JFP **29**, e7, 38 pages, 2019. © Cambridge University Press 2019
doi:10.1017/S0956796819000017

The full-reducing Krivine abstract machine KN simulates pure normal-order reduction in lockstep: A proof via corresponding calculus

ÁLVARO GARCÍA-PÉREZ
IMDEA Software Institute, Campus de Montegancedo s/n, 28223 Pozuelo de Alarcon, Madrid, Spain
(*e-mail:* alvaro.garcia.perez@imdea.org)

PABLO NOGUEIRA
ESNE, University School of Design, Innovation and Technology, Av. de Alfonso XIII, 97, 28016, Madrid, Spain
(*e-mail:* pablo.nogueira@esne.es)

Abstract

We exploit the idea of proving properties of an abstract machine by using a corresponding semantic artefact better suited to their proof. The abstract machine is an improved version of Pierre Crégut's full-reducing Krivine machine KN. The original version works with closed terms of the pure lambda calculus with de Bruijn indices. The improved version reduces in similar fashion but works on closures where terms may be open. The corresponding semantic artefact is a structural operational semantics of a calculus of closures whose reduction relation is purposely a reduction strategy. As shown in previous work, improved KN and the structural operational semantics 'correspond', i.e. both artefacts realise the same reduction strategy. In this paper, we prove in the calculus of closures that the reduction strategy simulates in lockstep (at every reduction step) the complete and standard normal-order strategy (i.e. leftmost reduction to normal form) of the pure lambda calculus. The simulation is witnessed by a substitution function from closures of the closure calculus to pure terms of the pure lambda calculus. Thus, KN also simulates normal-order in lockstep by the correspondence. This result is stronger than the known proof that KN is complete, for in the pure lambda calculus there are complete but non-standard strategies. The lockstep simulation proof consists of straightforward structural inductions, thanks to three properties of the closure calculus we call 'index alignment', 'parameters-as-levels' and 'balanced derivations'. The first two come from KN. Thanks to these properties, a proof in a calculus of closures involving de Bruijn indices and de Bruijn levels is unproblematic. There is no lexical adjustment at binding lookup, on-the-fly alpha-conversion or recursive traversals of the term to deal with bound and free variables as in other calculi. This paper contributes to the framework for environment machines of Biernacka and Danvy a full-reducing open-terms closure calculus, its corresponding abstract machine, and a lockstep simulation proof via a substitution function.

1 Introduction

At the core of programming-language and proof-assistant implementations, we predominantly find an implementation of an *abstract machine* (Diehl *et al.*, 2000). An abstract machine is a 'semantic artefact' in the sense that it is a specification of an operational semantics (Danvy, 2009; Danvy *et al.*, 2011). More concretely, an abstract machine is a

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

first-order state transition function that reduces directly the terms of a core language with the help of auxiliary data structures such as stacks, environments and stores. An abstract machine abstracts away the low-level and platform-specific details of a concrete physical machine, while remaining closer to the latter than other styles of operational semantics such as *structural operational semantics* (Plotkin, 1981), *context-based reduction semantics* (Felleisen, 1987) and *natural semantics* (Kahn, 1987). All these operational-semantics styles provide alternative ways to define the same *reduction strategy* of an *underlying term calculus* (Danvy *et al.*, 2011; Plotkin, 1975).

A reduction strategy specifies a fixed, deterministic order in which to reduce the *redexes* (short for *red*ucible *ex*pressions) of a term. The reduction of a redex, called a *contraction* in the jargon, is typically accompanied by other *administrative* steps, e.g. location of the redex and expansion of intermediate terms.

Semantic artefacts written in different styles of operational semantics *correspond* when they define the same reduction strategy: they *contract redexes in identical order*. The styles may differ in the administrative steps, in the use of auxiliary administrative terms and data structures, and in the level of definition (e.g. one-step versus multiple-step reduction). One way to prove the correspondence between artefacts is by *inter-deriving their implementations by means of program transformations*. For example, an abstract machine implementation can be derived from the implementation of a structural operational semantics or from an evaluator implementation of a natural semantics. See Danvy (2009) for an introduction.

The correspondence between semantic artefacts, and the inter-derivation of the implementations, can be exploited to *prove properties of the reduction strategy using a semantic artefact better suited to their proof*. For instance, proof techniques like structural induction may be more readily applicable in one artefact than in others. The idea of proving properties in a corresponding semantic artefact is not new. A seminal case in point is Plotkin (1975) who, among other things, proved properties of the SECD abstract machine using a corresponding big-step call-by-value evaluator defined in natural semantics.

1.1 Contribution

In this paper, we prove a property of the reduction strategy embodied by one semantic artefact (a full-reducing abstract machine) by proving that property in a corresponding semantic artefact (a structural operational semantics) where we can more readily deploy structural induction. We elaborate the details in the following paragraphs. In Section 1.2 we explain the importance of the contribution.

The full-reducing abstract machine. The abstract machine is an improved version of the original KN machine (Crégut, 2007, p. 217), which is the full-reducing variant of the widely known call-by-name Krivine abstract machine (KAM) (Krivine, 2007). The latter is the basis and inspiration for abstract machines of popular proof-assistants. The original KN abstract machine takes as input a pure lambda calculus term without free variables (i.e. a *closed* term) represented with de Bruijn indices. The machine reduces the term *fully* to a *normal form* (i.e. a term with no redexes) if and only if such normal

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

form exists.[1] The machine works with a term, an environment stack, a continuation stack, and the current nesting level at which it is reducing under lambda. In the terminology of Biernacka & Danvy (2007), KN is an environment-based, push/enter abstract machine.

The improved version of KN was introduced in García-Pérez *et al.* (2013). It combines the term with its environment to make up a closure and therefore works with a closure, a continuation stack and the current nesting level. Due to small but essential modifications, the improved version can work with open terms and requires less information to reduce applications that do not reduce to a redex. These modifications do not affect the reduction strategy: every closed term reducible by original KN is reduced in a similar fashion by improved KN.

We provide a tutorial presentation of both versions of the machine (Section 4). Understanding them properly is essential for understanding the calculus of closures and, consequently, the proof presented in this paper. We also hope to contribute to the popularisation of the original KN machine, which is not as widely known or cited as it deserves. It fully reduces pure lambda-calculus terms with de Bruijn indices simply and effectively. Our improved version is certainly less known.

The structural operational semantics of a calculus of closures. The corresponding semantic artefact is a structural operational semantics that defines the one-step reduction relation of the $\lambda\widetilde{\rho}$ calculus of closures. This calculus was designed in García-Pérez *et al.* (2013) with KN in mind and *its reduction relation is purposely defined to be a reduction strategy.* The proof of the correspondence consists of a program derivation that starts with an implementation of the structural operational semantics and ends with an implementation of the improved version of KN.

The proof of lockstep simulation. The property we prove in this paper is that $\lambda\widetilde{\rho}$'s reduction (and KN by the correspondence) simulates in lockstep the normal-order strategy of the pure lambda calculus, hereafter *pure normal-order*.

The pure normal-order strategy reduces pure lambda calculus terms by contracting the leftmost redex first, understanding 'leftmost' as in Curry & Feys (1958) or as 'leftmost-outermost' by looking at the redex's position in the abstract syntax tree of the term. Pure normal-order satisfies two important properties: (1) it is a *complete* reduction strategy, i.e. finds the normal form of a pure term if and only if such normal form exists; (2) it is a *standard* strategy, i.e. it doesn't reduce a redex to the left of the one just contracted.[2]

Lockstep simulation states that for every pure term M, each step in its normal-order reduction sequence (each involves β-contraction) corresponds, modulo administrative

[1] Some authors use *strong* reduction rather than *full* reduction, but the former can be confused with *strong normalisation* which means something different, namely, that reduction terminates for every term. Also, strong reduction seems the antonym of *weak reduction*, but the latter just refers to 'not reducing abstraction bodies'. For these reasons we use 'full-reduction' throughout the paper, with a dash to make it a technical name.

[2] Some authors use *normal-order* for the weaker strategy that reduces to weak head normal form. We follow Plotkin (1975); Sestoft (2002) and Ronchi Della Rocca & Paolini (2004) who refer to the latter weaker strategy as *call-by-name*. Note that call-by-name's definition in a pure calculus (Sestoft, 2002; Ronchi Della Rocca & Paolini, 2004) differs from its definition in a calculus with primitives (Plotkin, 1975).

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

steps, with one and only one step in the $\lambda\widetilde{\rho}$ reduction sequence starting at the closure $M[\varepsilon]$ where $[\varepsilon]$ is the empty environment. There is a one-to-one correspondence between the pure normal-order step and a segment of the $\lambda\widetilde{\rho}$ reduction sequence that comprises one step involving $\beta\widetilde{\rho}$-contraction (let us call it a 'non-administrative' step) plus zero or more administrative steps around the non-administrative step. The terms before and after contraction in the pure normal-order reduction step respectively correspond, via a substitution function σ from closures to terms, with the closures before and after the $\lambda\widetilde{\rho}$ reduction segment. Thus, the redexes contracted by both steps are 'the same'. Function σ forces the delayed substitutions, effectively simulating capture-avoiding substitution. For illustration, see the diagram on page 30.

Since pure normal-order and $\lambda\widetilde{\rho}$'s reduction are strategies, there is only one respective reduction sequence for M and for $M[\varepsilon]$ in which pure normal-order steps and $\lambda\widetilde{\rho}$ non-administrative steps correspond one-to-one in the way we have described. Since KN reduction and $\lambda\widetilde{\rho}$ reduction correspond, KN also simulates pure normal-order in lockstep.

Simulation versus correspondence. We call it lockstep *simulation* because it is a correspondence between a reduction strategy of a calculus of closures and a reduction strategy of a calculus of pure terms. Thus, a mediating substitution function σ from closures to terms is required to prove the correspondence. It cannot be proven directly by program transformation, for it is not a correspondence in the sense used above for semantic artefacts that embody the same reduction strategy of the same underlying calculus.

Our use of 'simulation' comes from the presence of σ and must not be confused with the notion of simulation or bisimulation in labelled transition systems (Keller, 1976). A bisimulation in the latter sense between terms of the pure lambda calculus and closures of $\lambda\widetilde{\rho}$ might be given by quotienting the set of closures with the equivalence relation induced by administrative reduction. However, we follow a direct approach and provide a commutation theorem that states a one-to-one correspondence between the contraction of redexes.

Simplicity of the proof. The lockstep simulation proof consists of straightforward structural inductions, thanks to the separation of terms and closures, and the separation in reduction judgements between the point at which the de Bruijn level of a formal parameter is pushed on the environment, and the point at which the current nesting level is incremented. Because of this separation the current nesting level remains constant at both sides of a reduction in a judgement, a property we call *balanced derivations*, which facilitates the deployment of structural induction (Section 5).

Carrying out the proof directly on any version of KN is harder. Both versions consist of first-order state transition rules, both use continuation stacks, and the original version separates terms and environments and does not decrement the current nesting level when scoping out of a lambda (Section 4).

As an overview, we list the proof's main definitions and lemmata:

- A well-formedness definition for closures with environments (Definition 6.1).
- A lemma stating that well-formedness is invariant under reduction (Lemma 6.1).
- A definition of the substitution function σ from closures to pure terms (Definition 6.2).

