there is a reduction strategy named 'head reduction' that converges iff the input term is solvable.

To summarise: λK has a definition of solvability synonymous with operational relevance, a sensible extended proof-theory, sensible models (*i.e.* models of the sensible extension), and an operational characterisation of solvables. All these ingredients are referred to in [Abr90, p.2] as a 'standard theory'.

However, in that work λK's standard theory is criticised as a basis for functional programming languages because program results are not normal forms, there are no canonical initial models, etc. (Strictly speaking, however, λK is as unfit as Turing Machines as a basis for practical programming languages.) A 'lazy' lambda calculus is proposed which is closer to a non-strict functional programming language, but that divorces solvability from operational relevance. The latter is modified according to the notion of 'order of a term' [Lon83]. Broadly, the order is the supremum (ordinal) number of operands accepted by the term in the following inductive sense: if the term converts to $\lambda x.M$ then it accepts $n+1$ operands where n is the number of operands accepted by M. Otherwise the term has order 0. Operationally irrelevant terms are only the unsolvables of order 0. Other unsolvables are operationally relevant and the extended proof-theory that equates unsolvables of order $n > 0$ is inconsistent.

Following similar steps, [PR99, EHR91, EHR92, RP04] describe a call-by-value calculus with a proof-theory induced by operational equivalence of terms under SECD reduction. A definition of solvability, called v-solvability, is proposed for λ_V. This definition is unsatisfactory because it does not adapt λK's original definition of solvable term, namely, 'the application of the term to suitable operands converts to a normal form'. It adapts a derived definition, namely, 'the application of the term to suitable operands converts to the identity term'. This definition is equivalent to the former in λK but not in λ_V. Consequently, v-solvability does not capture operational relevance in λ_V, some normal forms of λ_V (definite results) are v-unsolvable, and the extended proof-theory is not sensible. Moreover, the operational characterisation of v-solvables involves a reduction strategy of λK, not of λ_V, and the notion of order used is not defined in terms of λ_V's conversion in a way analogous to [Lon83]. The blame is put on λ_V's nature and continues to be put in recent related work [AP12, Gue13, CG14, GPR15].

We show that λ_V does indeed have a standard theory. First we revisit the original definition of solvability in λK and generalise it by connecting it with the notion of effective use of an arbitrary (closed or open) term. We then revisit v-solvability and show that it does not capture operational relevance in λ_V but rather 'transformability', *i.e.* the ability to send a term to a chosen value. (Values are not definite results of λ_V but a requirement for confluence.) We introduce λ_V-solvability as the ability to use the term effectively. Our λ_V-solvability captures transformability and 'freezability', *i.e.* the ability to send a term to a normal form, albeit not of our choice. The intuition is that terms can also be solved by sending them to normal forms that differ operationally from divergent terms at a point of potential divergence. The link between solvability and effective use is a definition of order that uses λ_V's conversion, and a Partial Genericity Lemma which states that λ_V-unsolvables of order n are generic (can be replaced by any term) for orders greater or equal than n. The λ_V-unsolvables of the same order can be equated without loss of consistency, and so we construct a consistent extension which we call $\mathcal{V}$. Our proof of the Partial Genericity Lemma is based on the proof of λK's Genericity Lemma presented in [BKC00] that uses origin tracking. An ingredient of the proof is the definition of a complete reduction strategy of λ_V which we call 'value normal order' because we have defined it by adapting to λ_V the results in [BKKS87] relative to the complete 'normal order' strategy of λK. Value normal order relies on what we call 'chest reduction' and 'ribcage reduction' in the spirit of the anatomical analogy for terms in [BKKS87]. The last two strategies illustrate that standard reduction sequences fall short of capturing all complete strategies of λ_V, and that a result analogous to 'quasi-needed reduction is normalising' [BKKS87, p.208] is missing for λ_V. An operational characterisation of solvables in terms of a reduction strategy of λ_V is complicated but we believe possible (Section 7.5).

To summarise, our contributions are: a definition of solvability in λ_V that is synonymous with operational relevance, the Partial Genericity Lemma, the reduction strategies value normal order, chest reduction and ribcage reduction, and finally the sensible proof-theory where unsolvables of the same order are equated.

The standard theory of λ_V has practical consequences other than reducing the mismatch between theory and practice, or the operational formalisation of call-by-value. Terms with the same functional result that may have different sequentiality under different reduction strategies can be distinguished operationally. Models for sequentiality exist [BC82]. The full-reducing and open-terms aspect of the calculus has applications in program optimisation by partial evaluation and type checking in proof assistants [Cré90], in the PoplMark challenge [ABF+05], in reasoning within local open scopes [Cha12], etc. The computational overload incurred by proofs-by-reflection can be mitigated by reducing terms fully [GL02]. Finally, that some non-terminating terms (unsolvables) can be equated without loss of consistency is of interest to proof assistants with a non-terminating full-reducing programmatic fragment, *e.g.* [ACP+08].

This paper can be read by anyone able to follow the basic conventional lambda calculus notions and notations that we recall in Section 2. The first part of the paper provides the necessary exegesis and intuitions on λK, λ_V, solvability, effective use, v-solvability, and introduces our λ_V-solvability. The more technical second part involves the proof of the Partial Genericity Lemma and the consistent proof-theory. Some background material and routine proofs are collected in the appendix. References to the latter are labelled 'App.' followed by a section number.

2. Overview of λK and λ_V

This preliminaries section must be of necessity terse. Save for the extensive use of EBNF grammars to define sets of terms, we follow definitions and notational conventions of [Bar84, HS08] for λK and of [Plo75] for λ_V. The book [RP04] collects and generalises both calculi. The set of lambda terms is $\Lambda ::= x \mid (\lambda x.\Lambda) \mid (\Lambda\,\Lambda)$ with 'x' one element of a countably infinite set of variables that we overload in grammars as non-terminal for such set. Uppercase, possibly primed letters M, M', N, etc., will stand for terms. In words, a term is a variable, or an abstraction $(\lambda x.M)$ with bound variable x and body M, or the application (MN) of an operator M to an operand N. We follow the common precedence and association convention where applications associate to the left and application binds stronger than abstraction. Hence, we can drop parenthesis and write $(\lambda x.x\,y)\,p\,q\,(\lambda x.x)$ rather than $((((\lambda x.(x\,y))p)q)(\lambda x.x))$, and we can write $\Lambda ::= x \mid \lambda x.\Lambda \mid \Lambda\,\Lambda$, and $\lambda x.M$, and MN. For brevity we write $\lambda x_1 \ldots x_n.M$ instead of $\lambda x_1.\lambda x_2.\ldots\lambda x_n.M$. We write FV for the function that delivers the set of free variables of a term. We assume the notions of bound and free variable and write $\equiv$ for the identity relation on terms modulo renaming of bound variables.[1] For example, $\lambda x.xz \equiv \lambda y.yz$. We also abuse $\equiv$ to define abbreviations, *e.g.* $\mathbf{I} \equiv \lambda x.x$. Like [CF58, HS08], we write $[N/x]M$ for the capture-avoiding substitution of N for the free occurrences of x in M. We write Λ^0 for the set of closed lambda terms, *i.e.* terms M such that $\mathrm{FV}(M) = \emptyset$. We use the same postfix superscript for the operation on a

[1]We are following here the convention of Appendix C in [Bar84] not to be confused with the 'Barendregt convention' or 'hygiene rule' of [Bar90] where bound variables and free variables must differ.

set of terms that delivers the subset of closed terms. The set of values (non-applications) is $\mathsf{Val} ::= x \mid \lambda x.\Lambda$. The set of closed values is Val^0 and consists of closed abstractions. A context $\mathbf{C}[\ ]$ is a term with one hole, *e.g.* $\mathbf{C}[\ ] \equiv \lambda x.[\ ]$. Plugging a term within the hole may involve variable capture, *e.g.* $\mathbf{C}[\lambda y.x] \equiv \lambda x.\lambda y.x$.

The conversion/reduction proof-theories of λK and λ_V can be presented as instances of the Hilbert-style proof-theory shown in Fig. 1 that is parametric (*cf.* [RP04]) on a set $\mathbb{P}$ of permissible operands N in the contraction rule (β) which describes the conversion/reduction of the term $(\lambda x.B)N$, that is, the application of an abstraction (a function) to an operand. Operands are arbitrary terms in λK and restricted to values in λ_V which means that λ_V has fewer conversions/reductions than λK.

$$(\beta)\ \frac{N \in \mathbb{P}}{(\lambda x.B)N = [N/x]B} \qquad (\mu)\ \frac{N = N'}{M\,N = M\,N'} \qquad (\nu)\ \frac{M = M'}{M\,N = M'\,N} \qquad (\xi)\ \frac{B = B'}{\lambda x.B = \lambda x.B'}$$

$$(\rho)\ \frac{}{M = M} \qquad (\tau)\ \frac{M = N \quad N = P}{M = P} \qquad (\sigma)\ \frac{M = N}{N = M}$$

Theory	$=$	$\mathbb{P}$	discarded rules
λK conversion	$=_\beta$	Λ	none
λK multiple-step reduction	$\rightarrow^*_\beta$	Λ	σ
λK single-step reduction	$\rightarrow_\beta$	Λ	ρ, τ, σ
λ_V conversion	$=_{\beta_V}$	Val	none
λ_V multiple-step reduction	$\rightarrow^*_{\beta_V}$	Val	σ
λ_V single-step reduction	$\rightarrow_{\beta_V}$	Val	ρ, τ, σ

Figure 1: λK and λ_V proof-theories.

Set			Description	Abbreviation in the text
Λ	$::=$	$x \mid \lambda x.\Lambda \mid \Lambda\,\Lambda$	lambda terms	
Val	$::=$	$x \mid \lambda x.\Lambda$	values	
Neu	$::=$	$x\,\Lambda\,\{\Lambda\}^*$	λK neutrals	
NF	$::=$	$\lambda x.\mathsf{NF} \mid x\,\{\mathsf{NF}\}^*$	λK normal forms	β-nfs (singular β-nf)
HNF	$::=$	$\lambda x.\mathsf{HNF} \mid x\,\{\Lambda\}^*$	head normal forms	hnfs (singular hnf)
NeuV	$::=$	$\mathsf{Neu} \mid \mathsf{Block}\,\{\Lambda\}^*$	λ_V neutrals	
Block	$::=$	$(\lambda x.\Lambda)\mathsf{NeuV}$	blocks	
VNF	$::=$	$x \mid \lambda x.\mathsf{VNF} \mid \mathsf{Stuck}$	λ_V normal forms	β_V-nfs (singular β_V-nf)
Stuck	$::=$	$x\,\mathsf{VNF}\,\{\mathsf{VNF}\}^*$	stucks	
	$\mid$	$\mathsf{BlockNF}\,\{\mathsf{VNF}\}^*$		
$\mathsf{BlockNF}$	$::=$	$(\lambda x.\mathsf{VNF})\,\mathsf{Stuck}$	blocks in β_V-nf	

Figure 2: Sets of terms.

In λ_V the rule (β) restricted to operand values is named (β_V). The term $(\lambda x.B)N$ is called a β-redex iff $N \in \Lambda$, and a β_V-redex iff $N \in \mathsf{Val}$. A term is a β-normal-form (hereafter abbrev. β-nf) iff it has no β-redexes. A term is a β_V-nf iff it has no β_V-redexes. The

Abbreviation	Term	has β-nf	has β_V-nf
$\mathbf{I}$	$\lambda x.x$	yes	yes
$\mathbf{K}$	$\lambda x.\lambda y.x$	yes	yes
$\mathbf{\Delta}$	$\lambda x.xx$	yes	yes
$\mathbf{\Omega}$	$\mathbf{\Delta\Delta}$	no	no
$\mathbf{U}$	$\lambda x.\mathbf{B}$	no	yes
$\mathbf{B}$	$(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}$	no	yes

Figure 3: Glossary of particular terms.

inference rules are: compatibility (μ) (ν) (ξ), reflexivity (ρ), transitivity (τ), and symmetry (σ). The table underneath names the proof-theory obtained, and the relation symbol, for given $\mathbb{P}$ and rules. The conversion relation includes the reduction relation. A term M has a β-nf N when $M =_\beta N$ and N is a β-nf. A term M has a β_V-nf N when $M =_{\beta_V} N$ and N is a β_V-nf. A term M has a value when $M =_{\beta_V} N$ and $N \in \mathsf{Val}$. All proof-theories are consistent (not all judgements are derivable) due to confluence (a term has at most one β-nf and at most one β_V-nf).

Fig. 2 defines sets of terms and Fig. 3 defines abbreviations of terms used in the following sections. A full table of sets of terms and abbreviations of terms is provided in App. A. Observe that every term of Λ has the form $\lambda x_1 \ldots x_n.\, H\, M_1 \cdots M_m$ where $n \geq 0$, $m \geq 0$, and $M_1 \in \Lambda$, ..., $M_m \in \Lambda$. The head term H is either a 'head variable' x (which may or may not be one of $x_1 \ldots x_n$) or an application $(\lambda x.B)N$ (which is a redex iff $N \in \mathbb{P}$). The set Neu of *neutrals* of λK contains applications $x\, M_1 \cdots M_n$ with $n \geq 1$. The expression $\{\Lambda\}^*$ in the grammar stands for zero or more occurrences of Λ. The applications associate as $(\ldots((x\, M_1)M_2)\cdots M_n)$ according to the standard convention. The set NF of β-nfs consists of abstractions with bodies in β-nf, free variables, and neutrals in β-nf. According to the grammar, every β-nf has the form $\lambda x_1 \ldots x_n.x\, N_1 \cdots N_m$ where $n \geq 0$, $m \geq 0$, $N_1 \in \mathsf{NF}$, ..., $N_m \in \mathsf{NF}$, and x may or may not be one of $x_1 \ldots x_n$. The set HNF of head normal forms (abbrev. hnfs) consists of terms that differ from β-nfs in that $N_1 \in \Lambda$, ..., $N_m \in \Lambda$. Clearly, $\mathsf{NF} \subset \mathsf{HNF}$.

Some examples: $\lambda x.\mathbf{I}$ is a β-nf and a hnf, $\lambda x.\mathbf{I\Delta}$ is not a β-nf (it contains the β-redex $\mathbf{I\,\Delta}$) nor a hnf (it has no head variable), $\lambda x.\, x\,\mathbf{I\Delta}$ is not a β-nf but it is a hnf, and both $x\,(\lambda x.\,\mathbf{I})$ and $x\,\mathbf{\Omega}$ are neutrals, with only the first in β-nf.

The set NeuV of neutrals of λ_V contains the neutrals Neu of λK and blocks applied to zero or more terms. The set Block of *blocks* contains applications $(\lambda x.B)N$ where $N \in \mathsf{NeuV}$. These are applications that do not convert to a β_V-redex and are therefore blocked. (Our blocks differ from the 'head blocks' of [RP04, p.8] and the 'pseudo redexes' of [HZ09, p.4] which require $N \notin \mathsf{Val}$ and so include terms like $(\lambda x.B)(\mathbf{I\,I})$ that convert to a β_V-redex.) The set VNF of β_V-nfs contains variables, abstractions in β_V-nf, and *stuck terms* ('stucks' for short) which are neutrals of λ_V in β_V-nf. The set Stuck of stucks contains Neu neutrals of λK in β_V-nf and blocks in β_V-nf. According to the grammar, every β_V-nf has the form $\lambda x_1 \ldots x_n.H\, Z_1 \cdots Z_m$ with $n \geq 0$, $m \geq 0$, $Z_1 \in \mathsf{VNF}$, ..., $Z_m \in \mathsf{VNF}$, and H either a variable or a block in β_V-nf.

Some examples: $x\,\mathbf{\Omega}$ is a neutral not in β_V-nf, $x\,\mathbf{\Delta}$ is a neutral in β_V-nf (a stuck), $(\lambda x.y)(x\,\mathbf{\Omega})$ is a block not in β_V-nf, and $(\lambda x.y)(x\,\mathbf{\Delta})$ is a block in β_V-nf (a stuck).

A *reduction strategy* of λK (resp. of λ_V) is a partial function that is a subrelation of $\to_\beta^*$ (resp. of $\to_{\beta_V}^*$). A reduction strategy is *complete* with respect to a notion of irreducible term when the strategy delivers the irreducible term iff the input term has one, diverging otherwise. A reduction strategy is *full-reducing* when the notion of irreducible term is a β-nf (resp. β_V-nf). The Quasi-Leftmost Reduction Theorem [HS08, Thm. 3.22] states, broadly, that any reduction strategy of λK that eventually contracts the leftmost redex is full-reducing and complete. One such well-known strategy is leftmost reduction [CF58], also known as leftmost-outermost reduction (when referring to the redex's position in the abstract syntax tree of the term) or, more commonly, as normal order. The Standardisation Theorem [Plo75, Thm. 3] guarantees that there are full-reducing and complete strategies of λ_V. One such strategy is described in [RP04] and discussed in Section 7.1.

3. Solvability reloaded

As explained in the introduction, a term is solvable iff a normal form can be obtained from it when used as a function. Solvability is usually defined first for closed terms and then extended to open terms.

Definition 3.1 (SolN)**.** A term $M \in \Lambda^0$ is solvable in λK iff there exists $N \in \mathsf{NF}$ and there exist operands $N_1 \in \Lambda, \ldots, N_k \in \Lambda$ with $k \geq 0$ such that $M\, N_1\, \cdots\, N_k =_\beta N$.

This definition is the seminal one on page 87 of [Bar71].[2] In words, a closed term is solvable iff it converts to a β-nf when used in operator position at the top level. If the term is or has a β-nf then it is trivially solvable by choosing $k = 0$. Let us illustrate with examples that also explain the focus on closed terms. First, take the diverging closed term $\mathbf{\Omega}$ (an abbreviation of $\mathbf{\Delta\Delta}$, *i.e.* $\mathbf{\Omega} \equiv \mathbf{\Delta\Delta} \equiv (\lambda x.xx)(\lambda x.xx)$). A β-nf cannot be obtained from it no matter to how many or to which operands it is applied, *e.g.* $(\mathbf{\Delta\Delta})N_1 \cdots N_k =_\beta ((\lambda x.x\,x)\mathbf{\Delta})N_1 \cdots N_k =_\beta (\mathbf{\Delta\Delta})N_1 \cdots N_k =_\beta \ldots$ is an infinite loop. Terms like $\mathbf{\Omega}$ are operationally irrelevant. Now take the closed terms $\lambda x.x\,\mathbf{I}\,\mathbf{\Omega}$ and $\lambda x.x\,\mathbf{K}\,\mathbf{\Omega}$. Both terms diverge and yet both deliver a β-nf when applied to suitable operands. For example, $(\lambda x.x\,\mathbf{I}\,\mathbf{\Omega})\mathbf{K} =_\beta \mathbf{I}$, and $(\lambda x.x\,\mathbf{K}\,\mathbf{\Omega})\mathbf{K} =_\beta \mathbf{K}$. The β-nfs obtained from such diverging function terms are different, therefore they have different operational behaviour and cannot be equated. More precisely, a proof-theory with judgements $M = N$ can be obtained by taking the conversion proof-theory (if $M =_\beta N$ then $M = N$) and adding the equation $\lambda x.x\,\mathbf{I}\,\mathbf{\Omega} = \lambda x.x\,\mathbf{K}\,\mathbf{\Omega}$. However, this extended proof-theory is inconsistent because the false equation $\mathbf{I} = \mathbf{K}$ is then provable.

The focus on closed terms is because some open terms contain neutral terms (Section 2) that block applications [Wad76]. For example, take the neutral $x\,\mathbf{\Omega}$ and apply it to operands: $(x\,\mathbf{\Omega})N_1 \cdots N_k$. The conversion to β-nf is impossible because the diverging subterm $\mathbf{\Omega}$ is eventually converted due to the presence of the free variable x that blocks the application to the operands. (Similarly, in $x\,y\,\mathbf{\Omega}$ the neutral subterm $x\,y$ blocks the application.) However, a free variable stands for some operator, so substituting a closed operator for the variable may yield a solvable term. For example, substitute $\mathbf{K}\,\mathbf{I}$ for x and choose $k = 0$, then $\mathbf{K}\,\mathbf{I}\,\mathbf{\Omega} =_\beta \mathbf{I}$. Traditionally, open terms are defined as solvable iff the closed term resulting from such substitutions is solvable. We postpone the discussion to Section 3.2 where we

[2] The provisos $M \in \Lambda^0$ and $k \geq 0$ are implicit in the original definition due to the context of the thesis (closed-term models) and its subscript convention. They are explicit in later definitions [Bar72, Wad76, Bar84]. The order of existential quantifiers is immaterial. The original definition says '$M\,N_1 \cdots N_k$ has a β-nf' which as explained in Section 2 is the same as 'converts to a β-nf'. In [Bar84] the requirement on N is immaterially changed from being a β-nf to having a β-nf.

show that fully closing is excessive in λK. In Section 5 we show that it is counterproductive for defining solvability in λ_V. We conclude this section with the role of solvables in the development of a standard theory.

Solvable terms are approximations of totally defined terms. They are 'at least partially defined' [Wad76]. In contrast, unsolvable terms are 'hereditarily' [Bar71] or 'totally' [Wad76] undefined, and can be equated without loss of consistency. More precisely, given the set of equations $\mathcal{H}_0 = \{M = N \mid M, N \in \Lambda^0 \text{ unsolvable}\}$, a consistent extended proof-theory $\mathcal{H}$ results from adding $\mathcal{H}_0$'s equations as axioms to λK (*i.e.* $\mathcal{H} = \mathcal{H}_0 + \lambda K$) [Bar84]. A consistent extension where unsolvables are equated (*i.e.* contains $\mathcal{H}$) is called *sensible*. A consistent extension that does not equate solvables and unsolvables is called *semi-sensible*. There are standard models that satisfy $\mathcal{H}$, with unsolvables corresponding to the least elements of the model [Bar72, Bar84]. By extension, such models are called *sensible models* [Bar84, p.505]. Solvable terms can be characterised operationally: there is a reduction strategy of λK called 'head reduction' that converges iff the input term is solvable. (Solvability, like having β-nf, is semi-decidable.) More precisely, solvable terms exactly correspond to terms with hnf, and head reduction delivers a hnf iff the input term has one, diverging otherwise [Wad76, Bar84]. (In the technical jargon, head reduction is said to be *complete* with respect to hnf.)