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

- A couple of structural induction principles on closures (Definitions 6.3 and 6.4).
- A lemma stating that de-Bruijn-index shifting is invariant under σ (Lemma 6.4).
- Lemmata connecting σ with meta-level substitution (Lemmas 6.6 and 6.7).
- Lemmata showing that if σ sends a closure to a term in normal form, then administrative reduction reduces that closure to a closure in normal form (Lemmas 6.10 and 6.11).
- A lemma stating that administrative reduction finds the next redex (Lemma 6.12).
- A commutation theorem stating that normal-order (both $\lambda\widetilde{\rho}$ and pure) commute with σ (Theorem 6.13).
- Lockstep simulation as Corollary 6.14 of Theorem 6.13.

1.2 Importance of the contribution

Importance of full-reduction. The importance of full reduction is widely recognised in the theory and implementation of higher-order programming languages and proof-assistants, optimisation by partial evaluation and automated reasoning. See for example Grégoire & Leroy (2002); Crégut (2007); Aydemir *et al.* (2008); Charguéraud (2012); Scherer & Rémy (2015); Accattoli *et al.* (2015). However, full-reducing abstract machines and their properties have surprisingly received less attention. For instance, it is not full-reducing KN but weak-reducing KAM that is often mentioned in the literature in connection with abstract machines of proof-assistants that perform full reduction. For a list of the scant related work on full-reducing abstract machines, see for example Accattoli *et al.* (2015, pp. 4–5).

Stronger correctness property. The property that original KN is complete for closed terms is a known result proven schematically in Crégut (2007, Theorem 6). A closed term has a normal form if and only if the KN machine reduces the term to the normal form. But, this provides no information about *how* KN finds the normal form. Pure normal order finds the normal form of a term in standard fashion. KN could be realising one of the several complete but non-standard strategies of the pure lambda calculus, namely the spine strategy 'hybrid normal order' (Sestoft, 2002, p. 430) or the eval-readback strategy `headNF-byName` (Paulson, 1996, p. 390).

Lockstep simulation shows that $\lambda\widetilde{\rho}$ and improved KN realise pure normal-order and are thus *complete* and *standard* for all terms. If the pure term has a normal form, then its corresponding closure also has a closure normal form, with the former obtained from the latter via σ.

A different property we find in Crégut (2007, Theorem 4) is that given a KN transition between two states, the interpretations of the states in the $\lambda\sigma$ calculus of explicit substitutions (Abadi *et al.*, 1991) are $\lambda\sigma$-convertible. However, there are simply typed terms whose reduction in $\lambda\sigma$ does not terminate. Therefore, $\lambda\sigma$'s reductions are not strictly bound to pure lambda calculus reductions (Melliès, 1995, p. 329).

A proof of lockstep simulation between a call-by-name strategy of $\lambda\sigma$ and call-by-name in the pure lambda calculus is given in Abadi *et al.* (1991, Theorem 3.9). The proof employs a substitution function that maps $\lambda\sigma$-terms to $\lambda\sigma$-normal-forms (Abadi *et al.*, 1991, Section 3.1). For a detailed comparison with calculi of explicit substitutions, see Section 7.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Contribution to the concrete framework for abstract machines. One outcome of the research on inter-derivation by program transformation of semantic artefacts is a framework for environment-based call-by-name/value abstract machines (Curien, 1991; Biernacka & Danvy, 2007). The call-by-name KAM and the call-by-value Categorical Abstract Machine correspond with reduction strategies of the $\lambda\rho$ calculus of closures (Curien, 1991). This calculus is superseded by the $\lambda\widehat{\rho}$ calculus (Biernacka & Danvy, 2007) which amends $\lambda\rho$ in order to specify proper structural operational semantics and derive by program transformation the call-by-name/value abstract machines KAM, CEK and Zinc.

All these abstract machines and reduction strategies are weak-reducing. A substitution function from $\lambda\widehat{\rho}$-terms to pure terms is introduced in Biernacka & Danvy (2007, p. 6:9), but no proof of lockstep simulation of call-by-name/value between $\lambda\widehat{\rho}$ and the pure lambda calculus is given. The present paper contributes to the framework for environment machines a full-reducing calculus, a full-reducing machine and a proof of lockstep simulation via a substitution function.

Vindication of $\lambda\widetilde{\rho}$ as a calculus of closures for full-reduction. The addition of lockstep simulation on top of the correspondence with KN vindicates $\lambda\widetilde{\rho}$ as a calculus of closures for complete and standard full-reduction with a corresponding well-known abstract machine. The simplicity of the lockstep simulation proof by mere structural induction vindicates several key features of $\lambda\widetilde{\rho}$, namely *index-alignment*, *parameters-as-levels* and *balanced derivations*. The first two are borrowed from KN and are explained in Sections 2 and 4. The last one is explained in Section 5. Thanks to these features, $\lambda\widetilde{\rho}$ reduces terms fully without α-conversion, lexical adjustments at binding lookup, or recursive traversals of the term to adjust indices, as is the case in other calculi and machines (Section 7). KN shows that terms can be reduced fully and effectively with a judicious combination of de Bruijn indices and de Bruijn levels. In $\lambda\widetilde{\rho}$ it has its fitting corresponding calculus.

Lockstep simulation proof between improved KN and a substitution-based version of KN. In García-Pérez & Nogueira (2014, Figure 9) we present a full-reducing *substitution-based* eval/apply abstract machine that works with open *pure* terms. The machine uses plain capture-avoiding substitution and carries no environment. What is of interest here is that the derivation starts from a structural operational semantics of pure normal-order, but one written in the so-called 'hybrid style'. We state in that paper that the derived machine is a substitution-based open-(and pure)-terms version of KN. Lockstep simulation proves that statement. The substitution-based machine and pure normal-order correspond by the program transformation shown in that paper. Environment-based improved KN simulates in lockstep pure normal-order as shown in this paper. Therefore, the substitution-based machine and the environment-based improved KN simulate each other in lockstep.

1.3 Structure and prerequisites of the paper

Section 2 Introduces the notion of current nesting level and the convention on de Bruijn indices and levels required to understand the index alignment and parameters-as-levels features of KN and $\lambda\widetilde{\rho}$. Unfortunately, de Bruijn indices and levels are usually taken for granted, but there are several design choices, and the one chosen by KN illuminates the machine's rationale.

Section 3 Reintroduces the structural operational semantics of pure normal-order.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Section 4 Provides a tutorial presentation of original and improved KN. A proper understanding of the machines is a prerequisite for understanding the $\lambda\widetilde{\rho}$ calculus and, consequently, the proof of lockstep simulation.

Section 5 Reintroduces $\lambda\widetilde{\rho}$.

Section 5.1 Discusses the correspondence between improved KN and $\lambda\widetilde{\rho}$.

Section 6 Presents the proof of lockstep simulation.

Section 7 Discusses related work, in particular other derivations of full-reducing machines, calculi with lexical adjustments, calculi of explicit substitutions and locally nameless representations of terms.

The original and improved versions of KN, the $\lambda\widetilde{\rho}$ calculus, the structural operational semantics of pure normal-order and the proof of correspondence by program transformation between improved KN and $\lambda\widetilde{\rho}$ were introduced in García-Pérez *et al.* (2013). In this paper, we provide a much better and clearer tutorial presentation of the machines and the calculus. We do not discuss the details of the proof of correspondence. We are mainly concerned with proving lockstep simulation.

We assume readers are familiar with lambda calculus concepts and notation as found in the standard reference (Barendregt, 1984), namely syntax of lambda terms, meta-linguistic conventions (lowercase letters range over variables, uppercase letters range over terms, etc.), application associates to the left and has higher precedence than abstraction, bound and free variables, identity of terms up to α-conversion (hereafter written $\equiv$), capture-avoiding substitution, redexes, normal forms, reduction relations $\longrightarrow_\beta$ and $\longrightarrow_\beta^*$, and reduction strategies as partial functions on terms. A reduction strategy can be defined as a one-step (small-step) or as a final-step (big-step) partial function. The latter corresponds to iterations (compositions) of the former until an irreducible term is reached.

Unlike Barendregt (1984), we use Extended Backus-Naur Form (EBNF) grammars to define sets. For a variable x and terms N and B, we write the capture-avoiding substitution function as $[N/x]B$. This notation flows in English when read from left to right: 'substitute N for free x in B' and embodies 'substitute-for' rather than 'replace-by'. It is also the original notation in Curry & Feys (1958).

2 Current nesting level, de Bruijn indices and de Bruijn levels

For precision we distinguish the *binding occurrence* of a bound variable (the occurrence between the λ and the dot in an abstraction) from the *applied occurrence* of a bound variable (an occurrence in the abstraction's body that is not a binding occurrence). Free variables have no binding occurrence. For example, parsing $\lambda x.(\lambda y.xz)y$ from left to right, there is a binding occurrence of x, a binding occurrence of y, an applied occurrence of x, a free variable z and a free variable y.

In a nameless representation of terms there are no binding occurrences, and applied occurrences refer unambiguously to their binding lambda. Two nameless representations are de Bruijn *indices* and de Bruijn *levels* (de Bruijn, 1972). Applied occurrences are represented by natural numbers. The set of terms becomes $\Lambda ::= n \mid \lambda.\Lambda \mid \Lambda\ \Lambda$, where we overload $n \in \mathbb{N}$ in the grammar as non-terminal for $\mathbb{N}$. The following picture helps to explain how the numbers are determined.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$
\begin{array}{rcl}
|\varepsilon| & = & 0 \\
|x:\rho| & = & 1+|\rho| \\
\\
ind(x,x:\rho) & = & 0 \\
ind(x,y:\rho) & = & 1+ind(x,\rho) \\
\\
lev(x,x:\rho) & = & |x:\rho| \\
lev(x,y:\rho) & = & lev(x,\rho) \\
\\
\varphi(f,x,\rho) & = & f(x,\rho) \\
\varphi(f,\lambda x.B,\rho) & = & \lambda.\varphi(f,B,x:\rho) \\
\varphi(f,MN,\rho) & = & (\varphi(f,M,\rho))\,(\varphi(f,N,\rho))
\end{array}
\qquad
\begin{array}{rcl}
[T/n]m & = & \begin{cases} \uparrow_0^n T & \text{if } m=n \\ m-1 & \text{if } m>n \\ m & \text{otherwise} \end{cases} \\
{[T/n](\lambda.B)} & = & \lambda.[T/n+1]B \\
{[T/n](MN)} & = & ([T/n]M)\,([T/n]N) \\
\\
\uparrow_k^n m & = & \begin{cases} m+n & \text{if } m \geq k \\ m & \text{otherwise} \end{cases} \\
\uparrow_k^n(\lambda.B) & = & \lambda.\uparrow_{k+1}^n B \\
\uparrow_k^n(MN) & = & (\uparrow_k^n M)\,(\uparrow_k^n N)
\end{array}
$$

Fig. 1. (*Left*) Environment size, index function *ind*, level function *lev* and encoding function φ. (*Right*) Capture-avoiding substitution and shift functions for terms with de Bruijn indices.

$$
\overbrace{\lambda y.\ \ldots\ \lambda x.\underbrace{\ \ldots\ \lambda z.}_{ind(x,\,\rho)}(\ldots\ x\ \ldots)}^{lev(x,\,\rho)}
\qquad
\underbrace{\qquad\qquad\qquad\qquad}_{|\rho|}
$$

The picture shows a term with several lambdas. The term in parenthesis has no lambdas and has an applied occurrence of x. An environment stack of binding occurrences ρ can be constructed by traversing the term from the root, down the syntax tree, to the last lambda. More precisely, let $\rho ::= \varepsilon \mid x:\rho$ be the set of environment stacks constructed by ε (empty stack) and by $x:\rho$ (push the binding occurrence x on top of the stack ρ). The value of ρ in the picture is $z:\ldots:x:\ldots:y:\varepsilon$. We write $|\rho|$ for the size of ρ, calculated as shown on the left-hand side of Figure 1.

At the point of an applied occurrence, the *current nesting level*, or number of lambdas gone under, is the size of the environment stack at that point. The nesting level 0 (empty environment stack) is for the scope with no lambdas.

In a de-Bruijn-*indices* representation, the applied occurrence of x is represented by a natural number that counts the number of nesting lambdas from the lambda closest to the applied occurrence of x, up the syntax tree, to the lambda binding x. Older presentations start counting at 1, including (de Bruijn, 1972), but these days counting starts at 0. At the applied occurrence of x, its de Bruijn index is calculated by $ind(x,\rho)$ which delivers the position of the first occurrence of x on the environment stack, with 0 the index of the top of the stack. The position from the top in the environment stack is what *index* means in de Bruijn (1972, p. 392), with no other mention of it in that paper.[3] The definition of function *ind* is shown on the left-hand side of Figure 1. For example, $\lambda x.(\lambda y.x\,y)x$ and all its α-convertible terms are uniquely represented using de Bruijn indices by $\lambda.(\lambda.1\ 0)0$.

In the alternative de-Bruijn-*levels* representation, the applied occurrence of x is represented by the number of nested lambdas from the root of the term, down the syntax tree, to the lambda binding x. Like Crégut (2007), we begin counting at 1, leaving 0 for the scope with no lambdas, as with the current nesting level. At the applied occurrence of x, its de Bruijn level is calculated by $lev(x,\rho)$ which pops the environment stack until it finds x at the top and then returns the size of the remaining environment. This is the size of the environment constructed from the root to the binding occurrence of x, or equivalently, the

[3] The equivalent number of inside-out nesting lambdas that we are using for intuition is not called *index* but *reference depth* in de Bruijn (1972, p. 383).

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

value of the current nesting level at the point of the *binding* occurrence. Thus, when going under lambda, the de Bruijn level for the *applied* occurrence is one plus the current nesting level. The definition of *lev* is shown in Figure 1. For example, $\lambda x.(\lambda y.x\,y)x$ and all its α-convertible terms are uniquely represented using de Bruijn levels by $\lambda.(\lambda.1\ 2)1$.

With de Bruijn indices, free variables are represented by out-of-bounds indices pointing to imaginary binding lambdas. Their value is fixed by convention. In other words, the representation is parametric on an initial environment for free variables. For example, $(\lambda x.y)z$ can be uniquely represented by $(\lambda.1)1$ with initial environment $y : z : \varepsilon$ such that $ind(y, x : y : z : \varepsilon) = 1$ and $ind(z, y : z : \varepsilon) = 1$. It can also be uniquely represented by $(\lambda.2)0$ with initial environment $z : y : \varepsilon$ such that $ind(y, x : z : y : \varepsilon) = 2$ and $ind(z, z : y : \varepsilon) = 0$. We omit the representation of free variables with de Bruijn levels because they are used only for bound variables by KN and $\lambda\widetilde{\rho}$.

Given a lambda term T and an initial environment ρ_0 such that $x \in \mathrm{FV}(T)$ if and only if $x \in \rho_0$, the expression $\varphi(ind, T, \rho_0)$ delivers the de-Bruijn-indices representation of T, and $\varphi(lev, T, \rho_0)$ delivers the de-Bruijn-levels representation of T. The definition of φ is shown in Figure 1. Observe that the environment only increases when going under lambda, and that it is distributed in applications such that the operator and the operand are at the same nesting level.

It is important to notice that, with de Bruijn indices, the environment stack increases when φ goes under lambda, so different applied occurrences of the same variable are represented by different indices depending on the nesting level. For example, x in $\lambda x.(\lambda y.x\,y)x$ is both 1 and 0 in $\lambda.(\lambda.1\ 0)0$. For this reason, the capture-avoiding substitution function $[T/n]M$ (Figure 1 top right) for this nameless representation has to shift the free variables of T when substituting T, and increment the index of the substituted variable as it goes under lambda. The shift function $\uparrow_k^n$ (Figure 1 bottom right) carries a cutoff parameter k informing of the number of lambdas gone under.

In contrast, different applied occurrences of the same variable are represented by the same de Bruijn level, because the size of the environment at which the variable is at the top is the same, or equivalently, because the current nesting level at the point of the binding occurrence is the same. For example, x in $\lambda x.(\lambda y.x\,y)x$ is 1 in $\lambda.(\lambda.1\ 2)1$.

It is a simple exercise to prove by induction that for every binding occurrence $x \in \rho$, then $lev(x, \rho) + ind(x, \rho) = |\rho|$. Thus, the de Bruijn index of an applied occurrence can be calculated from its de Bruijn level by subtracting the latter from the current nesting level. That is, $ind(x, \rho) = |\rho| - lev(x, \rho)$. That different applied occurrences have the same de Bruijn level, and that de Bruijn indices can be calculated by subtracting from the current nesting level the de Bruijn level for the variable is what motivates KN's and $\lambda\widetilde{\rho}$'s index alignment and parameters-as-levels properties discussed in Sections 4 and 5.

3 Pure normal-order

Pure normal-order is the standard and complete reduction strategy of the pure lambda calculus that reduces terms to normal form. When the strategy finds a leftmost redex $(\lambda.B)N$ it contracts that redex and not the redexes in B. To allow for this, pure normal-order reduces operators in applications weakly to weak head normal form.[4] The set WHNF of weak

[4] There is no redundancy in the sentence. Weak reduction means 'not reducing abstractions', which by itself does not deliver weak head normal forms as results.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$\begin{array}{lcll} \Lambda & ::= & n \mid \lambda.\Lambda \mid \Lambda\,\Lambda & \text{terms with de Bruijn indices} \\ \text{WHNF} & ::= & \lambda.\Lambda \mid n\,\{\Lambda\}^* & \text{weak head normal forms} \\ \text{NF} & ::= & \lambda.\text{NF} \mid n\,\{\text{NF}\}^* & \text{normal forms} \end{array}$$

$$\frac{}{(\lambda.B)N \longrightarrow_{no} [N/0]B}\ (\beta)$$

$$\frac{M \notin \text{WHNF} \quad M \longrightarrow_{no} M'}{MN \longrightarrow_{no} M'N}\ (\mu 1) \qquad \frac{M \in \text{WHNF} \quad M \notin \lambda.B \quad M \longrightarrow_{no} M'}{MN \longrightarrow_{no} M'N}\ (\mu 2)$$

$$\frac{M \in \text{NF} \quad M \notin \lambda.B \quad N \longrightarrow_{no} N'}{MN \longrightarrow_{no} MN'}\ (\nu) \qquad \frac{B \longrightarrow_{no} B'}{\lambda.B \longrightarrow_{no} \lambda.B'}\ (\xi)$$

Fig. 2. Structural operational semantics of pure normal-order. The sets of terms involved are defined at the top of the figure.

head normal forms and the set NF of normal forms are defined in Extended Backus-Naur Form at the top of Figure 2. Weak head normal forms consist of arbitrary abstractions and terms of the form n, $n\ \Lambda$, $n\ \Lambda\ \Lambda$, etc., which according to convention associate as n, $(n\ \Lambda)$, $((n\ \Lambda)\ \Lambda)$, etc. These are the so-called *neutral terms* (*neutrals*, for short) because they are variables or applications that cannot reduce to a redex. Normal forms are terms without redexes, that is, abstractions and neutrals in normal form. The reduction of abstractions and neutrals to normal form is what characterises full reduction.

Figure 2 shows a structural operational semantics that defines pure normal-order as a one-step partial function. Alternative definitions can be given for this strategy, e.g. Pierce (2002). There is no rule for indices because they are normal forms. There are four rules for applications: redex contraction (β), and compatibility rules ($\mu 1$), ($\mu 2$), and (ν) to locate the redex within an arbitrary term.

Rule ($\mu 1$) applies when the redex is in the operator M which is not in WHNF. (The last antecedent in this and other rules indicate that contraction takes place if a redex is found.) Rule ($\mu 2$) applies when the redex is in the operator which is in WHNF but is not an abstraction. If the operator is in WHNF and is an abstraction, then (β) would be applicable. Rule (ν) applies when the redex is in the operand N and the operator M is already in NF but is not an abstraction, meaning MN is a neutral with M already in NF. Although NF $\subset$ WHNF, rules ($\mu 2$) and (ν) do not overlap because the third antecedent of ($\mu 2$) is not the case when $M \in$ NF. There is a last compatibility rule for abstractions (ξ) which applies when the redex is under lambda. It is the presence of rule (ξ) for reducing abstractions, and of rules ($\mu 2$) and (ν) for reducing neutrals, that distinguishes pure normal-order from the weaker pure call-by-name. The splitting into two rules ($\mu 1$) and ($\mu 2$) in this presentation is to make explicit the dependency of pure normal-order on pure call-by-name (García-Pérez & Nogueira, 2014).

The following example illustrates pure normal-order reduction, where the redex contracted at each step is underlined.

Example 3.1 (Pure normal-order reduction of a closed term with two redexes)

$$\lambda x.x\underline{((\lambda y.y)((\lambda t.t)x))} \longrightarrow_{no} \lambda x.x\underline{((\lambda t.t)x)} \longrightarrow_{no} \lambda x.x\,x$$

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$
\begin{array}{lcll}
\Lambda & ::= & n \mid \lambda.\Lambda \mid \Lambda\,\Lambda & \text{terms with de Bruijn indices} \\
\mathsf{C} & ::= & \Lambda[\rho] \mid \overline{n} \mid \lfloor \Lambda, \ell \rfloor & \text{closures} \\
\rho & ::= & \varepsilon \mid \mathsf{C} : \rho & \text{environment stack} \\
S & ::= & \varepsilon \mid \Lambda[\rho] : S \mid \lambda : S \mid \lfloor \Lambda, \ell \rfloor : S & \text{continuation stack}
\end{array}
$$

$$
\begin{array}{lcccccccl}
(1) & & & & & T & \longrightarrow & (\ T[\varepsilon], & \varepsilon, \ 0\ ) \\
(2) & (\ (n+1)[\mathsf{C} : \rho], & S, & \ell\ ) & \longrightarrow & (\ n[\rho], & S, & \ell\ ) \\
(3) & (\ 0[\mathsf{C} : \rho], & S, & \ell\ ) & \longrightarrow & (\ \mathsf{C}, & S, & \ell\ ) \\
(4) & (\ (MN)[\rho], & S, & \ell\ ) & \longrightarrow & (\ M[\rho], & N[\rho] : S, & \ell\ ) \\
(5) & (\ (\lambda.B)[\rho], & N[\rho'] : S, & \ell\ ) & \longrightarrow & (\ B[N[\rho'] : \rho], & S, & \ell\ ) \\
(6) & (\ (\lambda.B)[\rho], & S, & \ell\ ) & \longrightarrow & (\ B[\overline{\ell+1} : \rho], & \lambda : S, & \ell+1\ ) \\
(7) & (\ \overline{n}, & S, & \ell\ ) & \longrightarrow & (\ \lfloor \ell - n, \ell \rfloor, & S, & \ell\ ) \\
(8) & (\ \lfloor M, \ell \rfloor, & N[\rho] : S, & \ell'\ ) & \longrightarrow & (\ N[\rho], & \lfloor M, \ell \rfloor : S, & \ell\ ) \\
(9) & (\ \lfloor B, \ell \rfloor, & \lambda : S, & \ell'\ ) & \longrightarrow & (\ \lfloor \lambda.B, \ell \rfloor, & S, & \ell'\ ) \\
(10) & (\ \lfloor N, \ell \rfloor, & \lfloor M, \ell' \rfloor : S, & \ell''\ ) & \longrightarrow & (\ \lfloor MN, \ell' \rfloor, & S, & \ell''\ ) \\
(11) & (\ \lfloor T, \ell \rfloor, & \varepsilon, & \ell'\ ) & \longrightarrow & T & & &
\end{array}
$$

Fig. 3. A version of KN that works with closures with closed terms (García-Pérez *et al.*, 2013, p. 89).