3.1. **Other equivalent definitions of solvability.** There are two other equivalent definitions of solvability that use different equations [Bar72, Wad76, Bar84].

Definition 3.2 (SolI)**.** A term $M \in \Lambda^0$ is solvable in λK iff there exist operands $N_1 \in \Lambda$, $\ldots$, $N_k \in \Lambda$ with $k \geq 0$ such that $M\, N_1 \cdots N_k =_\beta \mathbf{I}$.

Definition 3.3 (SolX)**.** A term $M \in \Lambda^0$ is solvable in λK iff for all $X \in \Lambda$ there exist operands $N_1 \in \Lambda$, $\ldots$, $N_k \in \Lambda$ with $k \geq 0$ such that $M\, N_1 \cdots N_k =_\beta X$.

In words, a closed term is solvable iff it is convertible by application to the identity term or to any given term. Definition SolI is *de facto* in most presentations. These definitions are equivalent to SolN (capture the same set of solvables) because of two properties that hold in λK. The first is stated in the following lemma.

Lemma 3.4 (Lemma 4.1 in [Wad76])**.** *If $M \in \Lambda^0$ has a β-nf then for all $X \in \Lambda$ there exist operands $X_1 \in \Lambda$, $\ldots$, $X_k \in \Lambda$ with $k \geq 0$ such that $M\, X_1 \cdots X_k =_\beta X$.*

In words, a closed term with β-nf can be converted by application to any given term. This lemma is the link between SolN's *existential* property of having a β-nf and SolX's *universal* property of converting to any term. The shape of a β-nf is the key to this link, as the proof of the lemma illustrates.

Proof of Lemma 3.4. As explained in Section 2, a β-nf has the form $\lambda x_1 \ldots x_n.\, x\, N_1 \cdots N_m$ with $n \geq 0$, $m \geq 0$, and $N_1 \in \mathsf{NF}$, $\ldots$, $N_m \in \mathsf{NF}$. Since M is closed, its β-nf M' has $n > 0$ with x is one of x_i. Lemma 3.4 holds by choosing $k = n$, X_j arbitrary for $j \neq i$, and $X_i \equiv \mathbf{K}^m X$, with $\mathbf{K}^m$ the term that takes $m + 1$ operands and returns the first one. Thus, $M\, X_1 \cdots (\mathbf{K}^m X)_i \cdots X_n =_\beta X$ holds because $M =_\beta M'$ and $M'\, X_1 \cdots (\mathbf{K}^m X)_i \cdots X_n =_\beta (\mathbf{K}^m X) N'_1 \cdots N'_m =_\beta X$, with N'_i the result of substitutions on N_i. $\square$

The link between SolI and SolX is provided by the property that for all $X \in \Lambda$ the conversion $\mathbf{I}\,X =_\beta X$ holds [Bar84, p.171ff]. We provide here an explicit proof.

Lemma 3.5. *The solvability definitions* SolN*,* SolI*, and* SolX *are equivalent in* λK*.*

Proof. We use different operand symbols and subscripts to distinguish the equations:

$$M\,N_1\,\cdots\,N_k =_\beta N \qquad \textsc{SolN}$$
$$M\,Y_1\,\cdots\,Y_l =_\beta \mathbf{I} \qquad \textsc{SolI}$$
$$M\,Z_1\,\cdots\,Z_j =_\beta X \qquad \textsc{SolX}$$

We first prove SolX iff SolN: From SolX we prove SolN by choosing $k = j$, $N_i \equiv Z_i$, and X the β-nf N. Conversely, given SolN then $MN_1 \cdots N_k$ has a β-nf, so by Lemma 3.4 we have that forall $X \in \Lambda$ the conversion $M\,N_1 \cdots N_k\,X_1 \cdots X_{k'} =_\beta X$ holds. Then SolX follows by choosing $j = k + k'$, $Z_1 \equiv N_1$, $\ldots$, $Z_k \equiv N_k$, $Z_{k+1} \equiv X_1$, $\ldots$, $Z_j \equiv X_{k'}$.

We now prove SolX iff SolI: From SolX we prove SolI by choosing $l = j$, $Y_i \equiv Z_i$, and $X \equiv \mathbf{I}$. Conversely:

(a)	$M\,Y_1\,\cdots\,Y_l =_\beta \mathbf{I}$	SolI
(b)	$M\,Y_1 \cdots Y_l\,X =_\beta \mathbf{I}\,X$	by (ν) on (a) with any X
(c)	$\mathbf{I}X =_\beta X$	by (β)
(d)	$M\,Y_1 \cdots Y_l\,X =_\beta X$	by (τ) on (b),(c)

Then, SolX holds by choosing $j = l + 1$, $Z_1 \equiv Y_1$, $\ldots$, $Z_{j-1} = Y_l$, $Z_j \equiv X$. ◻

Bear in mind that although all definitions are equivalent, SolI and SolX are possible because of properties that hold in λK, and therefore SolI and SolX are secondary. As we shall see in Section 5, the anaologous in λ_V of Lemma 3.4 is not the case, nor are the analogous of SolI, SolX, and Lemma 3.5. Adapting SolI or SolX to that calculus will leave solvable terms behind.

3.2. **Open terms, and open and non-closing contexts.** Solvability has been typically extended to open terms by requiring at least one closed substitution instance or all closures of the open term[3] to be solvable [Wad76, Bar72, Bar84]. As we discussed in Section 3, neutral terms are the reason for closing. Substituting closed operators for the blocking free variables of neutrals may yield solvable terms. For example, $[\mathbf{K}\,\mathbf{I}/x](x\,\mathbf{\Omega}) \equiv \mathbf{K}\,\mathbf{I}\,\mathbf{\Omega}$ is trivially solvable according to SolN by choosing $k = 0$. Similarly, the closure $\lambda x.x\,\mathbf{\Omega}$ is solvable by choosing $k = 1$ and $N_1 \equiv \mathbf{K}\,\mathbf{I}$.

A traditional definition of solvability for open and closed terms uses a 'head context' to close the term before passing the operands [Wad76] (head contexts are defined on page 491 and solvability with head contexts on page 503).

Definition 3.6 (SolH)**.** A term $M \in \Lambda$ is solvable in λK iff there exists $N \in \mathsf{NF}^0$ and there exists a head context $\mathbf{H}[\ ] \equiv ((\lambda x_1 \ldots x_n.[\ ])C_1 \cdots C_n)N_1 \cdots N_k$ with $n \geq 0$, $k \geq 0$, $\mathrm{FV}(M) = \{x_1, \ldots, x_n\}$, $C_1 \in \Lambda^0$, $\ldots$, $C_n \in \Lambda^0$, and $N_1 \in \Lambda^0$, $\ldots$, $N_k \in \Lambda^0$ such that $\mathbf{H}[M] =_\beta N$.

[3] A closed substitution instance of M is a closed term resulting from substituting closed terms for all the free variables of M. A closure of M is a term $\lambda x_1 \ldots x_n.M$ such that $\mathrm{FV}(M) = \{x_1, \ldots, x_n\}$. Since different closures differ only on the order of prefix lambdas, if one closure is solvable then all other closures are too by passing the operands to the closure in the appropriate order. Substitutions and closures are connected by the β-rule.

In words, the head context forces the closed C_i to be substituted for all the free variables (if there are any) of the term placed within the hole. The resulting closed substitution instance is then at the top-level operator position where it is applied to the closed N_i operands. The top-level operator position is a 'head' position (Section 2), hence the name of the context. Since $\mathbf{H}[\ ]$ is a *closed and closing* context, the β-nf N has to be closed too. In [PR99], SolH and SolI are combined and the conversion is $\mathbf{H}[M] =_\beta \mathbf{I}$.

However, using a closed and closing context is excessive. The nature of solvability and the previous definitions do not require it. To begin with, an open term that is or has a β-nf is, by its very nature, solvable. For other open terms not every free variable has to be substituted, only the blocking ones that prevent solving the term. In all the previous definitions the N_i operands are arbitrary, and so the requirement that N_i are closed in $\mathbf{H}[\ ]$ can be dropped. Since in SolI both M and $\mathbf{I}$ are closed then the open N_i or their open subterms must be eventually discarded in the conversion to $\mathbf{I}$. But in SolN the β-nf N is arbitrary too, so not every open N_i operand or open subterm therein has to be discarded.

A less restrictive definition is perfectly possible:

Definition 3.7 (SolF)**.** A term $M \in \Lambda$ is solvable in λK iff there exists $N \in \mathsf{NF}$ and there exists a *function context* $\mathbf{F}[\ ] \equiv (\lambda x_1 \ldots x_n.[\ ])N_1 \cdots N_k$ with $n \geq 0$, $k \geq 0$, and $N_1 \in \Lambda$, $\ldots$, $N_k \in \Lambda$ such that $\mathbf{F}[M] =_\beta N$

This definition is closer to SolN. The function context can be *open* and *non-closing*: N and N_i may be open, and not every free variable of M need be substituted. For example, $x\,\mathbf{\Omega}$ is solved by the open function context $(\lambda x.[\ ])(\mathbf{K}\,N)$ where N is an open β-nf. And $x\,y\,\mathbf{\Omega}$ is solved by the non-closing function context $(\lambda x.[\ ])\mathbf{K}$ which does not close y.

Lemma 3.8 (Generalisation of Lemma 3.4)**.** *If $M \in \Lambda$ has a β-nf then for all $X \in \Lambda$ there exists a function context $\mathbf{F}[\ ]$ such that $\mathbf{F}[M] =_\beta X$.*

Proof. The β-nf of M has the form $\lambda x_1 \ldots x_n.x\,N_1 \cdots N_m$ with $n \geq 0$, $m \geq 0$ and $N_1 \in \mathsf{NF}$, $\ldots$, $N_m \in \mathsf{NF}$. If $x \in \mathrm{FV}(M)$ the lemma holds by choosing $\mathbf{F}[\ ] \equiv (\lambda x.[\ ])(\mathbf{K}^m X)X_1 \cdots X_n$ with X_i arbitrary and $\mathbf{K}^m$ the term that takes $m+1$ operands and returns the first one. If $x \notin \mathrm{FV}(M)$ then x is x_i for some i. The lemma holds by choosing $\mathbf{F}[\ ] \equiv [\ ]X_1 \cdots X_{i-1}(\mathbf{K}^m X)X_{i+1} \cdots X_n$. □

Let us note that the lemma also holds with the proviso relaxed to 'M has a hnf'.

Theorem 3.9. *In λK the solvability definitions* SolH *and* SolF *are equivalent.*

Intuitively, if we have a solving head context then we have a solving function context because function contexts subsume head contexts. And if we have a solving function context then we can construct a solving head context by carefully closing the former and the β-nf. The proof of Thm. 3.9 is not so short and we have put it in App. B with an accompanying example illustrating the construction of a solving head context from a solving function context.

As we shall see in Section 6, the analogous in λ_V of Thm. 3.9 is not the case. Adapting SolH to that calculus will leave solvable terms behind.

3.3. **Solvability and effective use.** As noted in [PR99] there is a more general definition of solvability that connects the notions of 'operational relevance' and 'effective use' of a term. A term is effectively used when it is eventually used as an operator. The term is operationally relevant iff it then delivers a final result, which in λK is a β-nf. In all previous solvability definitions, the term to solve is placed at the top-level operator position and thus it is effectively used. If it were placed at other positions then it may be eventually used as operator or it may be trivially used (discarded). If placed at an operand position that is never discarded, never gets to an operator position where it is applied to operands, and is returned as the final result, then the term is effectively used. It is as if the term were placed within an empty function context. Thus, a final result is in operator position, is effectively used, and is operationally relevant.

An unsolvable term cannot be effectively used to deliver a β-nf: 'unsolvable terms can never have a nontrivial effect on the outcome of a reduction' [Wad76, p.506]. More precisely, if M is unsolvable then for all X, $M\,X$ is unsolvable [Bar84, Cor. 8.34]. Unsolvable terms that are not effectively used are generic: they can be substituted by arbitrary terms. This is formalised by the so-called Genericity Lemma. The following statement of the Lemma is a combination of the versions in [Bar84, Prop. 14.3.24] and [Wad76, Cor. 5.5] (both collected in App. C for ease of reference). These versions use arbitrary contexts $\mathbf{C}[\;]$ because $\mathbf{C}[M]$ is more general than $M\,X$. The latter is a particular case of the former for $\mathbf{C}[\;] \equiv [\;]\,X$. With the context, the term plugged into the hole may eventually appear in operator position.

Lemma 3.10 (Genericity Lemma)**.** *Let $M \in \Lambda$ and $N \in \mathsf{NF}$. M is unsolvable in λK implies that for all contexts $\mathbf{C}[\;]$, if $\mathbf{C}[M] =_\beta N$ then for all $X \in \Lambda$ it is the case that $\mathbf{C}[X] =_\beta N$. In formal logic:*

$$M \textit{ unsolvable} \Rightarrow (\forall \mathbf{C}[\;].\, \mathbf{C}[M] =_\beta N \Rightarrow (\forall X \in \Lambda.\, \mathbf{C}[X] =_\beta N))$$

In words, if plugging an unsolvable term in a given arbitrary context converts to a β-nf then plugging any other term also converts to that β-nf. The unsolvable is not used effectively in the context. Although the lemma is stated as an implication, it is actually an equivalence because the negation of the consequent is a necessary condition for 'M solvable' by the SolF definition of solvability. Clearly, if M is solvable then there exists $\mathbf{C}[\;] \equiv \mathbf{F}[\;]$ such that $\mathbf{F}[M] =_\beta N$, and by the shape of $\mathbf{F}[\;]$ it is not the case that for all $X \in \Lambda$, $\mathbf{F}[X] =_\beta N$. Take for instance $\mathbf{F}[\mathbf{\Omega}]$ which diverges. (Note that if M is solvable and $\mathbf{C}[M] =_\beta N$ holds then $\mathbf{C}[X] =_\beta N$ should not hold for terms X that are not convertible to M unless M is not effectively used in $\mathbf{C}[\;]$.)

The lemma is a definition of solvability when read as the inverse equivalence:

$$M \textit{ solvable} \Leftrightarrow (\exists \mathbf{C}[\;].\, \mathbf{C}[M] =_\beta N \wedge \neg(\forall X \in \Lambda.\, (\mathbf{C}[X] =_\beta N)))$$

The following definition simply moves N to the formula from the proviso.

Definition 3.11 (SolC)**.** A term $M \in \Lambda$ is solvable in λK iff there exists a context $\mathbf{C}[\;]$ such that $\mathbf{C}[M] =_\beta N$ for some $N \in \mathsf{NF}$ and not for all $X \in \Lambda$ it is the case that $\mathbf{C}[X] =_\beta N$.

In words, M solvable means there exists a context that uses M effectively to deliver a β-nf. Function contexts are just one possible type of context applicable in SolC.

4. Call-by-value and pure λ_V

In call-by-value functional programming languages, the evaluation of application expressions $e_1\,e_2$ can be broadly described in 'big-step' fashion as follows. The operator expression e_1 is first evaluated to a 'value' v_1 where 'value' means here a first-class final result of the language. Functions are first-class values in such languages and their bodies are compiled, not evaluated. (In the SECD machine, the corresponding abstraction is not reduced, SECD reduction is 'weak', meaning it does not 'go under lambda'.) The operand expression e_2 is next evaluated to a value v_2. Finally, the result of passing v_2 to v_1 is evaluated. Evaluation diverges at the point where the first sub-evaluation diverges. Evaluation may halt due to a run-time error. The order of evaluation matters w.r.t. the point of divergence or halting.[4]

In pure λ_V, an application $M\,N$ can be reduced to β_V-nf in several ways with the restriction that if M is an abstraction or reduces to an abstraction, say $\lambda x.B$, and N is a value or reduces to a value, say V, then the redex application $(\lambda x.B)V$ can be reduced in one step to $[V/x]B$, with reduction continuing on the result of the substitution. Either the abstraction $\lambda x.B$, or the value V, or both may be fully reduced in β_V-nf depending on the reduction strategy. If N is not a value or does not reduce to a value then $(\lambda x.B)N$ is a neutral which may only be reduced to a stuck. Abstractions are values, and so are free variables because they range over values as discussed in more detail below. Terms can be open, reduction may 'go under lambda' with free variables possibly occurring within that scope, and final results are not values but β_V-nfs.

The rationale behind the restricted reduction/conversion and the definition of values is not merely to model call-by-value but to uphold confluence which is a *sine qua non* property of the calculus because it upholds the consistency of the proof-theories. Intuitively, the rationale is *to preserve confluence by preserving potential divergence*. To preserve confluence, applications cannot be passed as operands unless given the opportunity to diverge first. This point is fundamental to understanding our approach to solvability for λ_V and so the rest of this section elaborates it.

In λ_V the reduction relation $\rightarrow^*_{\beta_V}$ is confluent [HS08, App. A2]. Confluence applies even for terms without β_V-nf. The implication is that terms have at most one β_V-nf, and so terms with different β_V-nf are not β_V-reducible/convertible. Not every β_V-reduction/conversion is provable and the reduction/conversion proof-theory is consistent. The proof of confluence requires substitutivity which is the property that reduction/conversion is preserved under substitution, *e.g.* if $M =_{\beta_V} N$ then $[L/x]M =_{\beta_V} [L/x]N$. In λ_V, permissible operands and subjects of substitutions cannot be applications, whether arbitrary or in β_V-nf. Otherwise, substitutivity and confluence would not hold. (This is explained in [Plo75, p.135-136], see App. D for a detailed discussion.) Substitutivity requires the proviso $L \in \mathsf{Val}$ which explains why free variables are members of Val, namely, because they range over members of Val.

For illustration, the neutral $x\,\mathbf{\Delta}$ cannot be passed in applications such as $(\lambda x.y)(x\,\mathbf{\Delta})$ because whether it diverges depends on what value x is. For example, substituting the value $\mathbf{I}$ for x yields $(\lambda x.y)(\mathbf{I}\,\mathbf{\Delta})$ which converges to y. But substituting the value $\mathbf{\Delta}$ for x yields $(\lambda x.y)(\mathbf{\Delta}\mathbf{\Delta})$ which diverges. Applications must be given the opportunity to diverge before being passed, not only to model call-by-value but because whether a neutral converges depends on which values are substituted for its free variables. The same goes for stucks: in the above examples $x\,\mathbf{\Delta}$ is actually a stuck.

4.1. **Neutrals, stucks, and sequentiality.** Before moving on we must recall that the nesting and order of neutrals confer the sequentiality character to λ_V. Take the following

[4]Some languages prefer to evaluate e_2 before e_1, or instead of binary applications consider applications with multiple operands, evaluating the latter in left-to-right or right-to-left fashion. Some languages eschew divergence and run-time errors by means of a strong but yet expressive type discipline.

neutrals adapted from [Mil90, p.25] and assume V and W are closed values:

$$\begin{array}{lcl} \mathbf{L}_1 & \equiv & (x\,V)(y\,W) \\ \mathbf{L}_2 & \equiv & (\lambda z.z(y\,W))(x\,V) \\ \mathbf{L}_3 & \equiv & (\lambda z.(x\,V)z)(y\,W) \end{array}$$

Respectively substituting values X and Y for x and y we get:

$$\begin{array}{lcl} \mathbf{L}'_1 & \equiv & (X\,V)(Y\,W) \\ \mathbf{L}'_2 & \equiv & (\lambda z.z(Y\,W))(X\,V) \\ \mathbf{L}'_3 & \equiv & (\lambda z.(X\,V)z)(Y\,W) \end{array}$$

If all $\mathbf{L}'_i$ have β_V-nf then it is the same and the instances are convertible. But different reduction sequences differ on the order in which $(X\,V)$ and $(Y\,W)$ are reduced in $\mathbf{L}'_2$ and $\mathbf{L}'_3$ and thus on which order is the same as in $\mathbf{L}'_1$. Under SECD reduction the $(X\,V)$ is reduced before $(Y\,W)$ in $\mathbf{L}'_1$ and $\mathbf{L}'_2$ whereas in $\mathbf{L}'_3$ the order is reversed. However, in a reduction sequence where abstraction bodies are reduced before operands then $(X\,V)$ is reduced before $(Y\,W)$ in $\mathbf{L}'_1$ and $\mathbf{L}'_3$ whereas in $\mathbf{L}'_2$ the order is reversed.[5]

Suppose operators and operands were reduced in separate processors. If x is instead substituted by a value X such that $X\,V$ converts to a stuck, then we can tell on which processor reduction got stuck first. If we substitute y for a value Y such that $Y\,W$ diverges then one processor would diverge whereas the other would get stuck.

As another example consider the following terms where now V and W are closed values in β_V-nf:

$$\begin{array}{lcl} \mathbf{L}_4 & \equiv & (\lambda z.V\,W)(xx) \\ \mathbf{L}_5 & \equiv & (\lambda z.(\lambda y.y\,W))(xx)V \end{array}$$

Observe that $\mathbf{L}_5$ is a β_V-nf whereas $\mathbf{L}_4$ is not. If $V\,W$ converges to a β_V-nf N then $(\lambda z.N)(xx)$ is a β_V-nf different from $\mathbf{L}_5$. If $V\,W$ diverges then $\mathbf{L}_4$ diverges but $\mathbf{L}_5$ does not (it is a β_V-nf). Let us now play with substitutions for the blocking variable x. Substitute in $\mathbf{L}_4$ and $\mathbf{L}_5$ a closed value X for x such that XX converges to a value:

$$\begin{array}{lcl} \mathbf{L}'_4 & \equiv & (\lambda z.V\,W)(XX) \\ \mathbf{L}'_5 & \equiv & (\lambda z.(\lambda y.y\,W))(XX)V \end{array}$$

In the case where $V\,W$ converges to a β_V-nf N then $\mathbf{L}'_4$ and $\mathbf{L}'_5$ converge to N, but in $\mathbf{L}'_4$ whether $(V\,W)$ is reduced before (XX) depends on whether the reduction strategy goes first under lambda, whereas in $\mathbf{L}'_5$ the term (XX) is reduced first with that same strategy. In the case where $V\,W$ diverges, whether $\mathbf{L}'_4$ diverges before reducing (XX) also depends on whether the reduction strategy goes first under lambda, whereas in $\mathbf{L}'_5$ the term (XX) is reduced first with that same strategy. Thus, $\mathbf{L}_4$ and $\mathbf{L}_5$ are operationally distinguishable. For example, the concrete instantiations $(\lambda z.\mathbf{II})(xx)$ and $(\lambda z.(\lambda y.y\,\mathbf{I}))(xx)\mathbf{I}$ are operationally distinguishable (here $V \equiv \mathbf{I}$, $W \equiv \mathbf{I}$, and $\mathbf{II}$ converges to a β_V-nf).

Neutral terms differ on the point at which a free variable pops up, that is, on the point of potential divergence. Stucks are only fully reduced neutrals that keep that point of divergence. Terms with neutrals that may convert to the same β_V-nf when placed in the same closed context are nonetheless operationally distinguishable when placed in an open context. And the choice of substitutions for the blocking free variables is important. Keep this in mind when reading the following sections.

[5] In this example we have in mind a complete reduction sequence. There is a complete reduction strategy of λ_V that goes under lambda in such 'spine' fashion (Section 7.1).

5. An overview of v-solvability

Solvability for λ_V is first studied in [PR99] where a definition of v-solvable term is introduced which adapts to λ_V the SolI definition of solvability for λK.

Definition 5.1 (v-solvability)**.** A term M is v-solvable in λ_V iff there exist closed values $N_1 \in \mathsf{Val}^0, \ldots, N_k \in \mathsf{Val}^0$ with $k \geq 0$ such that $(\lambda x_1 \ldots x_n.M)N_1 \cdots N_k =_{\beta_V} \mathbf{I}$ where $\mathrm{FV}(M) = \{x_1, \ldots, x_n\}$.

The definition can be stated alternatively in terms of the head contexts of Section 3.2 by requiring the C_i's and N_i's in the head contexts to be closed values instead of closed terms. The provisos $N_i \in \mathsf{Val}^0$ could have been omitted because they are required by the β_V-conversion to the closed value $\mathbf{I}$. In line with the discussion in Section 3.2, an open head context whose free variables are discarded in the conversion can also be used, and so it is in [AP12, p.9].

Adapting SolI to λ_V instead of SolN is surprising because, as anticipated in Section 3.1, the two properties that justify the equivalence between SolI and SolN in λK do not hold in λ_V. (And as discussed in Section 3.2, the use of a closed and closing head context is excessive, but more on this below.)

First, $\mathbf{I}\,X =_{\beta_V} X$ holds iff X has a value. Assuming such proviso, the SolX equivalent of Def. 5.1 is that a term is v-solvable iff it is convertible by application not to any term but to any *value*. Indeed, if M is v-solvable then $(\lambda x_1 \ldots x_n.M)N_1 \cdots N_k =_{\beta_V} \mathbf{I}$ and, by compatibility, $(\lambda x_1 \ldots x_n.M)N_1 \cdots N_k\, X =_{\beta_V} \mathbf{I}\, X$ for any $X \in \Lambda$. The conversion $(\lambda x_1 \ldots x_n.M)N_1 \cdots N_k\, X =_{\beta_V} X$ is obtained by transitivity with $\mathbf{I}\, X =_{\beta_V} X$ iff X has a value.

Second, the adaptation of Lemma 3.4 to λ_V does not hold.

Statement 5.2 (Adapts Lemma 3.4 to λ_V)**.** If $M \in \Lambda^0$ has a β_V-nf then for all $X \in \Lambda$ there exist operands $X_1 \in \Lambda, \ldots, X_k \in \Lambda$ with $k \geq 0$ such that $M\, X_1 \cdots X_k =_{\beta_V} X$.

This statement does not hold even with X_i and X values, whether open or closed. The controversial term $\mathbf{U} \equiv \lambda x.(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}$ mentioned in [PR99] is one possible counter-example. (Notice the close resemblance to the term $\mathbf{L}_5$ in Section 4.1.) This term is a closed value and a β_V-nf. It is an abstraction with a stuck body. There is no operand X_1, let alone further operands, that lets us convert $\mathbf{U}$ to any given X whether arbitrary, a value, or a closed value.

Suppose $X_1 \in \mathsf{Val}^0$. Then $\mathbf{U}\, X_1$ converts to $(\lambda y.\mathbf{\Delta})(X_1\,\mathbf{I})\mathbf{\Delta}$. If $(X_1\,\mathbf{I})$ diverges then the latter diverges. If $(X_1\,\mathbf{I})$ converts to a closed value V then $(\lambda y.\mathbf{\Delta})\,V\,\mathbf{\Delta}$ converts to $\mathbf{\Delta\Delta} \equiv \mathbf{\Omega}$ which diverges. However, $\mathbf{U}\, X_1$ converts to a β_V-nf if $(X_1\,\mathbf{I})$ converts to a stuck. But the shape of the β_V-nf, namely $(\lambda y.\mathbf{\Delta})(\ldots)\mathbf{\Delta}$, is determined by the shape of $\mathbf{U}$. Ony the concrete β_V-nf obtained depends on the choice of *open* value X_1 that generates the stuck. For example: $X_1 \equiv \lambda x.z\,\mathbf{I}$ leads to $(\lambda y.\mathbf{\Delta})(z\,\mathbf{I})\mathbf{\Delta}$ whereas $X_1 \equiv \lambda x.(\lambda x.x)(z\,\mathbf{K})$ leads to $(\lambda y.\mathbf{\Delta})((\lambda x.x)(z\,\mathbf{K}))\mathbf{\Delta}$, etc. We cannot send $\mathbf{U}$ to any arbitrary β_V-nf. The only degree of freedom is X_1.

The term $\mathbf{U}$ is controversial because, although a β_V-nf, it is considered operationally equivalent to $\lambda x.\mathbf{\Omega}$ in [PR99]. Certainly, $\mathbf{U}\, X_1$ and $(\lambda x.\mathbf{\Omega})X_1$ diverge for all $X_1 \in \mathsf{Val}^0$. But as illustrated in the last paragraph, $\mathbf{U}$ and $\lambda x.\mathbf{\Omega}$ are operationally distinguishable in an open context: there exists $X_1 \in \mathsf{Val}$ such that $\mathbf{U}\, X_1$ converts to a β_V-nf, but there is no $X_1 \in \mathsf{Val}$ such that $(\lambda x.\mathbf{\Omega})X_1$ converts to a β_V-nf. The difference between $\mathbf{U}$ and $\lambda x.\mathbf{\Omega}$ is illustrated by the old chestnut 'toss a coin, heads: you lose, tails: toss again'. We can pass a value to $\mathbf{U}$ to either diverge immediately or to postpone divergence, but this choice is not possible for $\lambda x.\mathbf{\Omega}$ which diverges whatever value passed. And since $\mathbf{U}$ is a β_V-nf, it should be by definition solvable in λ_V.

The restriction of operands to elements of Val^0 is natural in the setting of SECD's weak reduction of closed terms where final results are closed values. This is the setting considered in [PR99] where the proof-theory is not λ_V's but consists of equations '$M = N$ iff M and N are operationally equivalent under SECD reduction'. However, v-solvability (Def. 5.1) is defined for λ_V and its proof-theory, not the alternative pure-SECD-theory. Several problems arise. First, closed values such as $\mathbf{U}$ and $\lambda x.\mathbf{\Omega}$ which are definite results of SECD are v-unsolvable, so v-solvability is not synonymous with operational relevance. Second, there is a v-unsolvable $\mathbf{U}$ that is nevertheless a β_V-nf of λ_V. As discussed in the introduction, the blame is mistakenly put on λ_V, not on v-solvability.

The operational relevance of final results is partly recovered in [PR99, p.21] by adapting to v-unsolvables the notion of order of a term [Lon83, Abr90] in the following fashion: a v-unsolvable M is of order n iff it reduces under the so-called 'inner machine' to $\lambda x_1 \ldots x_n.B$ where n is maximal. That is, M reduces to a value with n lambdas. If M has order 0 then it does not reduce to a value. If M has order $n > 0$ then M accepts $n-1$ operands and reduces to a value. For example, $\mathbf{\Omega}$ has order 0, and $\lambda x.\mathbf{\Omega}$ and $\mathbf{U}$ have order 1. With this notion of order, definite results include v-solvables and v-unsolvables of order $n > 0$. This corresponds with the behaviour of SECD. The v-unsolvables of order 0 denote the least element of the model H of [EHR92] and can be equated without loss of consistency.

However, the 'inner machine' is a call-by-value reduction strategy of λK. It performs β-reduction, reducing redexes when the operand is not a value. Furthermore, v-unsolvables of order $n > 0$, which according to [PR99] are operationally irrelevant because no arbitrary result can be obtained from them, are definite results. These v-unsolvables cannot be consistently equated [PR99] and thus the model H is not sensible. Moreover, it is not semi-sensible since some v-solvables can be equated to v-unsolvables (Thm. 5.12 in [PR99, p.22]). Finally, the operational characterisation of v-solvability, namely having a v-hnf, is given by the so-called 'ahead machine' which is also a reduction strategy of λK, not of λ_V.

The reason why v-solvability does not capture operational relevance in λ_V is because it is based on SolI which requires the *universally* (any X) quantified Lemma 5.1 to hold. The solution lies in adapting to λ_V the *existentially* (has some β_V-nf) quantified SolN definition with open and non-closing contexts. As we shall see, there are two ways to solve a term in λ_V. One is to apply it to suitable values to obtain any given value (or closed value as in v-solvability). We call this to *transform* the application. Another is to pass suitable values to obtain some β_V-nf. We call this to *freeze* the application. Terms like $\mathbf{U}$ cannot be transformed but frozen.

In [RP04, p.36] it is the open body of $\mathbf{U}$, *i.e.* $\mathbf{B} \equiv (\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}$, what is considered operationally equivalent to $\mathbf{\Omega}$. Now, $\mathbf{B}$ is not a value, but it is a β_V-nf, a definite result of λ_V. The difference between $\mathbf{B}$ and $\mathbf{\Omega}$ lies in the value substituted for x. The intuition is best expressed using the following experiment paraphrased from [Abr90, p.4]:

> Given [an arbitrary] term, the only experiment of depth 1 we can do is to evaluate [weakly] and see if it converges to some abstraction [or to some neutral subsequently closed to some abstraction] $\lambda x.M_1$. If it does so, we can continue the experiment to depth 2 by supplying [an arbitrary value N_1 that may be open] as input to M_1, and so on. Note that what the experimenter can observe at each stage is only the *fact* of convergence, not which term lies

under the abstraction. [Note that the term *reports* the need to provide a value for the blocking free variable by closing the neutral to an abstraction.]

6. Introducing λ_V-solvability

We have seen that terms like the $\mathbf{L}'_i$ of Section 4.1, or $\mathbf{U}$ and $\lambda x.\mathbf{\Omega}$ in the previous section, are operationally distinguishable in open contexts. We thus define solvability in λ_V by adapting SolF to that calculus.

Definition 6.1 (SolF$_V$)**.** A term $M \in \Lambda$ is solvable in λ_V iff there exists $N \in \mathsf{VNF}$ and there exists a function context $\mathbf{F}[\ ]$ such that $\mathbf{F}[M] =_{\beta_V} N$.

Notice that operands in function contexts may be values if so wished. Hereafter we abbreviate 'M is solvable in λ_V' as 'M is λ_V-solvable'.

The set of λ_V-solvables is a proper superset of the union of the set of terms with β_V-nf and the set of v-solvables. A witness example is $\mathbf{T}_1 \equiv (\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}(x(\lambda x.\mathbf{\Omega}))$. This term has no β_V-nf. This term is not v-solvable: there is no closed and closing head context sending $\mathbf{T}_1$ to $\mathbf{I}$, or to a closed value, or to a closed β_V-nf. However, the function context $\mathbf{F}[\ ] \equiv (\lambda x.[\ ])(\lambda x.z\,\mathbf{I})$ sends $\mathbf{T}_1$ to the β_V-nf $(\lambda x.\mathbf{\Delta})(z\,\mathbf{I})\mathbf{\Delta}(z\,\mathbf{I})$. Therefore $\mathbf{T}_1$ is λ_V-solvable.

Notice that $\mathbf{T}_1$ has $\mathbf{B}$ as subterm, with the blocking variable x of $\mathbf{B}$ the same blocking variable of the neutral $x(\lambda x.\mathbf{\Omega})$. The use of the same blocking variable illustrates that the function context in SolF$_V$ has to be open. There is no closed function context (nor head context) sending $\mathbf{T}_1$ to a β_V-nf since substituting a closed value for x would make $\mathbf{B}$ diverge. In contrast, the free variable z in $\mathbf{F}[\ ]$ above is key to produce a stuck. We anticipated in Section 3.2 that adapting SolH to λ_V leaves solvable terms behind. The terms $\mathbf{U}$ and $\mathbf{T}_1$ are two witness examples.

We now connect λ_V-solvability and operational relevance with effective use in λ_V, as we did for λK in Section 3.3. To this end we adapt to λ_V the notion of 'order of a term' [Lon83].

Definition 6.2 (Order of a term in λ_V)**.** A term $M \in \Lambda$ is of order 0 iff there is no N such that $M =_{\beta_V} \lambda x.N$. A term $M \in \Lambda$ is of order $n+1$ iff $M =_{\beta_V} \lambda x.N$ and N is of order n. In the limit, *i.e.* when a maximum natural k does not exist such that $M =_{\beta_V} \lambda x_1 \ldots x_k.N$, we say M is of order ω.

This definition differs from the one in [PR99, p.21]. The latter is for v-unsolvables and uses the 'inner machine' which is a reduction strategy of λK (Section 5). Ours is for arbitrary terms (not just λ_V-unsolvables) and uses β_V-conversion.

The order of a term is an ordinal number that comprises the finite ordinals (*i.e.* the naturals) and the first limit ordinal ω. An example of a term of order ω is $\mathbf{Y}\,\mathbf{K}$ where $\mathbf{Y}$ is Curry's fixed-point combinator (see Prop. 2.7.(iv) in [Abr90, p.6] and Ex. 2 in [Wad76, p.502]). The term $\mathbf{Y}\,\mathbf{K}$ β_V-converts to $\lambda x_1 \ldots x_k.\mathbf{Y}\,\mathbf{K}$ with k arbitrarily large. Notice that a term of order ω has no β_V-nf and is λ_V-unsolvable.

With this notion of order at hand we can now state our version of SolC for λ_V.

Definition 6.3 (SolC$_V$)**.** A term $M \in \Lambda$ of order n is solvable in λ_V iff there exists a context $\mathbf{C}[\ ]$ such that $\mathbf{C}[M] =_{\beta_V} N$ for some $N \in \mathsf{VNF}$, and not for all $X \in \Lambda$ of order $m \geq n$ it is the case that $\mathbf{C}[X] =_{\beta_V} N$.

Note that $X \in \mathsf{Val}$ is allowed by the definition.

As was the case in λK (Section 3.3), the piece that lets us obtain SolC_V from SolF_V is a genericity lemma which in λ_V has to take into account the order of λ_V-unsolvables.

Lemma 6.4 (Partial Genericity Lemma)**.** *Let $M \in \Lambda$ be of order n and $N \in \mathsf{VNF}$. M is λ_V-unsolvable implies that for all contexts $\mathbf{C}[\ ]$, if $\mathbf{C}[M] =_{\beta_V} N$ then for all $X \in \Lambda$ of order $m \geq n$ it is the case that $\mathbf{C}[X] =_{\beta_V} N$.*

We postpone the proof to Section 7 and focus here on the intuitions. The lemma tells us that λ_V-unsolvables of order n are *partially* generic, *i.e.* they are generic for terms of order $m \geq n$. A λ_V-solvable can be used effectively to produce a β_V-nf therefore λ_V-solvability is synonymous with operational relevance. However, not all λ_V-unsolvables are totally undefined. Only λ_V-unsolvables of order 0 are totally undefined. A λ_V-unsolvable of order n cannot be used effectively to produce a β_V-nf, but it can be used trivially (discarded) after receiving at most $n-1$ operands. Hence, it is partially defined.

For example, take $M \equiv \lambda x.\lambda y.\mathbf{\Omega}$. This term is λ_V-unsolvable of order 2. The context $\mathbf{C}[\ ] \equiv (\lambda x.(\lambda y.\mathbf{I})(x\,\mathbf{\Delta}))[\ ]$ uses M first 'administratively' (*i.e.* passes $\mathbf{\Delta}$ to it) and then 'trivially' (*i.e.* discards the result) such that $\mathbf{C}[M] =_{\beta_V} \mathbf{I}$. Replacing M with a totally undefined term like $\mathbf{\Omega}$ would make $\mathbf{C}[\mathbf{\Omega}]$ diverge. But since $\mathbf{C}[\ ]$ uses M only up to passing one argument, M could be replaced by any term X of order 2 and still $\mathbf{C}[X] =_{\beta_V} \mathbf{I}$.

The Partial Genericity Lemma is stated as an implication but, as was the case with Lemma 3.10, it is an equivalence. Clearly, if M is λ_V-solvable then there exists $\mathbf{C}[\ ] \equiv \mathbf{F}[\ ]$ such that $\mathbf{F}[M] =_{\beta_V} N$, and by the shape of $\mathbf{F}[\ ]$ it is not the case that for all $X \in \Lambda$ of order $m \geq n$, $\mathbf{F}[X] =_{\beta_V} N$. Take for instance $\mathbf{F}[\lambda x_1 \ldots x_m.\mathbf{\Omega}]$ which diverges. Stated as an equivalence, the Partial Genericity Lemma coincides with SolC_V when read in the inverse.

Pure λ_V still has 'functional character' [CDV81, EHR92] but its notion of operational relevance takes into account trivial uses of terms that occur inside operands of other terms up to administratively passing them a number of operands. More precisely, if a term occurs inside the operand of another term then it has 'negative polarity'. Otherwise it has 'positive polarity'. The import of polarity for operational relevance is inherent to the duality between call-by-name and call-by-value [CH00]. Subterms with positive polarity are used effectively. Subterms with negative polarity may or may not occur eventually with positive polarity, in which case they would respectively be used effectively or trivially (perhaps after receiving some operands). The partially generic terms may only be used trivially (up to order n) to produce a β_V-nf if they occur with negative polarity.

Partially generic terms can be equated attending to their order without loss of consistency. More precisely, given the set

$$\mathcal{V}_0 = \{M = N \mid M, N \in \Lambda^0 \text{ are } \lambda_V\text{-unsolvables of the same order}\}$$

a consistent extended proof-theory $\mathcal{V}$ results from adding $\mathcal{V}_0$'s equations as axioms to λ_V (*i.e.* $\mathcal{V} = \mathcal{V}_0 + \lambda_V$). The consistency of $\mathcal{V}$ is proved in Section 8. We say that a consistent extension where λ_V-unsolvables of the same order are equated (*i.e.* contains $\mathcal{V}$) is *ω-sensible*.

Since the operational experiments that we have in mind (Sections 4.1 and 5) distinguish sequentiality features, no ω-sensible *functional models* (*e.g.*, models that are solution to the domain equation $D \cong [D \to_\perp D]$ for strict functions) seem to exist. However, we conjecture the existence of ω-sensible models that may resemble the 'sequential algorithms' of [BC82]. The notion of operational relevance in λ_V that we advocate calls for increased 'separating capabilities' (in the spirit of [Cur07]) that ω-sensible models would exhibit. Such capabilities are not present in existing models for 'lazy' call-by-value (*e.g.*, the model H in [EHR92] based on the solution to the domain equation $D \cong [D \to_{\perp} D]_{\perp}$ for lifted strict functions). We also conjecture that existing functional models could be constructed from ω-sensible models via some quotient that blurs the differences in sequentiality.

As for the operational characterisation of λ_V-solvables, that is, a reduction strategy of λ_V that terminates iff the input term is λ_V-solvable, we postpone the discussion to Section 7.5.

7. Towards the Partial Genericity Lemma

Our proof of the Partial Genericity Lemma is based on the proof of λK's Genericity Lemma presented in [BKC00] that uses origin tracking. Given a reduction sequence $M \to_{\beta} \ldots \to_{\beta} N$ with $N \in \mathsf{NF}$, origin tracking traces the symbols in N back to a prefix of M (*i.e.* a 'useful' part) which is followed by a lower part (*i.e.* the 'garbage') that does not affect the result N. The tracking mechanism employs a refinement of Lévy-labels [Lév75].