4 The KN machine and its improved open-terms version

The original KN machine (Crégut, 2007, p. 217) consists of first-order transition rules that operate on a quadruple consisting of a term, an environment stack, a continuation stack and the current nesting level. By attaching the term to its environment stack to make up a closure we have an equivalent version of the machine, shown in Figure 3, that operates on a closure C, a continuation stack S and the current nesting level ℓ. The syntax of terms, closures and both stacks is defined at the top of the figure.

We start with the syntax of closures. The first case $\Lambda[\rho]$ are *proper* closures, i.e. terms attached to environments ρ that stack up operand closures.[5] Following convention, we elide '$:\varepsilon$' in non-empty environments and write $M[N[\rho]]$ instead of $M[N[\rho] : \varepsilon]$. If ρ is empty, then we write $M[N[\varepsilon]]$. Proper closures are standard in closure calculi and should be relatively clear.

The remaining two cases are uncommon. The second case $\overline{n}$ are de Bruijn levels of formal parameters (of binding occurrences). The over-line is to distinguish them notationally from de Bruijn indices n. The last case $\lfloor \Lambda, \ell \rfloor$ are result terms (which thus carry no environment) embedded with a nesting level. We explain these constructs below along with the rules of Figure 3.

We briefly describe the stack S. It can be empty (same symbol as empty environment) or has at its top one of the following: an operand closure $\Lambda[\rho]$, the control character λ which indicates that the machine is currently reducing under lambda, or a result term embedded with a level $\lfloor \Lambda, \ell \rfloor$.

In rule (1), the machine takes an input closed term T and constructs an initial triple consisting of a proper closure (T with empty environment), an empty continuation stack and the initial nesting level.

Rules (2)–(5) explain the choice of de Bruijn indices for terms. In Rules (2) and (3), the machine reduces proper closures of the form $n[\rho]$ by looking up the nth binding on the environment ρ. The de Bruijn index n acts as a lexical offset. Recall from Section 2 that, with de Bruijn indices, 0 points to the closest lambda, so 0 is the *index* of the top of

[5] Do not confuse with the environments of Section 2 that stack up binding occurrences.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

the environment stack. The end's index is $|\rho| - 1$. Terms are closed and thus $n < |\rho|$. In rule (4), the machine reduces applications $(MN)[\rho]$ by distributing the environment over term application and pushing the operand closure on the continuation stack. In rule (5), the machine reduces a redex: the operator is an abstraction closure and there is an operand closure at the top of the continuation stack. The operand closure is moved from the top of the continuation stack to the top of the environment stack. A now-free occurrence of 0 within B (which points to the lambda of $\lambda.B$) is the index for the $N[\rho']$ operand in ρ. The environments ρ and ρ' could be different because the abstraction closure $(\lambda.B)[\rho]$ may come from reducing an operator $M[\rho]$ on the right-hand side of rule (4).[6]

For example, from an initial triple $(\,((\lambda.(\lambda.B)M)N)[\varepsilon],\ \varepsilon,\ 0\,)$ the machine eventually arrives at the triple $(\,B[M[N[\varepsilon]] : N[\varepsilon]],\ \varepsilon,\ 0\,)$ such that the now-free occurrences of 0 and 1 in B respectively point to $M[N[\varepsilon]]$ and $N[\varepsilon]$.

Rules (6) and (7) explain how KN reduces under lambda. In rule (6), there is an abstraction closure to reduce, with no operand closure at the top of the continuation stack. (This rule applies when rule (5) fails to match.) The machine has to go under lambda and reduce the abstraction's body. Suppose it simply pushed the environment down the body: $\lambda.B[\rho]$. Within B's scope the occurrences of 0 that refer to that very binding λ are *non-free* and must not serve as lexical offsets to the environment. Any proper closure of the form $0[\rho']$ eventually constructed in later triples for those non-free occurrences must be reduced to 0 by discarding the environment. And similarly with other non-free occurrences when going deeper under lambda inside B. One way to deal with such indices is *lexical adjustment*, i.e. to use some extra lexical offset parameter to adjust indices when retrieving bindings from the environment in rules (2) and (3). However, KN does something simpler. In rule (6), it removes the λ from the abstraction and pushes the de Bruijn *level* of the abstraction's formal parameter on the environment. As explained in Section 2, the value of this de Bruijn level is easily calculated as $\ell + 1$, which explains why KN carries ℓ. In the same rule (6), the machine also pushes a λ symbol on the continuation stack, and increments the current nesting level to mark that it is reducing under lambda. Given a proper closure of the form $n[\rho']$ eventually constructed for a non-free occurrence of a de Bruijn index n that points to a lambda on the continuation stack, the index also points to its de Bruijn level on the environment stack. In other words, the index deserves its name because it points to a binding on the environment stack (Section 2).

The following picture illustrates this for an abstraction with $m + n$ lambdas. The term in parenthesis has no lambdas and has an occurrence of the de Bruijn index n, which is bound by the nth lambda to the left. When the machine goes under $m + n$ lambdas, then n points to the nth binding on the environment, namely $\overline{m}$. Compare with the picture in Section 2.

$$\overbrace{\lambda.\ \ldots\ \lambda.}^{m}\ \underbrace{\ldots\ \lambda.}_{n}\,(\ldots\ n\ \ldots) \qquad\qquad \ell = 0$$

$$(\ldots\ n\ \ldots)[\underbrace{\overline{m+n} : \ldots : \overline{m}}_{n} : \ldots : \overline{1}\,] \qquad\qquad \ell = m + n$$

[6] Rule (5) could be written with $(\lambda.B)[\rho']$ and $N[\rho]$ to indicate that the abstraction closure's environment is what may change. But, then many more visually less pleasant ρ' symbols would have to be consistently written in other rules.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

As explained in Section 2, the looked-up de Bruijn level $\overline{m}$ can be easily converted to the de Bruijn index by subtracting $\overline{m}$ from the current nesting level ℓ, that is, $(m+n)-\overline{m}=n$.

The current nesting level is only incremented when going under lambda in rule (6) and remains unaffected when reducing redexes in rule (5). Compare the following picture to the one above.

$$\overbrace{\lambda.\ \ldots\ \lambda.}^{m}(\underbrace{\lambda.\ \ldots\ \lambda.}_{n}n)M \qquad \ell=0$$

$$((\underbrace{\lambda.\ \ldots\ \lambda.}_{n}n)M)[\,\overline{m}:\ \ldots\ :\overline{1}\,] \qquad \ell=m$$

$$(\underbrace{\lambda.\ \ldots\ \lambda.}_{n-1}n)[M[\overline{m}:\ \ldots\ :\overline{1}]\,:\overline{m}:\ \ldots\ :\overline{1}\,] \qquad \ell=m$$

$$n[\overline{m+n-1}\,:\ \ldots\ :\,\overline{m+1}\,:\,M[\overline{m}:\ \ldots\ :\overline{1}]\,:\overline{m}:\ \ldots\ :\overline{1}\,] \qquad \ell=m+n-1$$

$$M[\overline{m}:\ \ldots\ :\overline{1}] \qquad \ell=m+n-1$$

In the third row, the operand closure is pushed on the environment but the current nesting level remains m. In the last row, the binding for n is looked up on the environment. The current nesting level remains $m+n-1$.

We are in a position to explain the two main properties that characterise the KN machine:

Index alignment: All de Bruijn indices deserve their name because they point to a binding on the environment.
Parameters-as-levels: The binding for a non-free de Bruijn index is its corresponding de Bruijn level. The latter is pushed on the environment and its value is simply $\ell+1$. When a de Bruijn level is looked up on the environment stack, the original de Bruijn index can be calculated back using the formula 'current-nesting-level minus (looked-up) de-Bruijn-level equals (non-free) de-Bruijn-index'.

The computed de Bruijn index cannot be left as is (it is a term, not a closure), nor can it be returned in a proper closure with an empty environment ($n<|\rho|$ must hold). A result term $\lfloor \ell-n,\ \ell \rfloor$ is constructed in rule (7). The embedding with the current nesting level is due to the peculiar way in which original KN reduces neutrals.

Recall from Section 3 that neutrals are applications of the form $n\,N_1\ \cdots\ N_k$ with a head variable (de Bruijn index) n and with $k>0$ operands. All the operands have to be reduced at the same nesting level as the head variable. However, KN increments the current nesting level when it goes under lambda in rule (6), *but does not decrement the level when scoping out of it* in rule (9). Suppose KN goes under lambda and increments the current nesting level when reducing operand N_i. That value is invalid for operands $N_{i+1}, \ldots, N_k$. By embedding result terms with their nesting level, the appropriate value can be recovered from the embedded result term at the top of the continuation stack, which is the result of reducing the previous term (head variable or previous operand).

In rule (8), the machine moves an embedded result term to the continuation stack and continues reducing the next operand, resetting the current nesting level to that of the embedded result term. In rule (9), it scopes out of lambda and builds the embedded result

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$
\begin{array}{llll}
 & & \lambda.0(\lambda.M)N & \\
(1) & \to & (\ (\lambda.0(\lambda.M)N)[\varepsilon], & \varepsilon,\ 0\ ) \\
(6) & \to & (\ (0(\lambda.M)N)[\overline{1}], & \lambda,\ 1\ ) \\
(4) & \to & (\ (0(\lambda.M))[\overline{1}], & N[\overline{1}] : \lambda,\ 1\ ) \\
(4) & \to & (\ 0[\overline{1}], & (\lambda.M)[\overline{1}] : N[\overline{1}] : \lambda,\ 1\ ) \\
(3) & \to & (\ \overline{1}, & (\lambda.M)[\overline{1}] : N[\overline{1}] : \lambda,\ 1\ ) \\
(7) & \to & (\ \lfloor 1-1, 1 \rfloor, & (\lambda.M)[\overline{1}] : N[\overline{1}] : \lambda,\ 1\ ) \\
(8) & \to & (\ (\lambda.M)[\overline{1}], & \lfloor 0, 1 \rfloor : N[\overline{1}] : \lambda,\ 1\ ) \\
(6) & \to & (\ M[\overline{2}, \overline{1}], & \lambda : \lfloor 0, 1 \rfloor : N[\overline{1}] : \lambda,\ 2\ ) \\
 & \ldots\ldots & & \\
\ldots & \to & (\ \lfloor M', 2 \rfloor, & \lambda : \lfloor 0, 1 \rfloor : N[\overline{1}] : \lambda,\ 2\ ) \\
(9) & \to & (\ \lfloor \lambda.M', 2 \rfloor, & \lfloor 0, 1 \rfloor : N[\overline{1}] : \lambda,\ 2\ ) \\
(10) & \to & (\ \lfloor 0(\lambda.M'), 1 \rfloor, & N[\overline{1}] : \lambda,\ 2\ ) \\
(8) & \to & (\ N[\overline{1}], & \lfloor 0(\lambda.M'), 1 \rfloor : \lambda,\ 1\ ) \\
 & \ldots\ldots & &
\end{array}
$$

Fig. 4. Partial reduction trace of original KN for the term $\lambda.0(\lambda.M)N$ which has a neutral as body. Assume no abstraction occurs in M and that it reduces to M'.

abstraction. In rule (10), it pops out of the continuation stack the embedded result operator and constructs the embedded result application with the nesting level taken from the embedded result operator. The rules (8) and (10) are thus involved in resetting the current nesting level. The former when there is an operand closure that needs to be reduced on top of the continuation stack, and the latter when a neutral application result is constructed.