In our case the reduction sequence is $M \to_{\beta_V} \ldots \to_{\beta_V} N$ with $N \in \mathsf{VNF}$. Instead of tracking the symbols in N back to the the useful part in M, we mark as garbage a predefined subterm in M, namely, the λ_V-unsolvable of order n that we want to test for partial genericity. We track this subterm forwards and check that it is discarded in the reduction sequence before passing n operands to it. To this end we need two main ingredients: (i) a reduction strategy that is complete with respect to β_V-nf (Section 7.1) and (ii) a tracking mechanism that keeps count of the number of operands that are passed to a predefined subterm (Section 7.2). We prove that the predefined term is discarded by the complete reduction strategy after receiving at most $n-1$ operands (Section 7.3). Confluence allows us to generalise from the reduction strategy to any reduction sequence ending in β_V-nf.

7.1. **Value normal order.** The first ingredient we need is a reduction strategy of reference that is complete with respect to β_V-nf. We define one such strategy and call it *value normal order* because we have defined it by adapting to λ_V the results in [BKKS87] relative to the complete *normal order* strategy of λK mentioned in Section 2. Those results are collected in App. E for ease of reference. In this section we introduce their analogues for λ_V. The unacquainted reader may find it useful to read App. E and this section in parallel.

We advance that value normal order is not quite the same strategy as the complete reduction strategy of λ_V named $\to^{p}_{\Gamma}$ that is obtained as an instantiation of the 'principal reduction machine' of [RP04]. The latter reduces the body and operator of a block in right-to-left fashion whereas value normal order uses the more natural left-to-right order (see Section 9 for details). This difference does not affect completeness because both strategies entail standard reduction sequences (a notion defined in [Plo75, p.137] for the applied λ_V and adapted to pure λ_V in Def. 7.8 below). For every λ_V reduction sequence from M to N, there exists a standard reduction sequence that starts at M and ends at N. A reduction strategy that entails standard reduction sequences and that arrives at a β_V-nf is complete. And standard reduction sequences are not unique (Section 7.4).

Normal order can be defined as follows. The *active components* of a term [BKKS87, Def. 2.3] (*i.e.* the maximal subterms that are not in hnf) are considered in left-to-right fashion and reduced by *head reduction* [Bar84, Def. 8.3.10]. At the start, the input term is the only active component if it is not a hnf. Once a hnf is reached its active components occur as subterms inside a 'frozen' β-nf context. Every time the hnf of an active component is reached, the subsequent active components in it (if any) are recursively considered in left-to-right-fashion. We define value normal order by adapting this pattern to λ_V. In particular, we adapt the definition of needed redex, of active component, and of head reduction, whose analogue we have called 'chest reduction' following the convention of [BKKS87, Sec. 4] of considering the abstract syntax tree of a term and an anatomical analogy for terms.

First we adapt the notion of needed redex [BKKS87, p.212] to λ_V:

Definition 7.1 (Needed redex in λ_V)**.** Let $M \in \Lambda$ and R a β_V-redex in M. R is needed iff every reduction sequence of M to β_V-nf contracts (some residual of) R.

The *chest* and *ribcage* of a term provide progressively better approximations to the set of needed β_V-redexes of a term. The chest of the term contains the head of the term and the outermost *ribs*, that is, all the nodes connected by application nodes to the head of the term save for the *rib ends*. The rib ends are the nodes descending through lambda nodes from the ribs. The ribcage of a term consists of the head spine and the ribs connected to the head spine, that is, all the nodes connected by application nodes to the head spine of the term save for the rib ends. Fig. 4 illustrates with an example that is further developed after the following formal definition of chest and ribcage.

In Def. 7.2 below we define the functions bv, ch, and rc. The last two underline respectively the chest and the ribcage of a term. Both rely on auxiliary bv related to call-by-value as explained further below.

Definition 7.2 (Chest and ribcage)**.** Functions ch and rc underline the chest and the ribcage of a term respectively.

$$
\begin{array}{rcl}
\mathrm{bv}(x) & = & \underline{x} \\
\mathrm{bv}(\lambda x.B) & = & \underline{\lambda x}.B \\
\mathrm{bv}(M\,N) & = & \mathrm{bv}(M)\mathrm{bv}(N) \\[4pt]
\mathrm{ch}(x) & = & \underline{x} \\
\mathrm{ch}(\lambda x.B) & = & \underline{\lambda x}.\mathrm{ch}(B) \\
\mathrm{ch}(M\,N) & = & \mathrm{bv}(M)\mathrm{bv}(N) \\[4pt]
\mathrm{rc}(x) & = & \underline{x} \\
\mathrm{rc}(\lambda x.B) & = & \underline{\lambda x}.\mathrm{rc}(B) \\
\mathrm{rc}(M\,N) & = & \mathrm{rc}(M)\mathrm{bv}(N)
\end{array}
$$

A β_V-redex is chest (resp. ribcage) if the outermost lambda of it is underlined by function ch (resp. rc).

Function bv underlines the outermost lambda of the β_V-redexes that are reduced by the call-by-value strategy of pure λ_V (Def. 7.3). This strategy differs from its homonym in [Plo75, p.136] which is for an applied version of the calculus. See [Fel87, Ses02, RP04] for details on the difference. The chest and ribcage β_V-redexes realise the idea that operands in applications must be reduced to a value.

As an example, consider the term whose abstract syntax tree is depicted in Fig 4. The chest (thick edges in the figure) is underlined in $\underline{\lambda x.(\lambda y}.y((\lambda z.M_1)x))\underline{x((\lambda t}.M_2)\underline{x})$. The ribcage (thick edges and dotted edges) is underlined in $\underline{\lambda x.(\lambda y.y((\lambda z.M_1)x})\underline{)x((\lambda t}.M_2)\underline{x})$. The subterms M_1 and M_2 are the rib ends. The subterms $(\lambda y.y((\lambda z.M_1)x))x$ and $(\lambda t.M_2)x$ are both chest and ribcage β_V-redexes. (The former is also a head and head-spine β-redex, and the latter is neither head nor head-spine.) The subterm $(\lambda z.M_1)x$ is a ribcage β_V-redex but it is neither a chest β_V-redex, nor a head or head spine β-redex.

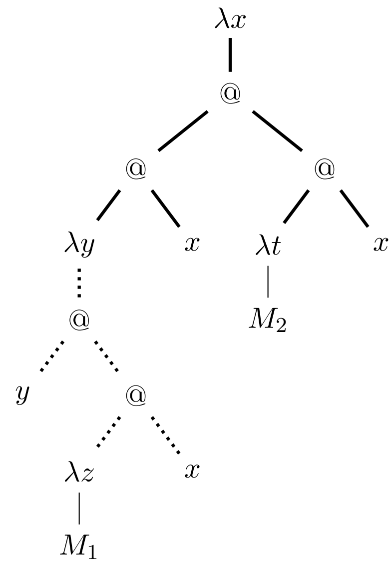

Figure 4: Chest (thick edges) and ribcage (thick edges and dotted edges) of the term $\lambda x.(\lambda y.y((\lambda z.M_1)x))x((\lambda t.M_2)x)$.

We now define call-by-value and chest reduction using a (context-based) reduction semantics [Fel87] which is a handy device for defining small-step reduction strategies. It consists of EBNF-grammars for terms, irreducible forms, and reduction contexts, together with a contraction rule for redexes within context holes. The reduction strategy is defined by the iteration of single-step reductions which consist of (i) uniquely decomposing the term into a reduction context plus a redex within the hole, (ii) contracting the redex within the hole and, (iii) recomposing the resulting term. The iteration terminates iff the term is irreducible.

Call-by-value is the strategy that contracts the leftmost β_V-redex that is not inside an abstraction [Fel87, p.42]. Chest reduction is the strategy that contracts the leftmost chest β_V-redex. Observe that the reduction contexts of chest reduction contain the reduction contexts of call-by-value.

Definition 7.3 (Call-by-value strategy)**.** The call-by-value strategy $\to_V$ is defined by the the following reduction semantics:

$$\begin{array}{rcl}
\mathbf{BV}[\ ] & ::= & [\ ] \mid \mathbf{BV}[\ ]\,\Lambda \mid \mathsf{VWNF}\,\mathbf{BV}[\ ] \\
\mathsf{VWNF} & ::= & \mathsf{Val} \mid \mathsf{NeuW} \\
\mathsf{NeuW} & ::= & x\,\mathsf{VWNF}\,\{\mathsf{VWNF}\}^* \mid \mathsf{BlockW}\,\{\mathsf{VWNF}\}^* \\
\mathsf{BlockW} & ::= & (\lambda x.\Lambda)\,\mathsf{NeuW}
\end{array}$$

$$\mathbf{BV}[(\lambda x.B)N] \to_V \mathbf{BV}[[N/x]B] \qquad \text{with } N \in \mathsf{Val}$$

The set VWNF of β_V-weak-normal-forms (vwnfs for short) consists of the terms that do not have β_V-redexes except under abstraction. It contains values and neutrals in vwnf.

Definition 7.4 (Chest reduction)**.** The chest-reduction strategy $\to_{ch}$ is defined by the following reduction semantics:

$$\mathbf{CH}[\;] \;::=\; [\;] \mid \mathbf{BV}[\;]\Lambda \mid \mathsf{VWNF}\,\mathbf{BV}[\;] \mid \lambda x.\mathbf{CH}[\;]$$

$$\mathbf{CH}[(\lambda x.B)N] \to_{ch} \mathbf{CH}[[N/x]B] \qquad \text{with } N \in \mathsf{Val}$$

The set $\mathsf{CHNF} ::= x \mid \lambda x.\mathsf{CHNF} \mid \mathsf{NeuW}$ of chest normal forms (chnfs for short) consists of variables, abstractions with body in chnf, and neutrals in vwnf. A chest normal form has the following shape:

$$\lambda x_1 \ldots x_n.(\lambda y_p.B_p)(\,\cdots((\lambda y_1.B_1)(z\,W^0_1 \cdots W^0_{m_0})W^1_1 \cdots W^1_{m_1})\cdots\,)W^p_1 \cdots W^p_{m_p}$$

where $n \geq 0$, $p \geq 0$, $m_1 \geq 0$, $\ldots$, $m_p \geq 0$, and W^j_i are in vwnf. We say that $M\,W^j_1 \cdots W^j_{m_j}$ is an *accumulator*, where M is its leftmost operator which is either a variable or a block. The operand of the block in an accumulator could be, in turn, an accumulator, and accumulators are nested in this way, where the innermost one has a variable as its leftmost operator. We call this variable the *blocking variable*, which is variable z in the term above.

The term $\mathbf{T}_1 \equiv (\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}(x(\lambda x.\mathbf{\Omega}))$ introduced in Section 6 is an example of a chnf that has no β_V-nf.

Definition 7.5 (Ribcage reduction)**.** The ribcage-reduction strategy $\to_{rc}$ is defined by the following reduction semantics:

$$\mathbf{RC}[\;] \;::=\; [\;] \mid \mathbf{RC}[\;]\Lambda \mid \mathsf{VWNF}\,\mathbf{BV}[\;] \mid \lambda x.\mathbf{RC}[\;]$$

$$\mathbf{RC}[(\lambda x.B)N] \to_{rc} \mathbf{RC}[[N/x]B] \qquad \text{with } B \in \mathsf{CHNF} \text{ and } N \in \mathsf{Val}$$

Ribcage reduction delivers a chnf if the term has some. (A term can convert to several β_V-convertible chnfs that differ in the rib ends.) The only difference with respect to chest reduction is that ribcage reduction contracts the body of a β_V-redex to chnf before contracting the β_V-redex.

Definition 7.6 (Active components in λ_V)**.** The λ_V-active components of $M \in \Lambda$ are the maximal subterms of M that are not in chnf.

Paraphrasing [BKKS87, p.195] to the λ_V case:

> The word "active" refers to the fact that the [λ_V-active] components are embedded in a context which is "frozen", *i.e.* a [β_V-nf] when the holes are viewed as variables. (This frozen context of M is the trivial empty context if M is not a [chnf].)

A β_V-nf has no λ_V-active components. The λ_V-active components of a term are disjoint. For example, the λ_V-active components of $\lambda x.x(\lambda y.\mathbf{I}\,\mathbf{I})(\lambda z.(\lambda t.z)(x\,y)(\mathbf{I}\,\mathbf{I}))$ are the subterms $\lambda y.\mathbf{I}\,\mathbf{I}$ and $\lambda z.(\lambda t.z)(x\,y)(\mathbf{I}\,\mathbf{I})$.

Value normal order is defined in terms of chest reduction as follows. The λ_V-active components of the term are considered in left-to-right fashion and reduced by chest reduction. (The following lines paraphrase the ones for normal order written at the beginning of the section.) At the start, the input term is the only λ_V-active component if it is not a chnf. Once a chnf is reached, the λ_V-active components in it (if any) are subterms inside a 'frozen' β_V-nf context. Every time the chnf of a λ_V-active component is reached, the subsequent λ_V-active components in it (if any) are recursively considered in left-to-right fashion.

Definition 7.7 (Value normal order)**.** The value normal order strategy $\rightarrow_{vn}$ is defined by the following reduction semantics:

$$\mathbf{A}[\mathbf{CH}[(\lambda x.B)N] \rightarrow_{vn} \mathbf{A}[\mathbf{CH}[[N/x]B] \qquad \text{with } N \in \mathsf{Val}$$

where $\mathbf{CH}[\ ]$ is a chest reduction context and $\mathbf{CH}[(\lambda x.B)N]$ is the leftmost λ_V-active component of $\mathbf{A}[\mathbf{CH}[(\lambda x.B)N]]$, *i.e.* either $\mathbf{A}[\ ] \equiv [\ ]$ and $\mathbf{CH}[(\lambda x.B)N]$ is not in chnf, or $\mathbf{A}[\ ] \not\equiv [\ ]$ and $\mathbf{A}[\mathbf{CH}[(\lambda x.B)N]$ is a chnf such that every subterm at the left of $\mathbf{CH}[(\lambda x.B)N]$ is in β_V-nf.

We now adapt to pure λ_V the notion of standard reduction sequence in [Plo75, p.137].

Definition 7.8 (Standard reduction sequence in λ_V)**.** A standard reduction sequence (abbrev. SRS) is a sequence of terms defined inductively as follows:

(1) Any variable x is a SRS.
(2) If $N_2, \ldots, N_k$ is a SRS and $N_1 \rightarrow_V N_2$, then $N_1, \ldots, N_k$ is a SRS.
(3) If $N_1, \ldots, N_k$ is a SRS then $\lambda x.N_1, \ldots, \lambda x.N_k$ is a SRS.
(4) If $M_1, \ldots, M_j$ and $N_1, \ldots, N_k$ are SRS then $M_1\, N_1, \ldots, M_j\, N_1, \ldots, M_j\, N_k$ is a SRS.

Theorem 7.9. *Value normal order entails a SRS.*

Proof. The reduction contexts of value normal order (Def. 7.7) are of the shape $\mathbf{A}[\mathbf{CH}[\ ]]$ where $\mathbf{CH}[\ ]$ is a chest-reduction context (Def. 7.4) and, if R is the next β_V-redex to be contracted, then $\mathbf{CH}[R]$ is the leftmost λ_V-active component of $\mathbf{A}[\mathbf{CH}[R]]$. The reduction contexts for value normal order can be broken down further into $\mathbf{A}[\lambda x_1 \ldots x_n.\mathbf{BV}[\ ]]$, where $n \geq 0$ and $\mathbf{BV}[\ ]$ is a call-by-value reduction context (Def. 7.3). Def. 7.8(2) says that any reduction sequence entailed by the reduction contexts $\mathbf{BV}[\ ]$ of $\rightarrow_V$ is standard. Def. 7.8(3) says that these reduction sequences can be lifted to any number of surrounding lambdas, and so it ensures that chest-reduction contexts $\lambda x_1 \ldots x_n.\mathbf{BV}[\ ]$ are standard. The step of locating the leftmost λ_V-active component of $\mathbf{A}[\ ]$ is standard by Def. 7.8(3) and Def. 7.8(4). □

7.2. **Labelling for counting operands.** The second ingredient for the proof of the Partial Genericity Lemma is a tracking mechanism that counts the number of operands that have been passed to a particular term. Following [Klo80, BKC00] we define this tracking by introducing a *lambda calculus labelling* [TeR03, Def. 8.4.26] that specifies a generalised notion of descendant. Def. 7.10 defines the labelling $\mathfrak{C}$ for counting. The labels range over $\{\varepsilon\} \cup \mathbb{N}$, *i.e.* either an empty count ε or a count $c \geq 0$. When non-empty, the count of the operator in a redex is increased, assigned to the body of the redex, and then the redex is contracted (*i.e.* the operand is substituted by the free occurrences of the formal parameter in the body of the redex).

Definition 7.10 (Counting labelling)**.** Let the labels $\mathbb{L} = \{\varepsilon\} \cup \mathbb{N}$ be the union of the the empty count and the natural numbers. The counting labelling $\mathfrak{C}$ and the bisimulation $\mathbb{C}$ are defined by mutual induction as follow:

- The labelled terms $\mathfrak{C}(\Lambda)$ are labelled variables x^ℓ (with $\ell \in \mathbb{L}$), labelled abstractions $(\lambda x.B)^\ell$ (with B a labelled term), and labelled applications $(M\,N)^\ell$ (with M and N labelled terms). The following statements about bisimulation $\mathbb{C}$ hold:
  - $x \mathbb{C} x^\ell$.
  - If $B \mathbb{C} B'$ then $(\lambda x.B)\mathbb{C}(\lambda x.B')^\ell$.

– If $M \mathbb{C} M'$ and $N \mathbb{C} N'$ then $(M\,N)\mathbb{C}(M'\,N')^{\ell}$.

- Suppose $B \mathbb{C} B'$ and $N \mathbb{C} N'$ with the β-rule of the form $(\lambda x.B)N \to_\beta [N/x]B$. Let $B' \equiv C^{\ell_1}$. Consider the $\beta_{\mathfrak{c}}$-rule

$$((\lambda x.C^{\ell_1})^{\ell_2} N')^{\ell_3} \to_{\beta_{\mathfrak{c}}} \begin{cases} [N'/x](C^c) & \text{if } \ell_1 = c \\ [N'/x](C^{c+1}) & \text{if } \ell_2 = c \\ [N'/x](C^c) & \text{if } \ell_3 = c \\ [N'/x](C^\varepsilon) & \text{if } \ell_1, \ell_2, \ell_3 = \varepsilon \end{cases}$$

where the capture avoiding substitution function for labelled terms (defined below) preserves the label of the subject of the substitution:

$$\begin{array}{rcl} [T^{\ell_1}/x](x^{\ell_2}) & = & T^{\ell_1} \\ [T^{\ell_1}/x]((\lambda x.B^{\ell_2})^{\ell_3}) & = & (\lambda x.[T^{\ell_1}/x](B^{\ell_2}))^{\ell_3} \\ [T^{\ell_1}/x]((M^{\ell_2}\,N^{\ell_3})^{\ell_4}) & = & (([T^{\ell_1}/x](M^{\ell_2}))([T^{\ell_1}/x](N^{\ell_3})))^{\ell_4} \end{array}$$

If $\ell_2 = c$, rule $\beta_{\mathfrak{c}}$ increments the count of the abstraction and assigns it to the body C before performing the substitution. (Below we show that if some of the ℓ_1, ℓ_2, and ℓ_3 are non-empty, the first three alternatives of the $\beta_{\mathfrak{c}}$-rule coincide.) We set $\beta \mathbb{C} \beta_{\mathfrak{c}}$.

The definition of labelled terms is extended to contexts $\mathfrak{C}(\mathbf{C}[\ ])$ in the trivial way, observing that the hole $[\ ]$ in a labelled context does not carry any label. When no confusion arises, we will omit the epithet 'labelled' for terms and contexts.

Initially, all subterms have empty count ε except for a particular subterm.

Definition 7.11. Function $\mathfrak{c}$ takes a term M and delivers M' such that $M \mathbb{C} M'$ and where all the subterms of M' have empty count ε. For example,

$$\mathfrak{c}((\lambda x.\lambda y.x)z(\lambda x.x)) = (((\lambda x.(\lambda y.x^\varepsilon)^\varepsilon)^\varepsilon z^\varepsilon)^\varepsilon (\lambda x.x^\varepsilon)^\varepsilon)^\varepsilon$$

The labelling function $\mathfrak{c}$ is extended to contexts in the trivial way.

Typically, we would assign the non-empty count '0' to the unsolvable subterm that we wish to trace.

Definition 7.12. Function $\mathfrak{s}$ selects a subterm M in $\mathbf{C}[M]$, assigning count 0 to M and empty count everywhere else in $\mathbf{C}[M]$, including the proper subterms of M.

$$\mathfrak{s}(\mathbf{C}[\ ], M) = \mathbf{C}'[M'^0] \qquad \text{where } \mathfrak{c}(\mathbf{C}[\ ]) \equiv \mathbf{C}'[\ ] \text{ and } \mathfrak{c}(M) \equiv M'^\varepsilon$$

Notice that $M \mathbb{C}(\mathfrak{s}(\mathbf{C}[\ ], M))$. When no confusion arises, we write $\mathfrak{s}(\mathbf{C}[M])$ instead of $\mathfrak{s}(\mathbf{C}[\ ], M)$.

Labelling $\mathfrak{C}$ serves two different purposes. It tracks some unsolvable with non-empty count, and it counts the operands that have been passed to it. Consider the $\beta_{\mathfrak{c}}$-reduction step $((\lambda x.B^\varepsilon)^c N')^\varepsilon \to_{\beta_{\mathfrak{c}}} [N'/x](B^{c+1})$ with $B \not\equiv x$ and c a non-empty count. We are interested in counting the number of operands passed to operator $\lambda x.B$, and thus the second line of $\beta_{\mathfrak{c}}$ assigns the non-empty count $c+1$ to the body B in the substitution instance $[N'/x](B^{c+1})$.

Notice that the tracking implemented by $\mathfrak{C}$ differs from the conventional notion of descendant [Klo80, Bar84, KvdV99]. In the example above, the term $[N'/x](B^{c+1})$ would be a trace of $(\lambda x.B^\varepsilon)^c$ if $B \not\equiv x$. And similarly, if the label to be preserved was that of the application, as the $\beta_{\mathfrak{c}}$-reduction step $((\lambda x.B^\varepsilon)^\varepsilon N')^c \to_{\beta_{\mathfrak{c}}} [N'/x](B^c)$ illustrates, then the term $[N'/x](B^c)$ would be a trace of $((\lambda x.B^\varepsilon)^\varepsilon N')^c$ if $B \not\equiv x$. But the *traces* $[N'/x](B^{c+1})$ and $[N'/x](B^c)$ could never be *descendants* in the conventional sense of $(\lambda x.B^\varepsilon)^c$ and $((\lambda x.B^\varepsilon)^\varepsilon N')^c$ (respectively), because according to the labelled β-reduction of [Klo80, p.19] the labels of the operator and of the redex (*i.e.* the c in each of the examples above) vanish. We distinguish our more refined tracking from the conventional notion of descendant by using 'trace' and 'origin' for the former and 'descendant' and 'ancestor' for the latter. Notice that all the descendants of M in $\mathfrak{s}(\mathbf{C}[M])$ are traces (*i.e.* have non-empty count), but not all the traces of M in $\mathfrak{s}(\mathbf{C}[M])$ are descendants.

The counting labelling $\mathfrak{C}$ can be applied to λ_V by restricting rule $\beta_{\mathfrak{c}}$ above with $N' \in \mathfrak{C}(\mathsf{Val})$. We call the restricted rule $\beta_{\mathfrak{c}V}$ and set $\beta_V \mathbb{C} \beta_{\mathfrak{c}V}$. The definition of λ_V-solvable, of order of a term, and of value normal order is extended to labelled terms in the trivial way.

Our counting labelling captures accurately the number of operands that are passed to the tracked unsolvable. That is, when tracking M (an unsolvable of order n) in $\mathfrak{s}(\mathbf{C}[M])$, all the traces M^c_t in the $\beta_{\mathfrak{c}}$-reduction sequence are unsolvables of order $n-c$. In order to prove this invariant we first need to show that unsolvability and 'order of an unsolvable' are preserved by substitution. This result holds respectively for λK and λ_V, by taking the definitions of solvability and of 'order of a term' in the corresponding calculus: for λK, the usual definition of solvability and the 'order of a term' in [Lon83]; for λ_V, Def. 6.3 and order of a term in Section 6. We present the result for λ_V first, since this one is the novel result. The result for λK follows straightforwardly by adapting the proof of the former.

Lemma 7.13 (Order of a λ_V-unsolvable is preserved by substitution)**.** *Let $M \in \Lambda$ be a λ_V-unsolvable of order n. For every $N \in \mathsf{Val}$, the substitution instance $[N/x]M$ is a λ_V-unsolvable of order n.*

Proof. We distinguish two cases:

(1) M is of order ω. Then $M =_{\beta_V} \lambda y_1 \ldots y_k.B$ with k arbitrarily large. If $x = y_i$ for some $i \leq k$, then by substitutivity and by definition of the substitution function $[N/x]M =_{\beta_V} \lambda y_1 \ldots y_k.B$. If $x \neq y_i$ with $i \leq k$, then by substitutivity of $=_{\beta_V}$ and by definition of the substitution function, then $[N/x]M =_{\beta_V} \lambda y_1 \ldots y_k.[N/x]B$ and we are done.

(2) M is of order $n < \omega$. Then $M =_{\beta_V} \lambda y_1 \ldots y_n.B$ and by substitutivity of $=_{\beta_V}$. If $x = y_i$ for some $i \leq n$ then $[N/x]M \equiv \lambda y_1 \ldots y_n.B$ and the lemma holds. If $x \neq y_i$ for every $i \leq n$, since B is λ_V-unsolvable of order 0 and by the definitions of λ_V-solvability and of the substitution function, it suffices to show that $[N/x]B$ is of order 0. We proceed by *reductio ad absurdum*. Assume that $[N/x]B$ is of order $m > 0$. Then $[N/x]B =_{\beta_V} \lambda z_1 \ldots z_m.C$. If $x = z_j$ for some $j \leq m$, then by substitutivity and by definition of the substitution function $M =_{\beta_V} \lambda x_1 \ldots x_n z_1 \ldots z_m.C$, which contradicts the assumptions. If $x \neq z_j$ for every $j \leq m$, then $C \equiv [N/x]B'$ for some B', and by substitutivity and by definition of the substitution function then $M =_{\beta_V} \lambda x_1 \ldots x_n z_1 \ldots z_m.B'$, which also contradicts the assumptions and we are done. □

Lemma 7.14 (Order of a λK-unsolvable is preserved by substitution)**.** *Let $M \in \Lambda$ be a λK-unsolvable of order n. For every $N \in \Lambda$, the substitution instance $[N/x]M$ is a λK-unsolvable of order n.*

Proof. By adapting the proof of Lemma 7.13 to λK in a straightforward way. □

The invariant stated in Lemma 7.15 and 7.16 below ensure that, even if several of the ℓ_1, ℓ_2, and ℓ_3 in the $\beta_{\mathfrak{c}}$-rule above are non-empty, all the alternatives coincide and thus $\beta_{\mathfrak{c}}$-reduction is confluent.

Lemma 7.15. *Let $M_0 \in \Lambda$ λ_V-unsolvable of order n_0. Every trace of M_0 with non-empty count c in any $\beta_{\mathfrak{c}V}$-reduction sequence starting at $\mathfrak{s}(\mathbf{C}[M_0])$ is λ_V-unsolvable of order n such that $n_0 = c + n$.*[6]

Proof. By definition, only the traces of M_0 (we refer to them as M_t) have non-empty count c. We prove that the contractum of a $\beta_{\mathfrak{c}V}$-redex preserves the invariant $n_0 = c + n$ (recall that we mind the left-cancellative addition for ordinals) for each labelled trace M_t with non-empty count c and order n. We consider any $\beta_{\mathfrak{c}V}$-reduction sequence and proceed by induction on the sequence order of the term in which the $\beta_{\mathfrak{c}V}$-redex occurs. (The general case coincides with the base case, except for the small differences pinpointed in Cases 2, 3, and 4 below.) Consider the $\beta_{\mathfrak{c}V}$-redex $R \equiv (\lambda x.B)N$ with $N \in \mathfrak{C}(\mathsf{Val})$ occurring at step s that is contracted in order to produce the reduct at step $s+1$. We focus on each occurrence (if any) of M_t with non-empty count c in R and distinguish the following cases:

(1) $R \equiv (\lambda x.\mathbf{C}[M_t])N$. The contractum is $C \equiv \mathbf{C}'[[N/x]M_t]$ where $\mathbf{C}'[\ ] \equiv [N/x](\mathbf{C}[\ ])$. By Lemma 7.13, if $\mathbf{C}[\ ] \equiv [\ ]$ then the occurrence of $[N/x]M_t$ in the contractum is λ_V-unsolvable of order n and the lemma holds. (Notice that the first line of rule $\beta_{\mathfrak{c}}$ of Def. 7.10 takes care of preserving the count c of the redex's body M_t if $\mathbf{C}[\ ] \equiv [\ ]$.) Otherwise the order and count of the occurrences of M_t in $[N/x](\mathbf{C}[M_t])$ are trivially preserved and the lemma follows.

(2) $R \equiv M_t\,N$. Then $M_t \equiv (\lambda x.B)^c$ with B λ_V-unsolvable of order $n-1$. By Lemma 7.13 $[N/x]B^{c+1}$ is λ_V-unsolvable of order $n' = n - 1$ and the lemma holds. Notice that left-subtraction allows for the limit case when both n and n' are infinite ordinals (*i.e.* $\omega = n = 1 + n' = 1 + \omega = \omega$). This is enough for the base case (*i.e.* $s = 1$), but for the general case there can be an overlap with Case 1 if some trace M'_t of M_0 occurs in B^c. The lemma follows as in Case 1 except if $\mathbf{C}[\ ] \equiv [\ ]$, because the first and the second lines of rule $\beta_{\mathfrak{c}}$ of Def. 7.10 produce a critical pair. But we show that both alternatives coincide. Let M'_t with non-empty count c' be λ_V-unsolvable of order n'. By the induction hypothesis, the invariant holds for M'_t (*i.e.* $n_0 = c' + n'$). In the limit case (*i.e.* $n_0 = \omega$) both $M_t \equiv \lambda x.M'_t$ and M'_t have infinite order (*i.e.* $n = n' = \omega$) and then $n_0 = c' + n' = c + n$ and the lemma follows. In the finite case, $n - n' = 1$ and then $n_0 = c' + n' = c + n' + 1$ and $c' = c + 1$. Therefore both alternatives for rule $\beta_{\mathfrak{c}V}$ coincide and the lemma follows.

(3) $R \equiv M_t$. Then $M_t \equiv ((\lambda x.B)N)^c$ with $N \in \mathfrak{C}(\mathsf{Val})$. For the base case the lemma follows because the third line of $\beta_{\mathfrak{c}}$ of Def. 7.10 preserves the count of the $\beta_{\mathfrak{c}V}$-redex. For the general case there can be overlap with Cases 1 and 2, and the lemma follows because the different alternatives for $\beta_{\mathfrak{c}}$ coincide by the induction hypothesis, as was illustrated in Case 2 above.

(4) $R \equiv (\lambda x.B)(\mathbf{C}[M_t])$. For each occurrence of x in B, the order and count of M'_t is trivially preserved by the definition of the substitution function (Def. 7.10) and the lemma holds. This is enough for the base case. For the general case there can be an overlap with Cases 1, 2, and 3, and the lemma follows because the different alternatives for $\beta_{\mathfrak{c}}$ coincide by the induction hypothesis, as was illustrated in Case 2 above. □

[6] We assume the standard conventions on ordinal number arithmetic [Sie65]. The successor of an ordinal α is $\alpha + 1$. Addition is non-commutative and left-cancellative, that is, let n be a finite ordinal, then $n + \omega = 0 + \omega = \omega$. Only left subtraction is definable, *i.e.* $\alpha - \beta = \gamma$ iff $\beta \leq \alpha$ and γ is the unique ordinal such that $\alpha = \beta + \gamma$.

Lemma 7.16. *Let $M_0 \in \Lambda$ λK-unsolvable of order n_0. Every trace of M_0 with non-empty count c in any $\beta_{\mathfrak{c}}$-reduction sequence starting at $\mathfrak{s}(\mathbf{C}[M_0])$ is λK-unsolvable of order n such that $n_0 = c + n$.*

Proof. By adapting the proof of Lemma 7.15 to λK in a straightforward way. □

7.3. **Generalised statement and illustration of the proof.** We generalise the statement of the Partial Genericity Lemma we gave in Lemma 6.4 to provide a proof by induction on the length of the reduction sequence of value normal order.

First, we take Lemma 6.4 and pull out the universal quantifier 'for all contexts $\mathbf{C}[\ ]$' from the consequent of the implication. We take value normal order (Def. 7.7) and the counting labelling $\mathfrak{C}(\lambda_V)$ (Def. 7.10). We take M, N, and $\mathbf{C}[\ ]$ in Lemma 6.4 and subscript them with a 0 to indicate that $M_0 \in \mathfrak{C}(\Lambda)$ is the initial labelled λ_V-unsolvable, $N_0 \in \mathfrak{C}(\mathsf{VNF})$ is the labelled normal form, and $\mathbf{C}_0[\ ]$ is the initial labelled context such that $\mathfrak{s}(\mathbf{C}[M]) = \mathbf{C}_0[M_0]$, $\mathbf{C}_0[M_0] =_{\beta_{\mathfrak{c}V}} N_0$ and $N \mathbb{C} N_0$. (We also rename n to n_0 for uniformity.) The generalised theorem reads as follows.

Theorem 7.17. *Let $M' \in \mathfrak{C}(\Lambda)$ of order $n' \leq n_0$ and $\mathbf{C}'[\ ]$ a labelled context. That $\mathbf{C}'[M']$ is a labelled reduct in the value-normal-order reduction sequence of $\mathbf{C}_0[M_0]$ and M' has non-empty count implies that if $\mathbf{C}'[M'] =_{\beta_{\mathfrak{c}V}} N_0$ then for all terms X of order $m \geq n_0$ it is the case that $\mathbf{C}_0[\mathfrak{c}(X)] =_{\beta_{\mathfrak{c}V}} N_0$.*

This theorem coincides modulo $\mathbb{C}$ bisimilarity with Lemma 6.4 by taking $\mathbf{C}'[\ ] \equiv \mathbf{C}_0[\ ]$, $M' \equiv M_0$, and $n' = n_0$. In that case $\mathbf{C}'[M'] \equiv \mathbf{C}_0[M_0]$ is the first term in the reduction sequence and M' has non-empty count 0 in $\mathbf{C}'[M']$.

Proof of Thm. 7.17. For brevity, we drop the $\mathfrak{C}$ and $\mathfrak{c}$ from the sets of terms and from the reduction rule respectively. Recall from Def. 7.7 that the terms in a value-normal-order reduction sequence have the shape $\mathbf{A}[\mathbf{CH}[R]]$, where R is the next β_V-redex to be contracted, $\mathbf{CH}[\ ]$ is a chest-reduction context, and $\mathbf{A}[\ ]$ is the context in which the leftmost λ_V-active component $A \equiv \mathbf{CH}[R]$ occurs. We focus on the traces of M_0 that pop up in the value-normal-order reduction sequence of $\mathbf{C}'[M']$. We proceed by induction on the length of the value-normal-order reduction sequence of $\mathbf{C}'[M']$.

The base case is when all the traces of M_0 (*i.e.* all the subterms with non-empty count) that occur in $\mathbf{C}'[M']$ (the M' itself is one of these traces since it has non-empty count) are discarded in the next value-normal-order reduction step. That is, the hole in $\mathbf{C}'[\ ]$, and every other trace of M_0, lie inside the operand of the next β_V-redex R, *i.e.* $\mathbf{C}'[M'] \equiv \mathbf{A}[\mathbf{CH}[R]]$, and $R \equiv (\lambda x.B)(\mathbf{C}_1[M'])$ such that x does not occur free in B and $\mathbf{C}_1[M']$ is a value that contains all the traces of M_0. The next reduct is $\mathbf{A}[\mathbf{CH}[B]]$. There is no $\mathbf{C}''[\ ]$ such that $\mathbf{C}''[M'] \rightarrow^*_{\beta_V} \mathbf{A}[\mathbf{CH}[B]]$ and such that the value-normal-order reduction sequence of $\mathbf{C}''[M']$ is of length less than the value-normal-order reduction sequence of $\mathbf{C}'[M']$, which explains why this is the base case.

Since M' is a trace of M_0 and M_0 has count 0 in $\mathbf{C}_0[M_0]$, if the count of M' is greater than 0 then by Def. 7.10 this can only be the result of a reduction step $\mathbf{A}[\mathbf{CH}[(\lambda x.B^c)N]] \rightarrow_{vn} \mathbf{A}[\mathbf{CH}[[N/x](B^{c+1})]$ with B^c a trace of M_0 with non-empty count c. Had the count of the contractum $[N/x](B^{c+1})$ reached n_0, then by Lemmata 7.13 and 7.15 the contractum would be λ_V-unsolvable of order 0 and the $\mathbf{A}[\mathbf{CH}[[N/x](B^{c+1})]$ would have diverged under value normal order. But this contradicts the assumption $\mathbf{C}'[M'] =_{\beta_V} N_0$. Therefore M' has count at most $n_0 - 1$ and order greater than 0, *i.e.* $M' \in \mathsf{Val}$. If the M_0 was replaced by a term X, the trace of X would have reached at most count $n_0 - 1$. Therefore, for any $X \in \Lambda$ of order greater or equal than n_0 it is the case that $\mathbf{C}_0[X] =_{\beta_V} N_0$, and the theorem follows.

Now we proceed with the general case. We analyse the cases:

(1) No trace of M_0 pops up, *i.e.* $\mathbf{C}'[M']$ is not of the form $\mathbf{A}[\mathbf{CH}[M_t]]$ where M_t is a β_V-redex with non-empty count c. Let R be the next β_V-redex to be contracted, *i.e.* $\mathbf{A}[\mathbf{CH}[R]]$. Either $R \equiv (\lambda x.B)(\mathbf{C}_1[M'])$ with x not free in B and this case matches the conditions of the base case and we are done, or contracting R does not discard all the traces of M_0 that occur in $\mathbf{C}'[M']$. Let R' be the contractum of R and M_t be one of the traces of M_0 that occurs in $\mathbf{A}[\mathbf{CH}[R']]$, *i.e.* $\mathbf{A}[\mathbf{CH}[R']] \equiv \mathbf{C}_2[M_t]$ and M_t has non-empty count c (it is immaterial for the proof which of the the existing traces of M_0 you pick). By an argument similar to the one in the base case, the count of M_t is at most $n_0 - 1$. The theorem holds for $\mathbf{C}_2[\ ]$ and M_t by the induction hypothesis.

(2) A trace of M_0 pops up, *i.e.* $\mathbf{C}'[M']$ is of the form $\mathbf{A}[\mathbf{CH}[M_t]]$ where M_t is a β_V-redex with non-empty count c. M_t is the next redex to be contracted, and thus it is of the shape $((\lambda x.B)N)^c$ with $N \in \mathsf{Val}$. By Def. 7.10 the contractum of M_t is $[N/x](B^c)$, which has count c by Lemma 7.15. The theorem holds for $\mathbf{A}[\mathbf{CH}[\ ]]$ and $[N/x](B^c)$ by the induction hypothesis. □

The following example illustrates the proof. (Remember we are dropping the $\mathfrak{C}$ and $\mathfrak{c}$ from the sets of terms and from the reduction rule respectively.) Consider the context $\mathbf{C}_0[\ ] \equiv (\lambda x.(\lambda y.\mathbf{I})(x\,x))[\ ]$ and the λ_V-unsolvable $M_0 \equiv (\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0$ of order 2 and with non-empty count 0. The conversion $\mathbf{C}_0[M_0] =_{\beta_V} \mathbf{I}$ holds, where $\mathbf{I} \in \mathsf{VNF}$. The proof proceeds by induction on the length of the value-normal-order reduction sequence of $\mathbf{C}_0[M_0]$. We analyse this reduction sequence and check that the $\mathbf{I}$ is reached when replacing M_0 by a generic term X of order $m \geq 2$. The first β_V-redex to be contracted is $\mathbf{C}_0[M_0]$. Not all traces of M_0 are discarded in the next reduct and we are at the sub-case of the general case where no trace of M_0 pops up in the reduction sequence.

$$
\begin{array}{ll}
 & (\lambda x.(\lambda y.\mathbf{I})(x\,x))(\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0 \\
\to_{vn} & \left\{ \begin{array}{l} \mathbf{A}[\mathbf{CH}[\ ]] \equiv [[\ ]] \\ R \equiv (\lambda x.(\lambda y.\mathbf{I})(x\,x))(\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0 \end{array} \right\} \\
 & (\lambda y.\mathbf{I})((\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0(\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0)
\end{array}
$$

The remaining reduction sequence has length less than the reduction sequence of $\mathbf{C}_0[M_0]$ and the property holds for $M_1 \equiv M_0$ and $\mathbf{C}_1[\ ] \equiv (\lambda y.\mathbf{I})([\ ](\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0)$ by the induction hypothesis. (Alternatively, we could have picked the rightmost trace of M_0 and the property would also hold for $M_1 \equiv M_0$ and $\mathbf{C}_1[\ ] \equiv (\lambda y.\mathbf{I})((\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0[\ ])$.) The next β_V-redex to be contracted is the leftmost occurrence of M_0 in $\mathbf{C}_1[M_1]$. Not all the traces of M_0 are discarded in the next reduct and we are at the sub-case of the general case where a trace of M_0 pops up in the reduction sequence.

$$
\begin{array}{ll}
 & (\lambda y.\mathbf{I})((\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0(\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0) \\
\to_{vn} & \left\{ \begin{array}{l} \mathbf{A}[\mathbf{CH}[\ ]] \equiv [(\lambda y.\mathbf{I})([\ ](\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0)] \\ R \equiv (\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0 \end{array} \right\} \\
 & (\lambda y.\mathbf{I})((\lambda x.\lambda y.x\,\mathbf{\Omega})^0(\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0)
\end{array}
$$

The trace converts to $(\lambda x.\lambda y.x\,\mathbf{\Omega})^0$, which is λ_V-unsolvable of order 2. The remaining reduction sequence has length less than the reduction sequence of $\mathbf{C}_1[M_1]$ and the property holds for $M_2 \equiv (\lambda y.\lambda z.x\,\mathbf{\Omega})^0$ and $\mathbf{C}_2[\ ] \equiv (\lambda y.\mathbf{I})([\ ](\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0)$ by the induction hypothesis. The next β_V-redex to be contracted is the rightmost occurrence of M_0 in $\mathbf{C}_2[M_2]$. Again, we are at the sub-case of the general case where a trace of M_0 pops up in the reduction sequence.

$$\begin{array}{c} (\lambda y.\mathbf{I})((\lambda x.\lambda y.x\,\mathbf{\Omega})^0(\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0) \\ \to_{vn} \left\{ \begin{array}{l} \mathbf{A}[\mathbf{CH}[\ ]] \equiv [(\lambda y.\mathbf{I})((\lambda x.\lambda y.x\,\mathbf{\Omega})^0[\ ])] \\ R \equiv (\mathbf{I}(\lambda x.\lambda y.x\,\mathbf{\Omega}))^0 \end{array} \right\} \\ (\lambda y.\mathbf{I})((\lambda x.\lambda y.x\,\mathbf{\Omega})^0(\lambda x.\lambda y.x\,\mathbf{\Omega})^0) \end{array}$$