The last rule (11) ends the execution when the control stack is empty.

For illustration, consider the input term $\lambda.0(\lambda.M)N$ which has a neutral as body. Figure 4 shows its partial reduction trace. Assume that the reduction of M does not go under lambda (the current nesting level is not incremented) and that the result of that reduction is M'. The rule applied at each step is shown on the left of the arrow. Observe in the application of rule (10) that the current nesting level is reset to that of the embedded result term on top of the continuation stack.

Figure 5 shows a complete reduction trace. Compare it to the pure normal-order reduction of the same input term in Example 3.1.

4.1 Improved open-terms KN

A slightly improved open-terms version of KN is easily obtained by letting de Bruijn indices point out of bounds, and by decrementing the current nesting level when scoping out of lambda, so that the result terms need not be embedded with the current nesting level. The rules are shown in Figure 6.

The difference with Figure 3 is the addition of rule (A) for out-of-bounds indices, the decrementing of the current nesting level in rule (9) when scoping out of lambda, and the absence of the current nesting level in result terms in rules (7), (8) and (9). All these rules are highlighted in the figure for comparison with Figure 3.

In a proper closure $n[\rho]$ the de Bruijn index n now acts as a lexical offset that points to its closure binding in the environment when $n < |\rho|$, or points out of bounds when $n \geq |\rho|$,

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$
\begin{array}{lll}
 & & \lambda.0((\lambda.0)((\lambda.0)0)) \\
(1) & \to & (\quad (\lambda.0((\lambda.0)((\lambda.0)0)))[\varepsilon], \qquad\qquad \varepsilon, \quad 0\ ) \\
(6) & \to & (\quad (0((\lambda.0)((\lambda.0)0)))[\overline{1}], \qquad\qquad \lambda, \quad 1\ ) \\
(4) & \to & (\quad 0[\overline{1}], \qquad ((\lambda.0)((\lambda.0)0))[\overline{1}] : \lambda, \quad 1\ ) \\
(2) & \to & (\quad \overline{1}, \qquad ((\lambda.0)((\lambda.0)0))[\overline{1}] : \lambda, \quad 1\ ) \\
(7) & \to & (\quad \lfloor 0,1 \rfloor, \qquad ((\lambda.0)((\lambda.0)0))[\overline{1}] : \lambda, \quad 1\ ) \\
(8) & \to & (\quad ((\lambda.0)((\lambda.0)0))[\overline{1}], \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(4) & \to & (\quad (\lambda.0)[\overline{1}], \qquad ((\lambda.0)0)[\overline{1}] : \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(5) & \to & (\quad 0[((\lambda.0)0)[\overline{1}] : \overline{1}], \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(2) & \to & (\quad ((\lambda.0)0)[\overline{1}], \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(4) & \to & (\quad (\lambda.0)[\overline{1}], \qquad 0[\overline{1}] : \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(5) & \to & (\quad 0[0[\overline{1}] : \overline{1}], \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(2) & \to & (\quad 0[\overline{1}], \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(2) & \to & (\quad \overline{1}, \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(7) & \to & (\quad \lfloor 0,1 \rfloor, \qquad \lfloor 0,1 \rfloor : \lambda, \quad 1\ ) \\
(10) & \to & (\quad \lfloor 00,1 \rfloor, \qquad \lambda, \quad 1\ ) \\
(9) & \to & (\quad \lfloor \lambda.00,1 \rfloor, \qquad \varepsilon, \quad 1\ ) \\
(11) & \to & \lambda.00
\end{array}
$$

Fig. 5. Complete reduction trace of the original KN for the term $\lambda.0((\lambda.0)((\lambda.0)0))$ that encodes $\lambda x.x((\lambda y.y)((\lambda t.t)x))$.

$$
\begin{array}{lcll}
\Lambda & ::= & n \mid \lambda.\Lambda \mid \Lambda\,\Lambda & \text{terms with de Bruijn indices} \\
\mathsf{C} & ::= & \Lambda[\rho] \mid \overline{n} \mid \lfloor \Lambda \rfloor & \text{closures} \\
\rho & ::= & \varepsilon \mid \mathsf{C} : \rho & \text{environment stack} \\
S & ::= & \varepsilon \mid \Lambda[\rho] : S \mid \lambda : S \mid \lfloor \Lambda \rfloor : S & \text{continuation stack}
\end{array}
$$

$$
\begin{array}{lllllllll}
(1) & & & & & & T & \longrightarrow & (\quad T[\varepsilon], \qquad \varepsilon, \quad 0\ ) \\
(2) & (\quad (n+1)[\mathsf{C} : \rho], & S, & \ell\ ) & \longrightarrow & (\quad n[\rho], & S, & \ell\ ) \\
(3) & (\quad 0[\mathsf{C} : \rho], & S, & \ell\ ) & \longrightarrow & (\quad \mathsf{C}, & S, & \ell\ ) \\
(A) & (\quad n[\varepsilon], & S, & \ell\ ) & \longrightarrow & (\quad \lfloor n-(0-\ell) \rfloor, & S, & \ell\ ) \\
(4) & (\quad (MN)[\rho], & S, & \ell\ ) & \longrightarrow & (\quad M[\rho], & N[\rho] : S, & \ell\ ) \\
(5) & (\quad (\lambda.B)[\rho], & N[\rho'] : S, & \ell\ ) & \longrightarrow & (\quad B[N[\rho'] : \rho], & S, & \ell\ ) \\
(6) & (\quad (\lambda.B)[\rho], & S, & \ell\ ) & \longrightarrow & (\quad B[\overline{\ell+1} : \rho], & \lambda : S, & \ell+1\ ) \\
(7) & (\quad \overline{n}, & S, & \ell\ ) & \longrightarrow & (\quad \lfloor \ell-n \rfloor, & S, & \ell\ ) \\
(8) & (\quad \lfloor M \rfloor, & N[\rho] : S, & \ell\ ) & \longrightarrow & (\quad N[\rho], & \lfloor M \rfloor : S, & \ell\ ) \\
(9) & (\quad \lfloor B \rfloor, & \lambda : S, & \ell\ ) & \longrightarrow & (\quad \lfloor \lambda.B \rfloor, & S, & \ell-1\ ) \\
(10) & (\quad \lfloor N \rfloor, & \lfloor M \rfloor : S, & \ell\ ) & \longrightarrow & (\quad \lfloor MN \rfloor, & S, & \ell\ ) \\
(11) & (\quad \lfloor T \rfloor, & \varepsilon, & \ell\ ) & \longrightarrow & T & &
\end{array}
$$

Fig. 6. An improved version of KN that works with closures with open terms (García-Pérez *et al.*, 2013, p. 95).

$$
\begin{array}{lllllll}
(2) & (\quad n[\rho], & S, & \ell\ ) & \longrightarrow & (\quad n^{\text{th}}(\rho), & S, & \ell\ ) \quad \text{if } n < |\rho| \\
(3) & (\quad n[\rho], & S, & \ell\ ) & \longrightarrow & (\quad \lfloor n-(|\rho|-l) \rfloor, & S, & \ell\ ) \quad \text{if } n \geq |\rho|
\end{array}
$$

Fig. 7. Rules (2), (3) and (A) in Figure 6 can be replaced for uniformity by these two rules.

meaning the index is for a free variable. In this case the size of the environment is 0 and the de Bruijn index is calculated by adding the current nesting level to n.

For uniformity, rules (2), (3) and (A) in Figure 6 can be replaced by the two rules in Figure 7. The expression $n^{\text{th}}(\rho)$ denotes the nth closure in ρ with $0^{\text{th}}(\rho)$ the closure at the top. When $n \geq |\rho|$ the result index is calculated by subtracting from n the size of the environment without the formal parameters in it, that is, $|\rho| - \ell$. This number equals the number of operands in the environment, that is, the number of binding occurrences removed by reducing redexes in rule (5), which is the amount by which the index n must be decremented to be correct in the absence of an environment in the current scope.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$
\begin{array}{llll}
 & & \lambda.0(\lambda.M)N & \\
(1) & \to & (\ (\lambda.0(\lambda.M)N)[\varepsilon], & \varepsilon,\ 0\ ) \\
(6) & \to & (\ (0(\lambda.M)N)[\overline{1}], & \lambda,\ 1\ ) \\
(4) & \to & (\ (0(\lambda.M))[\overline{1}], & N[\overline{1}] : \lambda,\ 1\ ) \\
(4) & \to & (\ 0[\overline{1}], & (\lambda.M)[\overline{1}] : N[\overline{1}] : \lambda,\ 1\ ) \\
(3) & \to & (\ \overline{1}, & (\lambda.M)[\overline{1}] : N[\overline{1}] : \lambda,\ 1\ ) \\
(7) & \to & (\ \lfloor 1-1 \rfloor, & (\lambda.M)[\overline{1}] : N[\overline{1}] : \lambda,\ 1\ ) \\
(8) & \to & (\ (\lambda.M)[\overline{1}], & \lfloor 0 \rfloor : N[\overline{1}] : \lambda,\ 1\ ) \\
(6) & \to & (\ M[\overline{2},\overline{1}], & \lambda : \lfloor 0 \rfloor : N[\overline{1}] : \lambda,\ 2\ ) \\
 & & \ldots\ldots & \\
\ldots & \to & (\ \lfloor M' \rfloor, & \lambda : \lfloor 0 \rfloor : N[\overline{1}] : \lambda,\ 2\ ) \\
(9) & \to & (\ \lfloor \lambda.M' \rfloor, & \lfloor 0 \rfloor : N[\overline{1}] : \lambda,\ 1\ ) \\
(10) & \to & (\ \lfloor 0(\lambda.M') \rfloor, & N[\overline{1}] : \lambda,\ 1\ ) \\
(8) & \to & (\ N[\overline{1}], & \lfloor 0(\lambda.M') \rfloor : \lambda,\ 1\ ) \\
 & & \ldots\ldots &
\end{array}
$$

Fig. 8. Partial reduction trace of improved KN for the term $\lambda.0(\lambda.M)N$. The differences with Figure 4 are highlighted.

$$
\begin{array}{llll}
 & & \lambda.0((\lambda.0)((\lambda.0)0)) & \\
(1) & \to & (\ (\lambda.0((\lambda.0)((\lambda.0)0)))[\varepsilon], & \varepsilon,\ 0\ ) \\
(6) & \to & (\ (0((\lambda.0)((\lambda.0)0)))[\overline{1}], & \lambda,\ 1\ ) \\
(4) & \to & (\ 0[\overline{1}], & ((\lambda.0)((\lambda.0)0))[\overline{1}] : \lambda,\ 1\ ) \\
(2) & \to & (\ \overline{1}, & ((\lambda.0)((\lambda.0)0))[\overline{1}] : \lambda,\ 1\ ) \\
(7) & \to & (\ \lfloor 0 \rfloor, & ((\lambda.0)((\lambda.0)0))[\overline{1}] : \lambda,\ 1\ ) \\
(8) & \to & (\ ((\lambda.0)((\lambda.0)0))[\overline{1}], & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(4) & \to & (\ (\lambda.0)[\overline{1}], & ((\lambda.0)0)[\overline{1}] : \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(5) & \to & (\ 0[((\lambda.0)0)[\overline{1}] : \overline{1}], & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(2) & \to & (\ ((\lambda.0)0)[\overline{1}], & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(4) & \to & (\ (\lambda.0)[\overline{1}], & 0[\overline{1}] : \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(5) & \to & (\ 0[0[\overline{1}] : \overline{1}], & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(2) & \to & (\ 0[\overline{1}], & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(2) & \to & (\ \overline{1}, & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(7) & \to & (\ \lfloor 0 \rfloor, & \lfloor 0 \rfloor : \lambda,\ 1\ ) \\
(10) & \to & (\ \lfloor 00 \rfloor, & \lambda,\ 1\ ) \\
(9) & \to & (\ \lfloor \lambda.00 \rfloor, & \varepsilon,\ 0\ ) \\
(11) & \to & \lambda.00 &
\end{array}
$$

Fig. 9. Complete reduction trace of improved KN for the term $\lambda.0((\lambda.0)((\lambda.0)0))$ that encodes $\lambda x.x((\lambda y.y)((\lambda t.t)x))$.

Figure 8 shows the partial reduction trace of the term $\lambda.0(\lambda.M)N$ by improved KN. The differences with Figure 4 are highlighted. In particular, the current nesting level is decremented in the application of rule (9) and therefore result terms need not carry their nesting level. Figure 9 shows a complete reduction trace. Compare it to the reduction of the same term in Figure 5 by the original version of the machine.

5 The $\lambda\widetilde{\rho}$ calculus of closures for full reduction

The $\lambda\widetilde{\rho}$ calculus does not require a continuation stack and carries the current nesting level with the reduction relation. The calculus consists of closures and a one-step reduction relation on closures that is a reduction strategy (a partial function). As usual, multiple-step reduction is the iteration of one-step reduction. The relation/strategy reduces a closure in

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$
\begin{array}{lcll}
\mathsf{C} & ::= & \Lambda[\rho] \mid \overline{n} \mid \lfloor n \rfloor \mid \lambda\!\!\!\!\diagup.\mathsf{C} \mid \mathsf{C} \cdot \mathsf{C} & \text{closures} \\
\mathsf{E} & ::= & \lfloor n \rfloor \mid \lambda\!\!\!\!\diagup.\mathsf{E} \mid \mathsf{E} \cdot \mathsf{E} & \text{ephemerals} \\
\mathrm{WHNF}_{\mathsf{C}} & ::= & \lambda\!\!\!\!\diagup.\mathsf{C} \mid \lfloor n \rfloor \{\cdot \mathsf{C}\}^* & \text{closure weak head normal forms} \\
\mathrm{NF}_{\mathsf{C}} & ::= & \lambda\!\!\!\!\diagup.\mathrm{NF}_{\mathsf{C}} \mid \lfloor n \rfloor \{\cdot \mathrm{NF}_{\mathsf{C}}\}^* & \text{closure normal forms}
\end{array}
$$

$$
\frac{}{(MN)[\rho] \xrightarrow[\widetilde{no}]{\ell} M[\rho] \cdot N[\rho]}\ (\textsc{App}\widetilde{\rho}) \qquad \frac{}{(\lambda.B)[\rho] \xrightarrow[\widetilde{no}]{\ell} \lambda\!\!\!\!\diagup.B[\overline{\ell+1} : \rho]}\ (\textsc{Lam}\widetilde{\rho})
$$

$$
\frac{}{(\lambda\!\!\!\!\diagup.B[\overline{\ell+1} : \rho]) \cdot N[\rho'] \xrightarrow[\widetilde{no}]{\ell} B[N[\rho'] : \rho]}\ (\beta\widetilde{\rho}) \qquad \frac{n < |\rho|}{n[\rho] \xrightarrow[\widetilde{no}]{\ell} n^{\text{th}}(\rho)}\ (\textsc{Var}\widetilde{\rho})
$$

$$
\frac{n \geq |\rho|}{n[\rho] \xrightarrow[\widetilde{no}]{\ell} \lfloor n - (|\rho| - \ell) \rfloor}\ (\textsc{Fre}\widetilde{\rho}) \qquad \frac{}{\overline{n} \xrightarrow[\widetilde{no}]{\ell} \lfloor \ell - n \rfloor}\ (\textsc{Par}\widetilde{\rho})
$$

$$
\frac{\mathsf{M} \notin \mathrm{WHNF}_{\mathsf{C}} \quad \mathsf{M} \xrightarrow[\widetilde{no}]{\ell} \mathsf{M}'}{\mathsf{M} \cdot \mathsf{N} \xrightarrow[\widetilde{no}]{\ell} \mathsf{M}' \cdot \mathsf{N}}\ (\mu 1\widetilde{\rho}) \qquad \frac{\mathsf{M} \in \mathrm{WHNF}_{\mathsf{C}} \quad \mathsf{M} \notin \lambda\!\!\!\!\diagup.\mathsf{B} \quad \mathsf{M} \xrightarrow[\widetilde{no}]{\ell} \mathsf{M}'}{\mathsf{M} \cdot \mathsf{N} \xrightarrow[\widetilde{no}]{\ell} \mathsf{M}' \cdot \mathsf{N}}\ (\mu 2\widetilde{\rho})
$$

$$
\frac{\mathsf{M} \in \mathrm{NF}_{\mathsf{C}} \quad \mathsf{M} \notin \lambda\!\!\!\!\diagup.\mathsf{B} \quad \mathsf{N} \xrightarrow[\widetilde{no}]{\ell} \mathsf{N}'}{\mathsf{M} \cdot \mathsf{N} \xrightarrow[\widetilde{no}]{\ell} \mathsf{M} \cdot \mathsf{N}'}\ (\nu\widetilde{\rho}) \qquad \frac{\mathsf{B} \xrightarrow[\widetilde{no}]{\ell+1} \mathsf{B}'}{\lambda\!\!\!\!\diagup.\mathsf{B} \xrightarrow[\widetilde{no}]{\ell} \lambda\!\!\!\!\diagup.\mathsf{B}'}\ (\xi\widetilde{\rho})
$$

Fig. 10. Syntax and structural operational semantics of $\lambda\widetilde{\rho}$. The M, N and B letters stand for elements of the set of closures C. The current nesting level ℓ is a natural number.

leftmost fashion to its closure normal form if and only if such closure normal form exists. Hence, hereafter we refer to the reduction relation/strategy as $\lambda\widetilde{\rho}$*-normal-order*. The proof of lockstep simulation (Section 6) warrants this name.

Figure 10 shows the syntax of closures and of other closure subterms at the top, and the structural operational semantics of the reduction relation/strategy at the bottom. In the figure and hereafter, sans-serif letters M, N, B, etc. stand for elements of the set of closures. The current nesting level ℓ is a natural number.

We start with the syntax of closures. The first two cases are proper closures and de Bruijn levels. The remaining three constructors are result indices $\lfloor n \rfloor$, closure abstractions $\lambda\!\!\!\!\diagup.\mathsf{C}$ and closure applications $\mathsf{C} \cdot \mathsf{C}$. These constructors are required for full-reduction. They are called *ephemerals* in García-Pérez *et al.* (2013) because they are eliminated by the shortcut optimisation of the evaluator implementing the natural semantics of the relation/strategy. This is of no concern here, but we keep the name for consistency.

Considered in isolation, the set of ephemerals E is isomorphic to Λ, with the precedence and association convention of Λ's term application and abstraction extended to E's closure application and abstraction. We write $\mathsf{M} \cong M$ to indicate that an ephemeral M is isomorphic to the pure term M, where $\cong$ is the equivalence relation extension of this definition:

$$
\frac{}{\lfloor n \rfloor \cong n} \qquad \frac{\mathsf{M} \cong M}{\lambda\!\!\!\!\diagup.\mathsf{M} \cong \lambda.M} \qquad \frac{\mathsf{M} \cong M \qquad \mathsf{N} \cong N}{\mathsf{M} \cdot \mathsf{N} \cong MN}
$$

For example, $\lambda\!\!\!\!\diagup.\lfloor 0 \rfloor \cdot (\lambda\!\!\!\!\diagup.\lfloor 1 \rfloor) \cdot \lfloor 0 \rfloor \cong \lambda.0(\lambda.1)0$.

Let us explain the rationale of each ephemeral. The $\lambda\rho$ calculus of Curien has only proper closures and a reduction relation such that, as noted in Biernacka & Danvy (2007), one-step reduction (and structural operational semantics) cannot be expressed. The $\lambda\widehat{\rho}$

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

calculus of Biernacka & Danvy (2007) minimally extends $\lambda\rho$ with closure applications and an ephemeral expansion rule $(MN)[\rho] \longrightarrow M[\rho] \cdot N[\rho]$ that distributes the environment by constructing a closure application. Redexes are reduced by pushing the operand closure on the abstraction body's environment: $(\lambda.B)[\rho'] \cdot N[\rho] \longrightarrow B[N[\rho] : \rho']$. One-step reduction can be defined in $\lambda\widehat{\rho}$ but only for weak-reduction as it has no provision for (ξ), $(\mu 2)$ and (ν).

The $\lambda\widetilde{\rho}$ calculus adds the required constructors and rules for the full reduction of open terms. A result index $\lfloor n \rfloor$ is either the result of reducing a closure $n[\rho]$ with $n \geq |\rho|$, or of reducing a de Bruijn level $\overline{n}$ delivering $\lfloor \ell - \overline{n} \rfloor$. Closure abstractions $\lambda.\mathsf{C}$ are required to push the environment under lambda to reduce abstraction bodies.

The set of closure weak head normal forms $\mathrm{WHNF}_{\mathsf{C}}$ and the set of closure normal forms NF_{C} are obtained by adapting their respective homonymous definitions in the pure lambda calculus (Figure 2). Neutral closures have the form $\lfloor n \rfloor \, \{ \cdot \, \mathsf{C} \}^*$, and neutral closures in closure normal form have the form $\lfloor n \rfloor \, \{ \cdot \, \mathrm{NF}_{\mathsf{C}} \}^*$.

Let us move to the structural operational semantics. The rules of the first three rows are *notions of reduction* (Barendregt, 1984) for the closure constructors. The rules of the last two rows are compatibility rules obtained by adapting the homonymous rules of pure normal-order (Figure 2).

The structural operational semantics defines a labelled transition system (Keller, 1976) where each transition is labelled by the current nesting level ℓ of the scope at which the reduction step (the transition) is taking place. The level is only increased by rule $(\xi\widetilde{\rho})$ of compatibility under closure abstraction. It is kept constant by the other rules. Consequently, *the level increases in a derivation tree, not in a reduction sequence*, and is therefore an annotation, not a side-effect as is the case with labels in labelled transition systems of process algebra.[7]

We now explain the reduction rules in detail. In the first row we have the ephemeral expansion rules $(\textsc{App}\widetilde{\rho})$ and $(\textsc{Lam}\widetilde{\rho})$. The former expands application closures to closure applications as expected, and the latter expands abstraction closures to closure abstractions, pushing the de Bruijn level of the formal parameter (one plus the current nesting level) on the environment. In the second row, rule $(\beta\widetilde{\rho})$ contracts a redex $(\lambda.B[\overline{\ell + 1} : \rho]) \cdot N[\rho]$. The level that was pushed on the environment by an immediately preceding $(\textsc{Lam}\widetilde{\rho})$ is discarded and replaced by the operand.