The trace converts to $(\lambda x.\lambda y.x\,\mathbf{\Omega})^0$, which is λ_V-unsolvable of order 2. The remaining reduction sequence has length less than the reduction sequence of $\mathbf{C}_2[M_2]$. The property holds for $M_3 \equiv (\lambda y.\lambda z.x\,\mathbf{\Omega})^0$ and $\mathbf{C}_3[\ ] \equiv (\lambda y.\mathbf{I})((\lambda y.\lambda z.x\,\mathbf{\Omega})^0[\ ])$ by the induction hypothesis. (Alternatively, we could have picked the leftmost occurrence of $(\lambda y.\lambda z.x\,\mathbf{\Omega})^0$ as the trace of M_0 and the property would also hold for M_3 as before and $\mathbf{C}_3[\ ] \equiv (\lambda y.\mathbf{I})([\ ](\lambda y.\lambda z.x\,\mathbf{\Omega})^0)$.) The next β_V-redex to be contracted is $(\lambda x.\lambda y.x\,\mathbf{\Omega})^0(\lambda x.\lambda y.x\,\mathbf{\Omega})^0$ (*i.e.* it is not a trace of M_0 itself, but it has the traces of M_0 both as the operator and as the operand). Not all the traces of M_0 are discarded in the next reduct and we are at the sub-case of the general case where no trace of M_0 pops up in the reduction sequence.

$$\begin{array}{c} (\lambda y.\mathbf{I})((\lambda x.\lambda y.x\,\mathbf{\Omega})^0(\lambda x.\lambda y.x\,\mathbf{\Omega})^0) \\ \to_{vn} \left\{ \begin{array}{l} \mathbf{A}[\mathbf{CH}[\ ]] \equiv [(\lambda y.\mathbf{I})[\ ]] \\ R \equiv (\lambda x.\lambda y.x\,\mathbf{\Omega})^0(\lambda x.\lambda y.x\,\mathbf{\Omega})^0 \end{array} \right\} \\ (\lambda y.\mathbf{I})(\lambda y.(\lambda x.\lambda y.x\,\mathbf{\Omega})^0\mathbf{\Omega})^1 \end{array}$$

This step increases the count of the operator to 1, which is now λ_V-unsolvable of order 1. The next redex discards all the traces of M_0, neither of which has reached count 2. We are at the base case.

$$\begin{array}{c} (\lambda y.\mathbf{I})(\lambda y.(\lambda x.\lambda y.x\,\mathbf{\Omega})^0\mathbf{\Omega})^1 \\ \to_{vn} \left\{ \begin{array}{l} \mathbf{A}[\mathbf{CH}[\ ]] \equiv [[\ ]] \\ R \equiv (\lambda y.\mathbf{I})(\lambda y.(\lambda x.\lambda y.x\,\mathbf{\Omega})^0\mathbf{\Omega})^1 \end{array} \right\} \\ \mathbf{I} \end{array}$$

Indeed, the property holds by replacing M_0 for any X of order $m \geq 2$. Consider $X \equiv (\lambda x.\lambda y.M)$ with $M \in \Lambda$. The reduction sequence becomes:

$$\begin{array}{l} (\lambda x.(\lambda y.\mathbf{I})(x\,x))(\lambda x.\lambda y.M) \to_{vn} (\lambda y.\mathbf{I})((\lambda x.\lambda y.M)(\lambda x.\lambda y.M)) \\ \to_{vn} (\lambda y.\mathbf{I})(\lambda y.[(\lambda x.\lambda y.M)/x]M)) \to_{vn} \mathbf{I} \end{array}$$

7.4. **Complete strategies of λ_V that are not standard.** Standard reduction sequences are not unique [HZ09, Sec.1.5]. To this we add that not every complete reduction sequence that only contracts needed redexes is standard! There are reduction strategies of λ_V which only contract needed redexes but do not entail standard reduction sequences. This fact is the analogous in λ_V to the result in [BKKS87] about spine strategies of λK. We shall see an example in Def. 7.5 below.

To illustrate the non-uniqueness of standard reduction sequences, consider the term $M \equiv (\lambda x.(\lambda y.z\,y)\mathbf{I})((\lambda y.z\,y)\mathbf{K})$ that converts to the stuck $(\lambda x.z\,\mathbf{I})(z\,\mathbf{K})$. The reduction sequence is standard and ends in M's β_V-nf:

$$(\lambda x.(\lambda y.z\,y)\mathbf{I})((\lambda y.z\,y)\mathbf{K}) \to_V (\lambda x.(\lambda y.z\,y)\mathbf{I})(z\,\mathbf{K}) \to_{\beta_V} (\lambda x.z\,\mathbf{I})(z\,\mathbf{K})$$

The first $\to_V$ step is a call-by-value step, which is standard by Def. 7.8(2). The second $\to_{\beta_V}$ step is standard by Def. 7.8(4), Def. 7.8(3), and Def. 7.8(2).

However, the following alternative reduction sequence is also standard and also ends in M's β_V-nf:

$$(\lambda x.(\lambda y.z\,y)\mathbf{I})((\lambda y.z\,y)\mathbf{K}) \to_{\beta_V} (\lambda x.z\,\mathbf{I})((\lambda y.z\,y)\mathbf{K}) \to_{\beta_V} (\lambda x.z\,\mathbf{I})(z\,\mathbf{K})$$

The first $\to_{\beta_V}$ step is standard by Def. 7.8(4), Def. 7.8(3), and Def. 7.8(2). The second $\to_{\beta_V}$ step is standard by Def. 7.8(4) and Def. 7.8(2).

Ribcage reduction (Def.7.5) is complete with respect to chnf and only contracts needed redexes. The definition of value normal order (Def. 7.7) can be modified to use ribcage reduction instead of chest reduction for λ_V-active components. The resulting strategy is full-reducing and complete with respect to β_V-nf, but it does not entail a standard reduction sequence. For example, consider the term $N \equiv (\lambda x.(\lambda y.x)z)\mathbf{I}$ which converts to the β_V-nf $\mathbf{I}$. Ribcage reduction entails the reduction sequence

$$(\lambda x.(\lambda y.x)z)\mathbf{I} \to_{rc} (\lambda x.x)\mathbf{I} \to_{rc} \mathbf{I}$$

This reduction sequence is not standard, although the steps, in isolation, are standard. The first is standard by Def. 7.8(4), Def. 7.8(3), and Def. 7.8(2). The second is standard by Def. 7.8(2). However, none of the rules of Def. 7.8 allow us to prepend the first step to the standard reduction sequence consisting of the second step.

Standard reduction sequences to β_V-nf fall short of capturing all complete strategies of λ_V. In [BKKS87, p.208] they generalise the Quasi-Leftmost Reduction Theorem [HS08, Thm. 3.22] and show that 'quasi-needed reduction is normalising'. An analogous result is missing for λ_V (Section 10).

7.5. **An operational characterisation of λ_V-solvability?** Although analogous to head reduction and similar in spirit, chest reduction does not provide an operational characterisation of λ_V-solvability. The term $\mathbf{T}_1 \equiv (\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}(x(\lambda x.\mathbf{\Omega}))$ introduced in Section 6 and the term $\mathbf{T}_2 \equiv (\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}(\lambda x.\mathbf{\Omega})$ are chnfs that are not λ_V-solvable. The diverging subterm $\lambda x.\mathbf{\Omega}$ cannot be discarded because $(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}$ is not transformable. Although $(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}$ is trivially freezable into a β_V-nf, there is no context $\mathbf{C}[\ ]$ that transforms that term to some term that could discard the trailing $\lambda x.\mathbf{\Omega}$ and obtain a β_V-nf.

The λ_V-solvables are 'more reduced' than chnfs. This brings us to the question of the existence of an operational characterisation of λ_V-solvables, that is, a reduction strategy of λ_V that terminates iff the input term is λ_V-solvable. We believe such strategy exists but cannot be compositional because it requires non-local information about the shape of the term to decide which is the next β_V-redex (Section 10).

8. The consistent λ_V-theory $\mathcal{V}$

We adapt [Bar84, Def. 4.1.1] and say that a *λ_V-theory* is a consistent extension of a conversion proof-theory of λ_V. In this section we prove the consistency of the λ_V-theory $\mathcal{V}$ introduced in Section 6. The proof proceeds in similar fashion to the proof of consistency of the λK-theory $\mathcal{H}$ introduced in Section 3. The latter proof is detailed in [Bar84, Sec. 16.1] and employs some technical machinery introduced in [Bar84, Sec. 15.2]. We prove the consistency of $\mathcal{V}$ in similar fashion, save for the use of a shorter proof technique in a particular lemma. We ask the reader to read this section in parallel with [Bar84, Sec. 16.1] and [Bar84, Sec. 15.2].

The reader also needs to recall the definition of 'notion of reduction' [Bar84, p.50ff] and 'substitutive' binary relation [Bar84, p.55ff]. Rule β_V is a notion of reduction from which relations $\rightarrow_{\beta_V}$, $\rightarrow^*_{\beta_V}$, and $=_{\beta_V}$ are generated (Section 2).

The structure of this section is as follows: We first define $\mathbf{\Omega}_V$-reduction that sends λ_V-unsolvables of order n to a special symbol $\mathbf{\Omega}_n$. We then consider the notion of reduction $\beta_V \cup \mathbf{\Omega}_V$ which, paraphrasing [Bar84, p.388], is interesting because it analyses provability in λ_V. We define $\beta_V\mathbf{\Omega}_V$-reduction as the compatible, reflexive, and transitive closure of $\beta_V \cup \mathbf{\Omega}_V$, and prove that it is a $\textsc{v}$-substitutive relation. At this point the storyline differs from [Bar84] in that we introduce the notion of complete $\mathbf{\Omega}_V$-development of a term, and use the Z property [vO08] to prove that $\beta_V \cup \mathbf{\Omega}_V$ is Church-Rosser ($\beta_V\mathbf{\Omega}_V$-reduction is confluent). Finally, we define $\mathcal{V}$ and the notion of ω-sensibility, and prove that $\mathcal{V}$ is generated by $\beta_V \cup \mathbf{\Omega}_V$. The consistency of $\mathcal{V}$ (Thm. 8.23) follows from the confluence of $\beta_V\mathbf{\Omega}_V$-reduction.

Definition 8.1. The $\mathbf{\Omega}_V$-reduction, $\rightarrow^*_{\mathbf{\Omega}_V}$, is the compatible, reflexive, and transitive closure of the notion of reduction

$$\mathbf{\Omega}_V = \{(M, \mathbf{\Omega}_n) \mid M \ \lambda_V\text{-unsolvable of order } n \text{ and } M \not\equiv \mathbf{\Omega}_n\}$$

where $\mathbf{\Omega}_n$ stands for the term $\lambda x_1 \ldots x_n.\mathbf{\Omega}$ (if $n \neq \omega$), or the term $\mathbf{Y}\,\mathbf{K}$ (if $n = \omega$). Notice that $\mathbf{Y}\,\mathbf{K}$ does not have a β_V-nf and that it reduces to $\lambda x_1 \ldots x_k.\mathbf{Y}\,\mathbf{K}$ with k arbitrarily large.

The $\mathbf{\Omega}_V$-conversion, $=_{\mathbf{\Omega}_V}$, is the symmetric closure of $\rightarrow^*_{\mathbf{\Omega}_V}$.

Definition 8.2. The $\beta_V\mathbf{\Omega}_V$-reduction, $\rightarrow^*_{\beta_V\mathbf{\Omega}_V}$, is the compatible, reflexive, and transitive closure of the notion of reduction $\beta_V \cup \mathbf{\Omega}_V$.

The $\beta_V\mathbf{\Omega}_V$-conversion, $=_{\beta_V\mathbf{\Omega}_V}$, is the symmetric closure of $\rightarrow^*_{\beta_V\mathbf{\Omega}_V}$.

Definition 8.3. Let $M, N \in \Lambda$ and $V \in \mathsf{Val}$. A binary relation R is $\textsc{v}$-substitutive iff $R(M, N)$ implies $R([V/x]M, [V/x]N)$.

Lemma 8.4. *If R is $\textsc{v}$-substitutive, then $\rightarrow_R$, $\rightarrow^*_R$, and $=_R$ are $\textsc{v}$-substitutive.*

Proof. Straightforward by structural induction on the derivations of $\rightarrow_R$, $\rightarrow^*_R$, and $=_R$, respectively (*i.e.* by considering the sets $\{\mu, \nu, \xi\}$, $\{\mu, \nu, \xi, \rho, \sigma\}$, or $\{\mu, \nu, \xi, \rho, \sigma, \tau\}$, respectively, from the rules in Section 2). □

Lemma 8.5. *The notion of reduction β_V is $\textsc{v}$-substitutive.*

Proof. Thm. 1 in [Plo75, p.135] states that $=_{\beta_V}$ is $\textsc{v}$-substitutive in the applied λ_V. By an argument similar to the proof of that theorem it is straightforward to prove that the β_V-rule is $\textsc{v}$-substitutive. □

Lemma 8.6. *The relations $\rightarrow_{\beta_V}$, $\rightarrow^*_{\beta_V}$, and $=_{\beta_V}$ are $\textsc{v}$-substitutive.*

Proof. Trivial by Lemma 8.4 above. □

Lemma 8.7. *Let R_1 and R_2 be two notions of reduction that are $\textsc{v}$-substitutive. The union $R_1 \cup R_2$ is $\textsc{v}$-substitutive.*

Proof. Trivial, by considering R_1 or R_2 individually. □

Lemma 8.8 ($\beta_V\mathbf{\Omega}_V$ is V-substitutive)**.** *Let $M, N \in \Lambda$ and $V \in \mathsf{Val}$. $M \to_{\beta_V\mathbf{\Omega}_V} N$ implies $[V/x]M \to_{\beta_V\mathbf{\Omega}_V} [V/x]N)$.*

Proof. By Lemma 8.7, it is enough to prove that $\mathbf{\Omega}_V$ is V-substitutive. Let $M \to_{\mathbf{\Omega}_V} \mathbf{\Omega}_n$. By Lemma 7.13, the substitution instance $[V/x]M$ is λ_V-unsolvable of order n for any $V \in \mathsf{Val}$. By Def. 8.1 above, $[V/x]M \to_{\mathbf{\Omega}_V} \mathbf{\Omega}_n$ and $\mathbf{\Omega}_n \equiv [V/x]\mathbf{\Omega}_n$ because all the $\mathbf{\Omega}_n$ (including $\mathbf{\Omega}_\omega$) are closed terms. □

Lemma 8.9. *The relations $\to^*_{\beta_V\mathbf{\Omega}_V}$, and $=_{\beta_V\mathbf{\Omega}_V}$ are V-substitutive.*

Proof. Trivial by Lemma 8.4 above. □

Definition 8.10. Let $M, N \in \Lambda$. M and N are λ_V-solvably equivalent, $M \sim_{s_V} N$, iff for every arbitrary context $\mathbf{C}[\ ]$, $\mathbf{C}[M]$ is λ_V-unsolvable of order n iff $\mathbf{C}[N]$ is λ_V-unsolvable of order n.

Relation $\sim_{s_V}$ is reflexive, symmetric, and transitive, and hence it is an equivalence relation.

Lemma 8.11. *Let $M, N \in \Lambda$.*
(1) *$M =_{\beta_V} N$ implies $M \sim_{s_V} N$.*
(2) *$M =_{\mathbf{\Omega}_V} N$ implies $M \sim_{s_V} N$.*

Proof. First consider (1). Since $=_{\beta_V}$ is compatible, for any context $\mathbf{C}[\ ]$ then $\mathbf{C}[M] =_{\beta_V} \mathbf{C}[N]$, and (1) trivially follows.

Now consider (2). Since $\sim_{s_V}$ is an equivalence relation, it is enough to show that $M \sim_{s_V} \mathbf{\Omega}_n$ for M λ_V-unsolvable of order n. Suppose $\mathbf{C}[M]$ is λ_V-solvable. Then there exists a function context $\mathbf{F}[\ ]$ such that $\mathbf{F}[\mathbf{C}[M]] =_{\beta_V} N$ for some $N \in \mathsf{VNF}$. By the Partial Genericity Lemma (Lemma 6.4) then $\mathbf{F}[\mathbf{C}[\mathbf{\Omega}_n]] =_{\beta_V} N$. Similarly, $\mathbf{C}[\mathbf{\Omega}_n]$ being λ_V-solvable implies $\mathbf{C}[M]$ is λ_V-solvable, and (2) follows. □

Remark 8.12. We write $\mathbf{\Omega}_V(M)$ for the $\mathbf{\Omega}_V$-normal-form (abbrev. $\mathbf{\Omega}_V$-nf) of the term M.

Lemma 8.13. *Every term has a unique $\mathbf{\Omega}_V$-nf.*

Proof. The maximal $\mathbf{\Omega}_V$-redexes are mutually disjoint. By replacing them by the appropriate $\mathbf{\Omega}_n$s, no new $\mathbf{\Omega}_V$-redexes are created, since $U_n \sim_{s_V} \mathbf{\Omega}_n$ for U_n λ_V-unsolvable of order n. The $\mathbf{\Omega}_V$-nf is unique since $\mathbf{\Omega}_V$-reduction is Church-Rosser. □

The complete $\mathbf{\Omega}_V$-development of a term defined below adapts the notion of complete development of a term [TeR03, Sec.4.5,p.106] to $\beta_V\mathbf{\Omega}_V$-reduction.

Definition 8.14. The complete $\mathbf{\Omega}_V$-development $M^{\circ\mathbf{\Omega}}$ of a term M consists of the complete development of the $\mathbf{\Omega}_V$-nf of M, *i.e.* $M^{\circ\mathbf{\Omega}} = (\mathbf{\Omega}_V(M))^\circ$

Lemma 8.15 (Confluence of $\to^*_{\mathbf{\Omega}_V}$)**.** *The relation $\to^*_{\mathbf{\Omega}_V}$ is Church-Rosser.*

Proof. It is enough to prove that $(\to_{\mathbf{\Omega}_V} \cup \equiv)$ has the diamond property. Consider $M \to_{\mathbf{\Omega}_V} M_1$ by contracting the $\mathbf{\Omega}_V$-redex U_1, and $M \to_{\mathbf{\Omega}_V} M_2$ by contracting the $\mathbf{\Omega}_V$-redex U_2. We analyse the cases:
(1) U_1 and U_2 are disjoint. The lemma trivially holds.
(2) U_1 and U_2 overlap. Let U_1, a λ_V-unsolvable of order m, be a superterm of U_2, a λ_V-unsolvable of order n. The diagram

$$\begin{tikzcd}
M \equiv \mathbf{C}_1[U_1] \equiv \mathbf{C}_1[\mathbf{C}_2[U_2]] \arrow[r, "U_1"] \arrow[d, "U_2"] & M_2 \equiv \mathbf{C}_1[\mathbf{\Omega}_m] \arrow[d, equals] \\
M_1 \equiv \mathbf{C}_1[\mathbf{C}_2[\mathbf{\Omega}_n]] \arrow[r, "\mathbf{C}_2[\mathbf{\Omega}_n]"] & \mathbf{C}_1[\mathbf{\Omega}_m]
\end{tikzcd}$$

commutes because $\mathbf{C}_2[\mathbf{\Omega}_n] \sim_{s_V} U_1$ holds by Lemma 8.11 above. □

Lemma 8.16 (Confluence of $\beta_V\mathbf{\Omega}_V$)**.** *$\beta_V\mathbf{\Omega}_V$-reduction is Church-Rosser.*

Proof. It is enough to prove that $\rightarrow_{\beta_V\mathbf{\Omega}_V}$ has the Z property [vO08]:

$$\begin{tikzcd}
M \arrow[r, "\beta_V\mathbf{\Omega}_V"'] & N \arrow[dl, dashed, "*"', "\beta_V\mathbf{\Omega}_V"] \\
M^{\circ\mathbf{\Omega}} \arrow[r, dashed, "*", "\beta_V\mathbf{\Omega}_V"'] & N^{\circ\mathbf{\Omega}}
\end{tikzcd}$$

There are two cases, $M \rightarrow_{\mathbf{\Omega}_V} N$ and $M \rightarrow_{\beta_V} N$:

(1) Case $M \rightarrow_{\mathbf{\Omega}_V} N$. It follows that $\mathbf{\Omega}_V(M) \equiv \mathbf{\Omega}_V(N)$ and $M^{\circ\mathbf{\Omega}} \equiv N^{\circ\mathbf{\Omega}}$. Therefore $N \rightarrow^*_{\beta_V\mathbf{\Omega}_V} M^{\circ\mathbf{\Omega}}$ and $M^{\circ\mathbf{\Omega}} \rightarrow^*_{\beta_V\mathbf{\Omega}_V} N^{\circ\mathbf{\Omega}}$ and so the lemma follows.

(2) Case $M \rightarrow_{\beta_V} N$. Let R be the β_V-redex contracted in $M \rightarrow_{\beta_V} N$. Let S be the set of maximal $\mathbf{\Omega}_V$-redexes in M. If R is disjoint with S then $M^{\circ\mathbf{\Omega}} \equiv N^{\circ\mathbf{\Omega}}$ and the lemma follows as in the previous case. If R is not disjoint with some $U \in \mathsf{S}$ then we consider the sub-cases:

   (a) Sub-case $U \equiv \mathbf{C}[R]$ is λ_V-unsolvable of order n. Let R' be the contractum of R. By Lemma 8.11 above we have $\mathbf{\Omega}_V(\mathbf{C}[R]) \equiv \mathbf{\Omega}_V(\mathbf{C}[R'])$ and $\mathbf{\Omega}_V(M) \equiv \mathbf{\Omega}_V(N)$. Therefore $M^{\circ\mathbf{\Omega}} \equiv N^{\circ\mathbf{\Omega}}$.