Thanks to the separation of closure-abstraction expansion $(\textsc{Lam}\widetilde{\rho})$ from contraction $(\beta\widetilde{\rho})$ and from compatibility under lambda $(\xi\widetilde{\rho})$, the current nesting level remains constant at both sides of a reduction. As stated in the introduction, we call this property *balanced derivations*. It facilitates proof by structural induction and allows us to annotate the level in the transition.

Finally, rule $(\textsc{Var}\widetilde{\rho})$ is for in-bounds indices n, and rules $(\textsc{Fre}\widetilde{\rho})$ and $(\textsc{Par}\widetilde{\rho})$ are for *result index calculations* in the same vein as improved KN (Figure 6). In particular, rule $(\textsc{Fre}\widetilde{\rho})$ is for out-of-bounds indices n (free variables) and rule $(\textsc{Par}\widetilde{\rho})$ is for formal parameters $\overline{n}$.

Figure 11 shows a $\lambda\widetilde{\rho}$ reduction for the term $\lambda x.x((\lambda y.y)((\lambda t.t)x))$. Compare with its pure normal-order reduction in Example 3.1 (page 10).

[7] In García-Pérez *et al.* (2013, p. 91) judgements have the form $\langle \mathsf{M}, \ell \rangle \longrightarrow\!\!\!\!\!\!\sim_{no} \langle \mathsf{M}', \ell \rangle$ where the unchanging current nesting level is paired with the closure rather than much better written under the arrow relation.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

$$(\lambda.0((\lambda.0)((\lambda.0)0)))[\varepsilon] \xrightarrow{0}_{n\widetilde{o}} \lambda.(0((\lambda.0)((\lambda.0)0)))[\overline{1}] \qquad 0(\text{Lam}\widetilde{\rho})$$

$$\lambda.(0((\lambda.0)((\lambda.0)0)))[\overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.0[\overline{1}] \cdot ((\lambda.0)((\lambda.0)0))[\overline{1}] \qquad 0(\xi\widetilde{\rho})1(\text{App}\widetilde{\rho})$$

$$\lambda.0[\overline{1}] \cdot ((\lambda.0)((\lambda.0)0))[\overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\overline{1} \cdot ((\lambda.0)((\lambda.0)0))[\overline{1}] \qquad 0(\xi\widetilde{\rho})1(\mu 1\widetilde{\rho})1(\text{Var}\widetilde{\rho})$$

$$\lambda.\underset{\sim}{\overline{1}} \cdot ((\lambda.0)((\lambda.0)0))[\overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)((\lambda.0)0))[\overline{1}] \qquad 0(\xi\widetilde{\rho})1(\mu 1\widetilde{\rho})1(\text{Par}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)((\lambda.0)0))[\overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot (\lambda.0)[\overline{1}] \cdot ((\lambda.0)0)[\overline{1}] \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\text{App}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)[\overline{1}] \cdot ((\lambda.0)0)[\overline{1}]) \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot ((\lambda.0[\overline{2} : \overline{1}]) \cdot ((\lambda.0)0)[\overline{1}]) \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\mu 1\widetilde{\rho})1(\text{Lam}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot ((\lambda.0[\overline{2} : \overline{1}]) \cdot ((\lambda.0)0)[\overline{1}]) \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot 0[((\lambda.0)0)[\overline{1}] : \overline{1}] \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\beta\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot 0[((\lambda.0)0)[\overline{1}] : \overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)0)[\overline{1}] \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\text{Var}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)0)[\overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)[\overline{1}] \cdot 0[\overline{1}]) \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\text{App}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot ((\lambda.0)[\overline{1}] \cdot 0[\overline{1}]) \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot ((\lambda.0[\overline{2} : \overline{1}]) \cdot 0[\overline{1}]) \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\mu 1\widetilde{\rho})1(\text{Lam}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot ((\lambda.0[\overline{2} : \overline{1}]) \cdot 0[\overline{1}]) \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 1 \rfloor \cdot 0[0[\overline{1}] : \overline{1}] \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\beta\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot 0[0[\overline{1}] : \overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot 0[\overline{1}] \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\text{Var}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot 0[\overline{1}] \xrightarrow{0}_{n\widetilde{o}} \lambda.\lfloor 0 \rfloor \cdot \overline{1} \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\text{Var}\widetilde{\rho})$$

$$\lambda.\lfloor 0 \rfloor \cdot \underset{\sim}{\overline{1}} \xrightarrow{0}_{n\widetilde{o}} \lambda.\ \lfloor 0 \rfloor \cdot \lfloor 0 \rfloor \cong \lambda.00 \qquad 0(\xi\widetilde{\rho})1(\nu\widetilde{\rho})1(\text{Par}\widetilde{\rho})$$

Fig. 11. Reduction example in $\lambda\widetilde{\rho}$ for the term $\lambda.0((\lambda.0)((\lambda.0)0))$ encoding $\lambda x.x((\lambda y.y)((\lambda t.t)x))$. The expression being reduced at each step is underlined. On the right are shown the rules of the derivation tree of the step, with the nesting level of each rule's consequent and antecedent shown respectively before and after the rule's name.

5.1 On the correspondence between $\lambda\widetilde{\rho}$ and KN

As shown in Section 4, the original and the improved versions of KN realise the same reduction strategy for closed terms, i.e. reduce redexes in identical order. The improved version, and therefore the reduction strategy, allows for open terms. The $\lambda\widetilde{\rho}$ calculus also realises that very reduction strategy. In other words, it *corresponds* with the improved KN machine.

The proof of correspondence is by program transformation. Concretely, the derivation starts from an implementation of the structural operational semantics of $\lambda\widetilde{\rho}$'s reduction and arrives at an implementation of improved KN. Every step in the derivation is semantics-preserving: it modifies the shape of the implementation and affects administrative reduction, terms and structures, but the reduction strategy (reduction order of redexes) is maintained. For details, see García-Pérez *et al.* (2013). The following theorem formally states the correspondence.

Theorem 5.1 (Correspondence between $\lambda\widetilde{\rho}$ and KN)**.** *Let $T \in \Lambda$ be a term with de Bruijn indices. If T has no normal form, then improved KN reduction and $\lambda\widetilde{\rho}$ reduction diverge on T. If T has normal form N, then the following hold:*

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

(i) *The improved KN reduces T to N.*
(ii) $T[\varepsilon] \xrightarrow[\widetilde{no}]{0}{}^{*} \mathsf{N}$, *with* $\mathsf{N} \in \mathsf{E}$ *and* $\mathsf{N} \cong N$.
(iii) *Every redex reduced by KN's rule (5) is reduced by* $\lambda\widetilde{\rho}$*'s rule* ($\beta\widetilde{\rho}$) *in the same order.*

Proof By derivation by program transformation. □

For illustration, compare the $\lambda\widetilde{\rho}$ reduction in Figure 11 with the reduction trace by improved KN in Figure 9, as an example of the correspondence between the calculus and the improved machine.

6 Proof of lockstep simulation

This section proves that $\xrightarrow[\widetilde{no}]{\ell}$ simulates $\longrightarrow_{no}$ in lockstep. Hereafter we respectively refer to these strategies in text mode as $\widetilde{no}$ and *no*. Lemma 6.1 states a well-formedness invariant on the closures in a $\widetilde{no}$ reduction sequence. Definition 6.2 introduces substitution function σ, and Lemma 6.3 formalises the interaction between $\uparrow$ and σ. Lemmas 6.5 and 6.7 show that σ simulates capture-avoiding substitution in the pure lambda calculus. Definition 6.5 splits $\xrightarrow[\widetilde{no}]{\ell}$ into sub-strategies $\xrightarrow[\widetilde{\rho}]{\ell}$ and $\xrightarrow[\beta\widetilde{\rho}]{\ell}$. The former implements the administrative steps of ephemeral expansion, on-demand substitution and result index calculation (akin to σ). The latter reduces the $\beta\widetilde{\rho}$-redexes analogous of the β-redexes of the pure lambda calculus. Lemma 6.9 states that $\xrightarrow[\widetilde{no}]{\ell}{}^{*}$ is equivalent to the diagrammatic relational composition of $\xrightarrow[\widetilde{\rho}]{\ell}{}^{*}$ with $\xrightarrow[\beta\widetilde{\rho}]{\ell}$. Lemmas 6.10 and 6.11 state that $\lambda\widetilde{\rho}$ administrative reduction is akin to substitution σ for closures that flatten to a normal form. Lemma 6.12 states that administrative reduction locates the next $\beta\widetilde{\rho}$-redex. Theorem 6.13 states that normal-order (pure and $\lambda\widetilde{\rho}$'s) commute with substitution, and Corollary 6.14 states lockstep simulation: the one-to-one correspondence between the steps in a *no* reduction sequence and the non-administrative steps in a $\widetilde{no}$ reduction sequence.

Definition 6.1 (Well-formed environments and closures)**.** *An environment ρ is l-well-formed, written $wf(\rho, \ell)$, if ρ is an interleaving of levels and proper closures s.t.:*

(i) *the subsequence of ρ's levels is strictly decreasing and bounded by ℓ,*
(ii) *for any decomposition $\rho_1 : T[\rho'] : \rho_2$ of ρ, then ρ' is ℓ'-well-formed where ℓ' is less or equal than the leftmost level of ρ_2 (or 0 if no such level exists).*

Formally:

$$\frac{}{wf(\varepsilon, \ell)} \qquad \frac{wf(\rho, i) \quad i+1 \leq \ell}{wf(\overline{i+1} : \rho, \ell)} \qquad \frac{wf(\rho', i) \quad wf(\rho, j) \quad i \leq j \leq \ell}{wf(T[\rho'] : \rho, \ell)}$$

Well-formedness of environments is trivially extended to closures $wf(\mathsf{M}, \ell)$:

$$\frac{}{wf(\lfloor n \rfloor, \ell)} \qquad \frac{n \leq \ell}{wf(\overline{n}, \ell)} \qquad \frac{wf(\rho, \ell)}{wf(T[\rho], \ell)} \qquad \frac{wf(\mathsf{M}, \ell) \quad wf(\mathsf{N}, \ell)}{wf(\mathsf{M} \cdot \mathsf{N}, \ell)} \qquad \frac{wf(\mathsf{B}, \ell+1)}{wf(\lambda.\mathsf{B}, \ell)}$$

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Lemma 6.1 (Well formedness is invariant under reduction)**.** *Let* M *such that* $wf(\mathsf{M}, \ell)$ *and* $\mathsf{M} \xrightarrow[no]{\ell} \mathsf{M}'$. *Then* $wf(\mathsf{M}', \ell)$.

Proof By induction on the depth of the derivation tree of $\mathsf{M} \xrightarrow[no]{\ell} \mathsf{M}'$. □

A proof by straightforward induction is possible, thanks to the properties of $\lambda\widetilde{\rho}$ mentioned in Section 5. We invite readers to prove this lemma in calculi with lexical adjustments such as Munk (2008) and Danvy *et al.* (2013). See Sections 1.2 and 7.