   (b) Sub-case $R \equiv (\lambda x.B)\mathbf{C}[U]$ is λ_V-solvable with B disjoint with S. Let n be the order of U. The following diagram

$$\begin{tikzcd}
M \arrow[rr, "\beta_V"'] \arrow[d, equals] & & N \arrow[d, equals] \\
\mathbf{C}'[(\lambda x.B)\mathbf{C}[U]] \arrow[d, "\mathbf{\Omega}_V"'] & & \mathbf{C}'[[\mathbf{C}[U]/x]B\,] \arrow[ddll, "\mathbf{\Omega}_V"] \arrow[ddd, "\beta_V\mathbf{\Omega}_V"', "*"] \\
\mathbf{C}'[(\lambda x.B)\mathbf{C}[\mathbf{\Omega}_n]] \arrow[d, "\beta_V"'] & & \\
\mathbf{C}'[[\mathbf{C}[\mathbf{\Omega}_n]/x]B] \arrow[d, "\beta_V\mathbf{\Omega}_V"', "*"] & & \\
M^{\circ\mathbf{\Omega}} \arrow[rr, equals] & & N^{\circ\mathbf{\Omega}}
\end{tikzcd}$$

   commutes because $\mathbf{C}'[[\mathbf{C}[\mathbf{\Omega}_n]/x]B] \rightarrow^*_{\beta_V\mathbf{\Omega}_V} M^{\circ\mathbf{\Omega}} \equiv N^{\circ\mathbf{\Omega}}$, since B and S are disjoint.

   (c) Sub-case $R \equiv (\lambda x.\mathbf{C}[U])V$ is λ_V-solvable with $V \in \mathsf{Val}$ not necessarily disjoint with S. Let n be the order of U. The following diagram

$$\begin{tikzcd}
M \arrow[rr, "\beta_V"'] \arrow[d, equal] & & N \arrow[d, equal] \\
\mathbf{C}'[(\lambda x.\mathbf{C}[U])V] \arrow[d, "\mathbf{\Omega}_V"'] & & \mathbf{C}'[[V/x](\mathbf{C}[U])] \arrow[d, "\mathbf{\Omega}_V"'] \\
\mathbf{C}'[(\lambda x.\mathbf{C}[\mathbf{\Omega}_n])V] \arrow[d, "\mathbf{\Omega}_V"'] & & \mathbf{C}'[[V/x](\mathbf{C}[\mathbf{\Omega}_n])] \arrow[ddll, "*\ \mathbf{\Omega}_V"] \arrow[ddd, "\beta_V\mathbf{\Omega}_V"', "*"] \\
\mathbf{C}'[(\lambda x.\mathbf{C}[\mathbf{\Omega}_n])\mathbf{\Omega}_V(V)] \arrow[d, "\beta_V"'] & & \\
\mathbf{C}'[[\mathbf{\Omega}_V(V)/x](\mathbf{C}[\mathbf{\Omega}_n])] \arrow[d, "\beta_V\mathbf{\Omega}_V"', "*"] & & \\
M^{\circ\mathbf{\Omega}} \arrow[rr, equal] & & N^{\circ\mathbf{\Omega}}
\end{tikzcd}$$

commutes because

$$\mathbf{C}'[[V/x](\mathbf{C}[\mathbf{\Omega}_n])] \rightarrow^*_{\mathbf{\Omega}_V} \mathbf{C}'[[\mathbf{\Omega}_V(V)/x](\mathbf{C}[\mathbf{\Omega}_n])]$$

follows because of (i) Prop. 2.1.17(ii) in [Bar84], (ii) $\mathbf{C}[\mathbf{\Omega}_n]$ and $\mathsf{S} \setminus \{U\}$ are disjoint, and (iii) $\mathbf{\Omega}_V$ is V-substitutive. □

Definition 8.17. We say that any theory containing $\mathcal{V}$ is ω-sensible (and by extension, any model satisfying $\mathcal{V}$ is ω-sensible).

Definition 8.18 (Consistent theory)**.** Let $\mathcal{T}$ be a set of equations between terms. $\mathcal{T}$ is consistent, $\mathrm{Con}(\mathcal{T})$, iff $\mathcal{T}$ does not prove every closed equation, *i.e.*

$$\mathcal{T} \nvdash M = N \text{ with } M, N \in \Lambda^0$$

Definition 8.19 (λ_V-theory)**.** Let $\mathcal{T}$ be a set of closed equations between terms. $\mathcal{T}$ is a λ_V-theory iff $\mathrm{Con}(\mathcal{T})$ and

$$\mathcal{T} = \{M = N \mid M, N \in \Lambda^0 \text{ and } \lambda_V + \mathcal{T} \vdash M = N\}$$

Proposition 8.20. *The theory of β_V-convertible closed terms, λ_V, is a λ_V-theory. Observe that λ_V is consistent by confluence of β_V-reduction.*

Definition 8.21 (Theory $\mathcal{V}$)**.** Let $\mathcal{V}_0$ be the following set of equations:

$$\mathcal{V}_0 = \{M = N \mid M, N \in \Lambda^0 \ \lambda_V\text{-unsolvable of the same order } n\}$$

The theory $\mathcal{V}$ is the set of equations:

$$\mathcal{V} = \{M = N \mid M, N \in \Lambda^0 \text{ and } \lambda_V + \mathcal{V}_0 \vdash M = N\}$$

Lemma 8.22. *$\beta_V\mathbf{\Omega}_V$-reduction generates $\mathcal{V}$, i.e.*

$$\mathcal{V} \vdash M = N \textit{ iff } M =_{\beta_V\mathbf{\Omega}_V} N \textit{ with } M, N \in \Lambda^0$$

Proof. We first consider the direction ($\Longrightarrow$). If $\mathcal{V}_0 \vdash M = N$ then $M \rightarrow_{\mathbf{\Omega}_V} \mathbf{\Omega}_n$ and $M \rightarrow_{\mathbf{\Omega}_V} \mathbf{\Omega}_n$ because both M and N are λ_V-unsolvable of order n. Consequently, for all axioms

$M_0 = N_0$ in the set $\mathcal{V}_0$ that generates $\mathcal{V}$, $M_0 =_{\beta_V \mathbf{\Omega}_V} N_0$ holds, and then $M =_{\beta_V \mathbf{\Omega}_V} N$ follows by compatibility, reflexivity, symmetry and transitivity.

Now for the direction ($\Longleftarrow$). The theory $\mathcal{V}$ is generated by $\lambda_V + \mathcal{V}_0$, and then each β_V- or $\mathbf{\Omega}_V$-reduction step is provable in $\mathcal{V}$. □

Theorem 8.23. *$\mathcal{V}$ is a λ_V-theory.*

Proof. By Def. 8.21 and because Con($\mathcal{V}$) by Lemmata 8.22 and 8.16. □

9. Related work

We have commented at length from the introduction onwards on the relevant related work on solvability in λK and λ_V. We only comment here briefly on several outstanding points and on other work of related interest.

As mentioned in Section 7.1, value normal order is not the same strategy as the complete reduction strategy of λ_V named $\rightarrow^p_\Gamma$ that is obtained as an instantiation of the 'principal reduction machine' of [RP04, p.70]. The principal reduction machine is actually a template of small-step reduction strategies that is parametric on a set of permissible operands and a set of irreducible terms. The complete reduction strategy $\rightarrow^p_\Gamma$ is obtained by instantiating the template with the set of permissible operands fixed to Val and the set of irreducible terms fixed to VNF (in [RP04] Val is called Γ and VNF is called Γ-NF). Value normal order differs from $\rightarrow^p_\Gamma$ when reducing a term $(\lambda x.B)N$ where N converts to a neutral. In $\rightarrow^p_\Gamma$ the operand N is reduced to the neutral N' using call-by-value so that $(\lambda x.B)N'$ is a block. At this point $\rightarrow^p_\Gamma$ keeps reducing N' fully to β_V-nf before reducing B fully to β_V-nf. In contrast, value normal order proceeds in left-to-right fashion with the block $(\lambda x.B)N'$, first reducing B fully to β_V-nf and then reducing N' fully to β_V-nf. The left-to-right order is the regular one, at least so in all the strategies cited in this paper. And we have defined value normal order as the λ_V analogue of λK's normal order following the results in [BKKS87]. At any rate, reducing blocks left-to-right or right-to-left does not affect completeness. Both $\rightarrow^p_\Gamma$ and value normal order entail standard reduction sequences (Def. 7.8) and are therefore complete (this is shown for $\rightarrow^p_\Gamma$ in [RP04, p.11]).

The $\lambda^*_{\beta_V}$ calculus of [EHR91, EHR92, Def. 11] is a calculus with partial terms. There is a unique constant Ω that represents 'bottom'. The calculus has reduction rules $M\,\Omega \rightarrow \Omega$ and $\Omega\,M \rightarrow \Omega$ which capture preservation of unsolvability by application (Section 3.3). In [Wad76, p.508] we find conversion rules $\Omega\,M = \Omega$ and $\lambda x.\Omega = \Omega$ now in the context of λK. In both approaches Ω is uniquely used as 'bottom'. However, we have considered infinite bottoms with different orders, and have followed in Section 8 the syntactic approach of [Bar84] where $\mathbf{\Omega}$ is a term (not a constant representing 'bottom') and $M \rightarrow \mathbf{\Omega}$ when M unsolvable. The $\mathbf{\Omega}_n$ of Section 8 are terms.

The computational lambda calculus of [Mog91] adds the equations $\mathbf{I}\,X = X$ and $(\lambda x.y\,x)X = y\,X$, for all $X \in \Lambda$, as axioms to the proof-theory. These equations affect sequentiality (Section 4.1).

The occurrence of a free variable can be seen as the result of implicitly applying the 'opening' operation to a locally-nameless representation of a program (a closed term) [Cha12]. In the local scope operational equivalence is refined by considering open and non-closing contexts (Section 3.2) that disclose the differences in sequentiality. After that, the program can be recovered by the 'closing' operation.

The Genericity Lemma (Section 3.3) conforms with the axiomatic framework for meaningless terms of [KvdV99]. The axioms for λK state that meaningless terms are closed under reduction and substitution (Axioms 1 and 3) and that if M is meaningless then $M\,N$ is meaningless, *i.e.* M cannot be used as a function (Axiom 2). For λK, Axioms 1, 2, and 3 are enough to prove the Genericity Lemma and the consistency of the proof-theory extended with equations between all meaningless terms.

However, in λ_V there is partiality in meaninglessness, *i.e.* not all meaningless terms are bottom. The analogues of the axioms have to be order-aware. In particular, Lemma 7.13 is the order-aware analogue of Axiom 3. The analogue of Axiom 1 is trivial, just consider $=_{\beta_V}$. As for Axiom 2, if M λ_V-unsolvable of order n, then $M\,N$ (with $N \in \mathsf{Val}$) is λ_V-unsolvable of order $n-1$. However, if $N \notin \mathsf{Val}$, then $M\,N$ is λ_V-unsolvable of order 0. We leave the proof of the analogy as future work.

10. Conclusions and future work

The presupposition of v-solvability (Section 5) is that terms with β_V-nf that are not transformable to a value of choice (such as $\mathbf{B}$ and $\mathbf{U}$) are observationally equivalent to terms without β_V-nf that are also not transformable to a value of choice (such as $\mathbf{\Omega}$ and $\lambda x.\mathbf{\Omega}$), and that all of them are operationally irrelevant and meaningless. This gives rise to an inconsistent λ_V-theory. We have shown that these terms can be separated operationally and that this conforms to λ_V's nature. Neutral terms differ at the point of potential divergence, *i.e.* at the blocking variable which has to be given the opportunity to be substituted by an arbitrary value according to λ_V's principle of 'preserving confluence by preserving potential divergence' (Section 4). The actual choice of values for blocking variables lets us separate terms with the same functional result that nonetheless have different sequentiality, or may have different sequentiality when using a different complete reduction strategy. The functional models of λ_V do not have such separating capabilities, but functional models are not the only possible models. We have to follow the other line of investigation, namely, to 'vary the model to fit the intended calculus'. Models that capture sequentiality exist, and we believe there are ω-sensible models that resemble the sequential algorithms of [BC82] (Section 6).

As discussed in Section 7.4, standard reduction sequences fall short of capturing all complete strategies of λ_V. A result analogous to λK's 'quasi-needed reduction is normalising' [BKKS87, p.208] is missing for λ_V. We are currently developing the analogue for λ_V of quasi-needed reduction, and the proof that it is normalising.

As discussed in Section 7.5, we believe it is possible to give an operational characterisation of λ_V-solvability, *i.e.* a reduction strategy of λ_V that terminates iff the input term is λ_V-solvable. But we believe it cannot be compositional because it requires non-local information about the shape of the term to decide which is the next β_V-redex. We have a preliminary implementation that uses a mark-test-and-contract algorithm. Terms with positive polarity are tested for transformability and terms with negative polarity are tested for valuability. In order to test we keep a sort of stratified environment that references the operands in the nested accumulators of a chnf. The environment grows as reduction proceeds inside the body of nested blocks, where a table of lexical offsets defines what is visible at each layer. The β_V-redexes are marked for contraction, but are only contracted after testing the λ_V-solvability of the subterm in which they occur.

Our implementation can be refined using the 'linear blocking structure' of the sequent term calculus [Her95, CH00, San07]. The blocking structure of chnfs (*i.e.* the structure of nested blocks around the blocking variable) becomes a linear structure when injecting the chnfs into their sequent-term representation. The sequent-term representation seems promising to develop the analogue of Böhm trees in λ_V. Let us illustrate this by adopting the untyped lambda-Gentzen calculus of [San07] (λ^{Gtz} for short). Assume the injection $\widehat{\ }: \mathsf{CHNF} \to \Lambda^{\mathsf{Gtz}}$ and consider the shape of a chnf from Section 7.1:

$$\lambda x_1 \ldots x_n.(\lambda y_p.B_p)(\ \ldots((\lambda y_1.B_1)((z\,W_0^0)W_1^0\cdots W_{m_0}^0)W_1^1\cdots W_{m_1}^1)\ldots\ )W_1^p\cdots W_{m_p}^p$$

This shape is injected into the sequent term:

$$\lambda x_1 \ldots x_n.(z[\widehat{W_0^0}])[\widehat{W_1^0},\ldots,\widehat{W_{m_0}^0}]@(y_1)(\widehat{B_1}[\widehat{W_1^1},\ldots,\widehat{W_{m_1}^1}] \\ @(y_2)(\ldots(y_p)(\widehat{B_p}[\widehat{W_1^p},\ldots,\widehat{W_{m_p}^p}])\ldots))$$

The λ^{Gtz} representation reflects the blocking structure of the nested blocks and accumulators in linear fashion, where the blocking variable z appears in the leftmost position, and each accumulator in the trailing context 'unblocks' the subsequent accumulator.

Acknowledgement

A preliminary version of this work was presented at the 11th International Workshop on Domain Theory and Applications (Domains XI), Paris, 10th September 2014. We are grateful to Thomas Ehrhard and Alberto Carraro for their insightful comments during the workshop. We also thank Beniamino Accattoli and Flavien Breuvart for the stimulating discussions in the early stages of this work. Our gratitude to Luca Aceto at the Icelandic Centre of Excellence in Theoretical Computer Science, and to Manuel Carro and Manuel Hermenegildo at the IMDEA Software Institute of Madrid, for providing an excellent supporting environment. We are also indebted to Luca for his thoughtful review and encouragement.

References

[ABF+05] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics*, number 3603 in Lecture Notes in Computer Science, pages 50–65, 2005.

[Abr90] Samson Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116. Addison-Welsey, 1990.

[ACP+08] Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 3–15, 2008.

[AK10] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In *Proceedings of the 24th International Workshop on Computer Science Logic*, pages 381–395, 2010.

[AK12] Beniamino Accattoli and Delia Kesner. The permutative λ-calculus. In *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 7180 of *Lecture Notes in Computer Science*, pages 23–36, 2012.

[AP12] Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In *Proceedings of the 11th International Symposium on Functional and Logic Programming*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16, 2012.

[Bar71] Henk P. Barendregt. *Some Extensional Term Models for Combinatory Logics and Lambda Calculi*. PhD thesis, University of Utrecht, NL, 1971.

[Bar72] Henk P. Barendregt. Solvability in lambda caculi. In *Proceedings of the Orléans Congrès de Logique*, 1972.

[Bar84] Henk P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North Holland, 1984.

[Bar90] Henk P. Barendregt. Functional programming and lambda calculus. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 7, pages 321–364. Elsevier / MIT Press, 1990.

[BC82] Gerard Berry and Pierre-Louis Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20(3):265–321, 1982.

[BKC00] Inge Bethke, Jan W. Klop, and Roel C. de Vrijer. Descendants and origins in term rewriting. *Information and Computation*, 159(1-2):59–124, 2000.

[BKKS87] Henk P. Barendregt, J. Richard Kennaway, Jan W. Klop, and M. Ronan Sleep. Needed reduction and spine strategies for the lambda calculus. *Information and Computation*, 75:191–231, 1987.

[CDV81] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory Logic*, volume 1. North-Holland, 1958.

[CG14] Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118, 2014.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the ACM Sigplan International Conference on Functional Programming*, volume 35(9) of *ACM Sigplan Notices*, pages 233–243, 2000.

[Cha12] Arthur Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, 49(3):363–408, 2012.

[Cré90] Pierre Crégut. An abstract machine for lambda-terms normalization. In *Proceedings of the ACM conference on LISP and Functional Programming*, pages 333–340, 1990.

[Cur07] Pierre-Louis Curien. Definability and full abstraction. *Electronic Notes on Theoretical Computer Science*, 172:301–310, 2007.

[EHR91] Lavinia Egidi, Furio Honsell, and Simona Ronchi della Rocca. The lazy call-by-value λ-calculus. In *Proceedings of Mathematical Foundations of Computer Science*, pages 161–169, 1991.

[EHR92] Lavinia Egidi, Furio Honsell, and Simona Ronchi Della Rocca. Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticae*, 16(1):149–169, 1992.

[Fel87] Mathias Felleisen. *The Calculi of Lambda-v-CS Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. PhD thesis, Department of Computer Science, Indiana University, 1987.

[FF86] Mathias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the lambda-calculus. In *Formal Description of Programming Concepts III*, pages 193–217, 1986.

[GL02] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *Proceedings of the 7th International Conference on Functional Programming*, volume 37(9), pages 235–246, 2002.

[GPR15] Giulio Guerreri, Luca Paolini, and Simona Ronchi Della Rocca. Standardization of a call-by-value lambda calculus. In *Proceedings of the 13th International Conference on Typed Lambda Calculus and Applications*, pages 211–225, 2015.

[Gue13] Giulio Guerreri. *Differential nets, experiments and reduction*. PhD thesis, Dipartamento di Filosofia, Università degli Studi Roma Tre and U.F.R. d'Informatique - Laboratoire PPS, Université Paris Diderot - Paris 7, 2013.

[Her95] Hugo Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. *Lecture Notes in Computer Science*, 933:61–75, 1995.

[HS08] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators, An Introduction.* Cambridge University Press, 2008.

[HZ09] Hugo Herbelin and Stèphane Zimmermann. An operational account of call-by-value minimal and classical λ-calculus in "natural deduction" form. In *Proceedings of the 9th International Conference on Typed Lambda Calculi and Applications*, volume 5608, pages 142–156, 2009. (We have used the 2010 revision on Herbelin's website).

[Klo80] Jan W. Klop. *Combinatory Reduction Systems.* PhD thesis, University of Utrecht, 1980. Published as Mathematical Center Tract 129.

[KvdV99] J. Richard Kennaway, Vincent van Oostrom, and Fer-Jan de Vries. Meaningless terms in rewriting. *Journal of Functional and Logic Programming*, 1999(1):1–35, 1999.

[Lév75] Jean-Jacques Lévy. An algebraic interpretation of the $\lambda\beta K$-calculus and a labeled λ-calculus. In *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, March 25-27, 1975*, volume 37 of *Lecture Notes in Computer Science*, pages 147–165, 1975.

[Lon83] Giuseppe Longo. Set-theoretical models of lambda calculus: Theories, expansions and isomophisms. *Annals of Pure and Applied Logic*, 24:153–188, 1983.

[Mil90] Robin Milner. Functions as processes. Research Report RR-1154, INRIA, 1990.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[Plo75] Gordon Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

[PR99] Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *Theoretical Informatics and Applications*, 33(6):507–534, 1999.

[RP04] Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus.* Springer, 2004.

[San07] José Espírito Santo. Completing Herbelin's programme. In *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications*, volume 4583 of *Lecture Notes in Computer Science*, pages 118–132, 2007.

[Ses02] Peter Sestoft. Demonstrating lambda calculus reduction. In *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, volume 2566 of *Lecture Notes in Computer Science*, pages 420–435. Springer, 2002.

[Sie65] Waclaw Sierpinski. *Cardinal and Ordinal Numbers.* PWN-Polish Scientific Publishers, 1965.

[TeR03] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2003.

[vO08] Vincent van Oostrom. Z. Slides available on `http://www.phil.uu.nl/~oostrom/publication/talk/lix010208.pdf`., 2008.

[Wad76] Christopher Wadsworth. The relation between computational and denotational properties for Scott's D_∞-models of the lambda calculus. *Siam Journal of Computing*, 5(3):488–521, 1976.

Appendix A. Full glossary of terms and sets of terms

Set			Description
Λ	$::=$	$x \mid \lambda x.\Lambda \mid \Lambda\,\Lambda$	lambda terms
Val	$::=$	$x \mid \lambda x.\Lambda$	values
Neu	$::=$	$x\,\Lambda\,\{\Lambda\}^*$	λK neutrals
NF	$::=$	$\lambda x.\mathsf{NF} \mid x\,\{\mathsf{NF}\}^*$	β-nfs
HNF	$::=$	$\lambda x.\mathsf{HNF} \mid x\,\{\Lambda\}^*$	hnfs
NeuV	$::=$	$\mathsf{Neu} \mid \mathsf{Block}\,\{\Lambda\}^*$	λ_V neutrals
Block	$::=$	$(\lambda x.\Lambda)\mathsf{NeuV}$	blocks
VNF	$::=$	$x \mid \lambda x.\mathsf{VNF} \mid \mathsf{Stuck}$	β_V-nfs
Stuck	$::=$	$x\,\mathsf{VNF}\,\{\mathsf{VNF}\}^*$	stucks
	$\mid$	$\mathsf{BlockNF}\,\{\mathsf{VNF}\}^*$	
$\mathsf{BlockNF}$	$::=$	$(\lambda x.\mathsf{VNF})\,\mathsf{Stuck}$	blocks in β_V-nf
CHNF	$::=$	$x \mid \lambda x.\mathsf{CHNF} \mid \mathsf{NeuW}$	chnfs
VWNF	$::=$	$\mathsf{Val} \mid \mathsf{NeuW}$	vwnfs
NeuW	$::=$	$x\,\mathsf{VWNF}\,\{\mathsf{VWNF}\}^*$	neutral vwnfs
	$\mid$	$(\lambda x.\Lambda)\,\mathsf{NeuW}\,\{\mathsf{VWNF}\}^*$	

Abbreviation	Term	has β-nf	has β_V-nf
$\mathbf{I}$	$\lambda x.x$	yes	yes
$\mathbf{K}$	$\lambda x.\lambda y.x$	yes	yes
$\mathbf{K}^m$	$\lambda x.\mathbf{K}_1(\cdots(\mathbf{K}_m\,x)\cdots)$	yes	yes
$\mathbf{\Delta}$	$\lambda x.xx$	yes	yes
$\mathbf{\Omega}$	$\mathbf{\Delta}\mathbf{\Delta}$	no	no
$\mathbf{U}$	$\lambda x.\mathbf{B}$	no	yes
$\mathbf{B}$	$(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}$	no	yes
$\mathbf{T}_1$	$(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}(x(\lambda x.\mathbf{\Omega})$	no	no
$\mathbf{T}_2$	$(\lambda y.\mathbf{\Delta})(x\,\mathbf{I})\mathbf{\Delta}(\lambda x.\mathbf{\Omega})$	no	no
$\mathbf{\Omega}_n$	$\lambda x_1 \ldots x_n.\mathbf{\Delta}\mathbf{\Delta}$	no	no
$\mathbf{Y}$	$\lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x)))$	no	no
$\mathbf{\Omega}_\omega$	$\mathbf{Y}\,\mathbf{K}$	no	no

Appendix B. Proof of Thm. 3.9 on page 10 and example

Proof. From SolH we prove SolF immediately because function contexts subsume head contexts and therefore SolF subsumes SolH.

From SolF with the function context $\mathbf{F}[\ ]$ we prove SolH by closing the function context to produce a head context $\mathbf{H}[\ ]$.

$\mathbf{F}[\ ]$ is of the form $(\lambda x_1 \ldots x_n.[\ ])N_1 \cdots N_k$, with $n \geq 0$, $k \geq 0$, and $N_i \in \Lambda$. Let $\{y_1, \ldots, y_m\}$ (with $m \geq 0$) be the free variables in N, and $\{y_1, \ldots, y_m, y_{m+1}, \ldots, y_{m+p}\}$ (with $p \geq 0$) be the free variables in $\mathbf{F}[M]$. Since the $\{y_{m+1}, \ldots, y_{m+p}\}$ do not occur in N, they are eventually discarded in the conversion to N and can therefore be substituted by arbitrary closed terms without violating SolF.

We focus on the $\{y_1, \ldots, y_m\}$. The β-nf N is of the form $\lambda z_1 \ldots z_q.h\,M_1 \cdots M_r$ with $q \geq 0$, $r \geq 0$, h the head variable, and $M_1 \in \mathsf{NF}$, ..., $M_r \in \mathsf{NF}$. Since the M_1, ..., M_r are β-nfs, all the variables in N are head variables of some β-nf subterm, and so are the free variables $\{y_1, \ldots, y_m\}$. Let $\{P_{i1}, \ldots, P_{is_i}\}$ (with $i \in \{1, \ldots, m\}$ and $s_i \geq 0$) be the maximal subterms that are in β-nf and that have a particular occurrence of the free variable y_i as the head variable.[7] The $\{P_{11},\dots,P_{1s_1},\dots,P_{m1},\dots,P_{ms_m}\}$ need not be disjoint. For each $i\in\{1,\dots,m\}$ let o_i be the maximum number of operands of y_i in any β-nf subterm P_{ij} having y_i as the head variable:

$$o_i = \max\{\ell_j \mid P_{ij}\equiv \lambda u_1\dots u_t.y_i\,Q_1\cdots Q_{\ell_j} \text{ with } j\le s_i\}$$

We let $T_i\equiv \lambda v_1\dots v_{o_i}\,w.w\,v_1\cdots v_{o_i}$. The y_1, …, y_m can be replaced by the respective T_1, …, T_m without violating SolH, since for any $i\le m$ and $j\le s_i$ we have

$$\begin{array}{rcl}[T_i/y_i]P_{ij} & \equiv & \lambda u_1\dots u_t.(\lambda v_1\dots v_{o_i}\,w.w\,v_1\cdots v_{o_i})Q'_1\cdots Q'_{\ell_j}\\ & =_\beta & \lambda u_1\dots u_t\,v_{\ell_j+1}\dots v_{o_i}\,w.w\,Q'_1\cdots Q'_{\ell_j}\,v_{\ell_j+1}\cdots v_{o_i}\end{array}$$

where $Q'_c\equiv[T_i/y_i]Q_c$ (with $c\le\ell_j$). The term obtained is a hnf in which the free variable y_i no longer occurs, the closed w is now the head variable, and there are additional binding occurrences $\lambda v_{\ell_j}\dots v_{o_i}\,w$ and trailing closed operands $v_{\ell_j}\cdots v_{o_i}$. The term obtained can be proved to be a β-nf by a straightforward induction, since $Q'_c\equiv[T_i/y_i]Q_c$ (with $c\le\ell_j$), and the Q_c are subterms of P_{ij}. Consequently, the term $[T_1/y_1]\dots[T_m/y_m]N$ is a closed β-nf. (Notice that although the P_{ij} may not be disjoint, the substitutions $[T_1/y_1]\dots[T_m/y_m]$ commute by the Substitution Lemma [Bar84, Lemma 2.1.16] because the $\{T_1,\dots,T_m\}$ are closed terms, *i.e.* the $\{y_1,\dots,y_m\}$ do not occur free in them.)

The head context $\mathbf{H}[\ ]$ we are looking for is

$$\begin{array}{rcl}\mathbf{H}[\ ] & \equiv & (\lambda y_1\dots y_m\,y_{m+1}\dots y_{m+p}\,x_1\dots x_n.[\ ])\\ & & \quad T_1\cdots T_m\,\mathbf{I}_1\cdots\mathbf{I}_p\\ & & \quad [T_1/y_1]\dots[T_m/y_m][\mathbf{I}/y_{m+1}]\dots[\mathbf{I}/y_{m+p}]N_1\\ & & \quad \dots\\ & & \quad [T_1/y_1]\dots[T_m/y_m][\mathbf{I}/y_{m+1}]\dots[\mathbf{I}/y_{m+p}]N_k\end{array}$$

where the y_1, …, y_m are substituted respectively by T_i, …, T_m and the operationally irrelevant $\{y_{m+1},\dots,y_{m+p}\}$ are substituted by a closed term (we pick $\mathbf{I}$ but any other closed term would do). By the Substitution Lemma [Bar84, Lemma 2.1.16] the equation $\mathbf{H}[M]=_\beta[T_1/y_1]\dots[T_m/y_m]N$ holds. □

The next example illustrates the proof of Thm. 3.9 by constructing a solving head context from the solving function context $(\lambda x.[\ ])\mathbf{K}$ that solves the term $M\equiv x(y\,z(y\,\mathbf{I}))(t\,\mathbf{\Omega})$.

Example B.1. The term $x(y\,z\,(y\,\mathbf{I}))(\mathbf{\Omega}\,t)$ is solved by the function context $(\lambda x.[\ ])\mathbf{K}$, *i.e.* $(\lambda x.[x(y\,z\,(y\,\mathbf{I}))(\mathbf{\Omega}\,t)])\mathbf{K}=_\beta y\,z(y\,\mathbf{I})$ where $y\,z(y\,\mathbf{I})$ is an open term in β-nf. The free variables of the RHS of the equation are $\{y,z\}$, and the free variables of the LHS are $\{y,z,t\}$.

The maximal subterms in β-nf having y as the head variable are $y\,z(y\,\mathbf{I})$ and $y\,\mathbf{I}$. The maximum number of operands to which the y is applied is $o_y=2$ (*i.e.* the z and the $y\,\mathbf{I}$ in $y\,z(y\,\mathbf{I})$). The maximal subterm in β-nf having z as the head variable is z, with $o_z=0$. Let $T_y\equiv\lambda v_1v_2w.w\,v_1\,v_2$ and $T_z\equiv\lambda w.w$. The solving head context is

$$\begin{array}{rl} & (\lambda yztx.[\ ])(\lambda v_1v_2w.w\,v_1\,v_2)(\lambda w.w)\mathbf{I}([\lambda v_1v_2w.w\,v_1\,v_2/y][\lambda w.w/z][\mathbf{I}/t]\mathbf{K})\\ \equiv & (\lambda yztx.[\ ])(\lambda v_1v_2w.w\,v_1\,v_2)(\lambda w.w)\mathbf{I}\,\mathbf{K}\end{array}$$

[7]Here 'maximal' is used as in Def. 2.3 of [BKKS87], *i.e.* it refers to the subterm ordering. However, notice that different subterms with different particular occurrences of the same variable y_i as the head variable may not be disjoint. Consider the term $\lambda x_1.\lambda x_2.y_1(y_1\,\mathbf{I}\,\mathbf{I})\mathbf{I}\,\mathbf{I}$. Both $P_{11}\equiv\lambda x_1.\lambda x_2.y_1(y_1\,\mathbf{I}\,\mathbf{I})\mathbf{I}\,\mathbf{I}$ and $P_{12}\equiv y_1\,\mathbf{I}\,\mathbf{I}$ are maximal subterms with *each of the two particular occurrences* of y_1 as head variable.

Let us show that it solves the term:

$$
\begin{array}{ll}
& (\lambda yztx.[x(y\,z(y\,\mathbf{I}))(\mathbf{\Omega}\,t)])(\lambda v_1 v_2 w.w\,v_1\,v_2)(\lambda w.w)\mathbf{I}\,\mathbf{K} \\
=_\beta & \{\textit{substitute } y\} \\
& (\lambda ztx.[x((\lambda v_1 v_2 w.w\,v_1\,v_2)z((\lambda v_1 v_2 w.w\,v_1\,v_2)\mathbf{I}))(\mathbf{\Omega}\,t)])(\lambda w.w)\mathbf{I}\,\mathbf{K} \\
=_\beta & \{\textit{substitute rightmost } v_1\} \\
& (\lambda ztx.[x((\lambda v_1 v_2 w.w\,v_1\,v_2)z(\lambda v_2 w.w\,\mathbf{I}\,v_2))(\mathbf{\Omega}\,t)])(\lambda w.w)\mathbf{I}\,\mathbf{K} \\
=_\beta & \{\textit{substitute } z\} \\
& (\lambda tx.[x((\lambda v_1 v_2 w.w\,v_1\,v_2)(\lambda w.w)(\lambda v_2 w.w\,\mathbf{I}\,v_2))(\mathbf{\Omega}\,t)])\mathbf{I}\,\mathbf{K} \\
=_\beta & \{\textit{substitute } v_1 \textit{ and leftmost } v_2\} \\
& (\lambda tx.[x(\lambda w.w(\lambda w.w)(\lambda v_2 w.w\,\mathbf{I}\,v_2))(\mathbf{\Omega}\,t)])\mathbf{I}\mathbf{K} \\
=_\beta & \{\textit{substitute } t\} \\
& (\lambda x.[x(\lambda w.w(\lambda w.w)(\lambda v_2 w.w\,\mathbf{I}\,v_2))(\mathbf{\Omega}\,\mathbf{I})])\mathbf{K} \\
=_\beta & \{\textit{substitute } x \textit{ and convert constant operator}\} \\
& \lambda w.w(\lambda w.w)(\lambda v_2 w.w\,\mathbf{I}\,v_2) \in \mathsf{NF}^0
\end{array}
$$

Appendix C. Genericity Lemma in [Wad76] and [Bar84]

Our statement of the Genericity Lemma (Lemma 3.10 on page 11) is a combination of the following versions. We state them using the term identifiers M and X of Lemma 3.10 for uniformity.

Corollary 5.5 on page 510 of [Wad76]: *Suppose M is unsolvable and $\mathbf{C}[\ ]$ is any context. Then $\mathbf{C}[M]$ has a normal form (a head normal form) iff $\mathbf{C}[X]$ has the same normal form (a similar head normal form) for all terms X.*

Proposition 14.3.24 on page 374 of [Bar84]: *Let $M, N \in \Lambda$ with M unsolvable and N having a nf. Then for all $\mathbf{C}[\ ] \in \Lambda$, $\mathbf{C}[M] =_\beta N \Rightarrow \forall X \in \Lambda\ \ \mathbf{C}[X] =_\beta N$.*

Appendix D. Values are required for substitutivity and confluence

Permitting applications in β_V-nf as members of Val breaks confluence. Such applications would be permissible operands in the conversion rule (β_V). The counter-example used in [Plo75, p.135-136] is $(\lambda x.(\lambda y.z)(x\,\mathbf{\Delta}))\mathbf{\Delta}$. If the application in β_V-nf $(x\,\mathbf{\Delta})$ is in Val then the conversion $(\lambda x.(\lambda y.z)(x\,\mathbf{\Delta}))\mathbf{\Delta} =_{\beta_V} (\lambda x.z)\mathbf{\Delta}$ would be allowed (the innermost redex is converted). From that conversion $(\lambda x.z)\mathbf{\Delta} =_{\beta_V} z$ follows. However, $(\lambda x.(\lambda y.z)(x\,\mathbf{\Delta}))\mathbf{\Delta} =_{\beta_V} (\lambda y.z)(\mathbf{\Delta}\mathbf{\Delta})$ is a valid conversion (the outermost redex is converted), but $(\lambda y.z)(\mathbf{\Delta}\mathbf{\Delta}) =_{\beta_V} z$ does not follow because $\mathbf{\Omega} \equiv \mathbf{\Delta}\mathbf{\Delta}$ is not an application in β_V-nf and it cannot be converted to one.

Permitting arbitrary applications as subjects of substitutions breaks substitutivity. The counter-example used in [Plo75, p.135-136] is to consider $(\lambda x.\mathbf{I})x =_{\beta_V} \mathbf{I}$ and to show that $[\mathbf{\Omega}/x]((\lambda x.\mathbf{I})x) =_{\beta_V} [\mathbf{\Omega}/x]\mathbf{I}$, that is, $(\lambda x.\mathbf{I})\mathbf{\Omega} =_{\beta_V} \mathbf{I}$, does not hold. The LHS has no β_V-nf because the diverging term $\mathbf{\Omega}$ is converted before substitution whereas the RHS is a β_V-nf.

An subtle point unstated in [Plo75, p.135-136] is that permitting applications in β_V-nf as subjects of substitutions also breaks substitutivity even if permissible operands were values. The counter-example is to consider $(\lambda x.\mathbf{I})x =_{\beta_V} \mathbf{I}$ and to show that $[(x\,\mathbf{\Delta})/x]((\lambda x.\mathbf{I})x) =_{\beta_V} [(x\,\mathbf{\Delta})/x]\mathbf{I}$, that is, $(\lambda x.\mathbf{I})(x\,\mathbf{\Delta}) =_{\beta_V} \mathbf{I}$, does not hold. The LHS cannot convert to the RHS because the operand $(x\,\mathbf{\Delta})$ is not permissible.

As a consequence of the last two paragraphs, the substitutivity property in λ_V has the proviso $L \in \mathsf{Val}$ in its statement [Plo75, p.135].

Appendix E. Head and head spine of a term

For ease of reference we collect here the results of [BKKS87] relative to the complete normal order strategy of λK on which we base the analogue results for λ_V in Section 7.1.

A redex of $M \in \Lambda$ is *needed* [BKKS87, p.212] if the redex or its residual is contracted in every reduction sequence starting in M and arriving at a β-nf. The contraction of a needed redex always decrements the length of a normalising reduction sequence. Neededness is an undecidable property, but there exist decidable approximations of the set of needed redexes that can be computed efficiently. The so-called *spine strategies* reduce redexes in several of these decidable approximations of the needed set.

The *head* and *head spine* of a term [BKKS87, Def. 4.2] provide progressively better approximations to the set of *needed redexes* in the term [BKKS87, p.212]. The head is the segment of the abstract syntax tree of the term that starts at the root node and descends through lambda nodes and to the left through operators in applications. The head spine is the segment of the abstract syntax tree that starts at the root node and descends either through lambda nodes or to the left through operators in applications. The head spine of a term includes the head of the term and, recursively, the head of the innermost node reached so far. Fig. 5 illustrates with an example that is further developed after the following formal definition of head and head spine.

In Def. E.1 below we define the functions bn, he, and hs. The head spine of a term is underlined by function hs whose definition we have taken from [BKKS87, Def. 4.2]. The head of a term is underlined by function he that relies on the auxiliary function bn which is related to call-by-name as explained below. The definition of he is based on the definition of the head reduction strategy in [Bar84] that reduces up to hnf. We define head reduction and call-by-name using a reduction semantics in Def. E.3 and Def. E.2.

Definition E.1 (Head and head spine)**.** Functions he and hs underline the head and the head spine of a term respectively.

$$
\begin{array}{rcl}
\mathrm{bn}(x) & = & \underline{x} \\
\mathrm{bn}(\lambda x.B) & = & \underline{\lambda x}.B \\
\mathrm{bn}(M\,N) & = & \mathrm{bn}(M)N \\[4pt]
\mathrm{he}(x) & = & \underline{x} \\
\mathrm{he}(\lambda x.B) & = & \underline{\lambda x}.\mathrm{he}(B) \\
\mathrm{he}(M\,N) & = & \mathrm{bn}(M)N \\[4pt]
\mathrm{hs}(x) & = & \underline{x} \\
\mathrm{hs}(\lambda x.B) & = & \underline{\lambda x}.\mathrm{hs}(B) \\
\mathrm{hs}(M\,N) & = & \mathrm{hs}(M)N
\end{array}
$$

A β-redex is head (resp. head spine) if its outermost lambda is underlined by function he (resp. hs).

Function bn underlines the outermost lambda of the β-redexes that are reduced by the call-by-name strategy of pure λK (Def. E.2). This strategy differs from its homonym in [Plo75] which is for an applied version of the calculus. See [Ses02] for details on the difference.

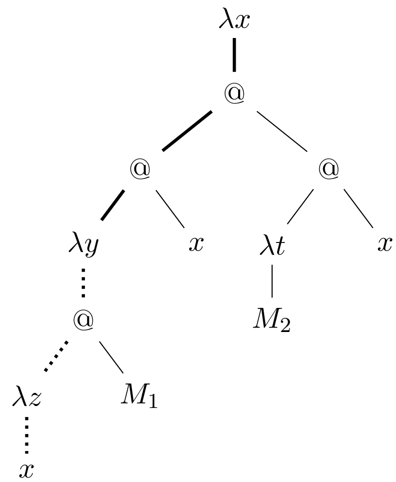

Figure 5: Head (thick edges) and head spine (thick edges and dotted edges) of the term $\lambda x.(\lambda y.(\lambda z.x)M_1)x((\lambda t.M_2)x)$.

As an example, consider the term whose abstract syntax tree is depicted in Fig 5. The head (thick edges in the figure) is underlined in $\underline{\lambda x.(\lambda y}.(\lambda z.x)M_1)x((\lambda t.M_2)x)$. The head spine (thick edges and dotted edges) is underlined in $\underline{\lambda x.(\lambda y.(\lambda z.x})M_1)x((\lambda t.M_2)x)$. The subterm $(\lambda y.(\lambda z.x)M_1)x$ is both a head and a head spine β-redex. The subterm $(\lambda z.x)M_1$ is a head spine β-redex. The subterm $(\lambda t.M_2)x$ is neither a head nor a head spine β-redex.

We now define call-by-name and head reduction using a reduction semantics. Call-by-name is the leftmost strategy that never contracts under lambda abstraction. Head reduction is the leftmost strategy that stops at a hnf. Observe that the reduction contexts of head reduction contain the reduction contexts of call-by-name.

Definition E.2 (Call-by-name strategy)**.** The call-by-name strategy $\rightarrow_{bn}$ is defined by the following reduction semantics:

$$\mathbf{BN}[\ ] \ ::= \ [\ ] \mid \mathbf{BN}[\ ]\,\Lambda$$

$$\mathbf{BN}[(\lambda x.B)N] \rightarrow_{bn} \mathbf{BN}[[N/x]B]$$

Definition E.3 (Head reduction strategy)**.** The head reduction strategy $\rightarrow_{he}$ is defined by the following reduction semantics:

$$\mathbf{HR}[\ ] \ ::= \ [\ ] \mid \mathbf{BN}[\ ]\,\Lambda \mid \lambda x.\mathbf{HR}[\ ]$$

$$\mathbf{HR}[(\lambda x.B)N] \rightarrow_{he} \mathbf{HR}[[N/x]B]$$