Definition 6.2 (Substitution function)**.** *Function* σ *flattens a closure to a term by performing the delayed substitutions in it.*

$$\begin{aligned}
\sigma(\mathsf{C}, \mathbb{N}) &\to \Lambda \\
\sigma(n[\rho], \ell) &= \begin{cases} \sigma(n^{\text{th}}(\rho), \ell) & \text{if } n < |\rho| \\ n - (|\rho| - \ell) & \text{if } n \geq |\rho| \end{cases} \\
\sigma((\lambda.B)[\rho], \ell) &= \sigma(\lambda.B[\overline{\ell + 1} : \rho], \ell) \\
\sigma((MN)[\rho], \ell) &= \sigma(M[\rho] \cdot N[\rho], \ell) \\
\sigma(\lambda.\mathsf{B}, \ell) &= \lambda.\sigma(\mathsf{B}, \ell + 1) \\
\sigma(\overline{n}, \ell) &= \ell - n \\
\sigma(\mathsf{M} \cdot \mathsf{N}, \ell) &= (\sigma(\mathsf{M}, \ell))\,(\sigma(\mathsf{N}, \ell)) \\
\sigma(\lfloor n \rfloor, \ell) &= n
\end{aligned}$$

Function σ simulates capture-avoiding substitution in the pure lambda calculus, as proven by Lemma 6.7 below. The function improves the homonymous substitution function in Biernacka & Danvy (2007, p. 6:9). The current nesting level is carried as a parameter, and both index alignment and parameters-as-levels are taken into account. In the 2nd clause, σ increments the de Bruijn level of the formal parameter that is pushed on the environment, but does not increment the current nesting level. It is in the 4th clause, when going under closure abstraction, that the current nesting level is incremented without affecting the body's environment. The remaining clauses are unsurprising in the light of Figure 10.

Definition 6.3 (Structural induction principle)**.** *Given a predicate on closures* $\mathscr{P}$ *the (strong) structural induction principle for* $\mathscr{P}$ *is*

$$\begin{aligned}
&(\forall n.\mathscr{P}(\overline{n})) \\
&\wedge\, (\forall n.\mathscr{P}(\lfloor n \rfloor)) \\
&\wedge\, (\forall \mathsf{M}.\forall \mathsf{N}.\mathscr{P}(\mathsf{M}) \implies \mathscr{P}(\mathsf{N}) \implies \mathscr{P}(\mathsf{M} \cdot \mathsf{N})) \\
&\wedge\, (\forall \mathsf{B}.\mathscr{P}(\mathsf{B}) \implies \mathscr{P}(\lambda.\mathsf{B})) \\
&\wedge\, (\forall T.\forall \rho.(\forall \mathsf{M} \in \rho.\mathscr{P}(\mathsf{M})) \implies \mathscr{P}(T[\rho])) \\
&\implies \forall \mathsf{M}.\mathscr{P}(\mathsf{M})
\end{aligned}$$

The proofs of Lemmas 6.3–6.7, the proofs of Lemmas 6.10 and 6.11 and the proof of Theorem 6.13 rely on this induction principle.

A weaker induction principle can be obtained using the order on closures induced by their height, whether the closures are arbitrary or ℓ-well formed. A closure decreases its height after retrieving a binding or after ephemeral expansion.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Definition 6.4 (Height of a closure)**.** *The height of a closure is calculated by function h:*

$$\begin{aligned} h(\mathsf{C}) &\to \mathbb{N} \\ h(n[\rho]) &= \begin{cases} 1 + h(n^{\text{th}}(\rho)) & \text{if } n < |\rho| \\ 0 & \text{if } n \geq |\rho| \end{cases} \\ h((\lambda.B)[\rho]) &= 1 + h(\lambda.B[\overline{n} : \rho]) \\ h((MN)[\rho]) &= 1 + h(M[\rho] \cdot N[\rho]) \\ h(\lambda.\mathsf{B}) &= 1 + h(\mathsf{B}) \\ h(\overline{n}) &= 0 \\ h(\mathsf{M} \cdot \mathsf{N}) &= 1 + h(\mathsf{M}) + h(\mathsf{N}) \\ h(\lfloor n \rfloor) &= 0 \end{aligned}$$

The proof of Lemma 6.12 relies on the induction principle obtained from this height function.

Notation 6.2. *We write μ_m^n for an environment consisting uniquely of formal parameters $\overline{m+n} : \ldots : \overline{m+1}$, where $n, m \geq 0$.*

Lemma 6.3 (Shifting and σ commute)**.** *For any $\ell, m \geq 0$ and $T[\rho]$ a proper closure such that $wf(T[\rho], \ell)$, then*

$$\sigma(T[\rho], \ell + m) \equiv \uparrow_0^m (\sigma(T[\rho], \ell))$$

In words, flattening a closure with a level $\ell + m$ yields the same result as flattening the closure with level ℓ and then shifting it by m with a 0 cutoff. In other words, adding m to the nesting level ℓ represents an implicit shifting operation. To be able to use induction, Lemma 6.3 has to be generalised by considering how the environment ρ grows with new formal parameters $\overline{n}$ at its left, i.e. when $T \equiv \lambda \ldots \lambda.B$. The shifting function $\uparrow_0^m$ crosses the lambda symbols in T, and for that some appropriate ranges for the formal parameters at the left of ρ have to be provided.

Lemma 6.4 (Generalises Lemma 6.3)**.** *For any $\ell_0, \ell, m, k_0, k \geq 0$, if $wf(T[\rho], \ell_0)$ and $\ell_0 \leq \ell$ then*

$$\sigma(T[\mu_{\ell+m+k_0}^k : \rho], \ell + m + k_0 + k) \equiv \uparrow_{k_0+k}^m (\sigma(T[\mu_{\ell+k_0}^k : \rho], \ell + k_0 + k))$$

Proof By induction on $T[\mu_{\ell+k_0}^k : \rho]$ generalising the property over $\ell_0, \ell, m, k_0, k \geq 0$. Since $wf(T[\rho], \ell_0)$ holds, we only need to consider the following cases, which include the base case of the induction:

Case $T \equiv n$**:** We distinguish the sub-cases:

Case $n < k$**:** Then n points respectively to the formal parameters in each environment, i.e. $n^{\text{th}}(\mu_{\ell+m+k_0}^k : \rho) = \overline{\ell + m + k_0 + k - n}$ and $n^{\text{th}}(\mu_{\ell+k_0}^k : \rho) = \overline{\ell + k_0 + k - n}$. We need

$$\sigma(\overline{\ell + m + k_0 + k - n}, \ell + m + k_0 + k) \equiv \uparrow_{k_0+k}^m (\sigma(\overline{\ell + k_0 + k - n}, \ell + k_0 + k))$$

which simplifies to $n \equiv \uparrow_{k_0+k}^m n$. The latter trivially holds since $n < k_0 + k$.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Case $k \le n < k + |\rho|$**:** Then: $n^{\text{th}}(\mu^k_{\ell+m+k_0} : \rho) = n^{\text{th}}(\mu^k_{\ell+k_0} : \rho) = \mathsf{M}$. We distinguish the sub-cases:

Case $\mathsf{M} \equiv \overline{p}$**:** From the assumption $wf(T[\rho], \ell_0)$, we know $p \le \ell_0 \le \ell$. We need

$$\sigma(\overline{p}, \ell + m + k_0 + k) \;\equiv\; \uparrow^m_{k_0+k} (\sigma(\overline{p}, \ell + k_0 + k))$$

Let $q = \ell - p \ge 0$. The goal simplifies to $q + m + k_0 + k \;\equiv\; \uparrow^m_{k_0+k} (q + k_0 + k)$, which holds by definition of $\uparrow^m_{k_0+k}$ since $q + k_0 + k \ge k_0 + k$.

Case $\mathsf{M} \equiv T'[\rho']$**:** We need $\sigma(T'[\rho'], \ell + m + k_0 + k) \;\equiv\; \uparrow^m_{k_0+k} (\sigma(T'[\rho'], \ell + k_0 + k))$. Let $k' = k_0 + k$, then the goal is $\sigma(T'[\rho'], \ell + k' + m) \;\equiv\; \uparrow^m_{k'} (\sigma(T'[\rho'], \ell + k'))$. From the assumption $wf((T[\rho]), \ell_0)$, we know that $wf(T'[\rho'], \ell')$ and $\ell' \le \ell_0 \le \ell$. The goal holds by the induction hypothesis replacing ℓ_0 by ℓ' and k_0 by k', and letting $k = 0$.

Case $n \ge k + |\rho|$**:** We need

$$n - (|\mu^k_{\ell+m+k_0} : \rho| - (\ell + m + k_0 + k)) \;\equiv\; \uparrow^m_{k_0+k} (n - (|\mu^k_{\ell+k_0} : \rho| - (\ell + k_0 + k)))$$

Let $q = n - (k + |\rho|) \ge 0$. The goal simplifies to

$$q + \ell + m + k_0 + k \;\equiv\; \uparrow^m_{k_0+k} (q + \ell + k_0 + k)$$

which holds by definition of $\uparrow^m_{k_0+k}$ since $q + \ell + k_0 + k \ge k_0 + k$.

Case $T \equiv \lambda.B$**:** Both σ and $\uparrow^m_{k_0+k}$ cross the lambda, and we need

$$\lambda.\sigma(B[\mu^{k+1}_{\ell+m+k_0} : \rho], \ell + m + k_0 + k + 1) \;\equiv\; \lambda.\, \uparrow^m_{k_0+k+1} (\sigma(B[\mu^{k+1}_{\ell+k_0} : \rho], \ell + k_0 + k + 1))$$

which holds by the induction hypothesis.

Case $T \equiv MN$**:** By the induction hypothesis.

□

We illustrate the intuition of Lemma 6.4 with an example. Consider the term $T \equiv (\lambda.\lambda.\lambda.2)0$ and the $\widetilde{no}$ reduction sequence

$$((\lambda.\lambda.\lambda.2)0)[\varepsilon] \;\xrightarrow{0}{}^{*}_{\widetilde{no}}\; (\lambda.(\lambda.\lambda.2)[\overline{1}]) \cdot 0[\varepsilon] \;\xrightarrow{0}_{\widetilde{no}}\; (\lambda.\lambda.2)[0[\varepsilon]]$$

Let us check Lemma 6.3 over the obtained 0-well-formed proper closure $(\lambda.\lambda.2)[0[\varepsilon]]$ and then calculate:

$$\begin{array}{ll} & \sigma((\lambda.\lambda.2)[0[\varepsilon]], m) \;\equiv\; \uparrow^m_0 (\sigma((\lambda.\lambda.2)[0[\varepsilon]], 0)) \\ \iff & \quad \{\sigma \text{ and } \uparrow \text{ cross outer lambda}\} \\ & \lambda.\, \sigma((\lambda.2)[\overline{m+1} : 0[\varepsilon]], m + 1) \;\equiv\; \lambda.\, \uparrow^m_1 (\sigma((\lambda.2)[\overline{1} : 0[\varepsilon]], 1)) \end{array}$$

This calculation explains whence the coefficient that increases the current nesting level when crossing lambdas (let us call it $k' := 1$) comes from, and whence the padding $\overline{m+1}$ and $\overline{1}$ that keep index alignment respectively at each side of the goal come from. By stripping off the outermost lambda the goal now has the shape

$$\sigma((\lambda.2)[\mu^{k'}_{\ell+m} : 0[\varepsilon]], \ell + k' + m) \;\equiv\; \uparrow^m_{k'} (\sigma((\lambda.2)[\mu^{k'}_{\ell} : 0[\varepsilon]], \ell + k'))$$

which is not what Lemma 6.4 says but close. Let us show what is missing. Let us check another instance of the statement of the lemma over the 1-well-formed proper closure $(\lambda.2)[\overline{1} : 0[\varepsilon]]$ that we obtain by dropping the outermost lambda on the right-hand side of the goal before. We set coefficient ℓ to the current level 1, reset coefficient k' to zero and

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

calculate:

$$\begin{array}{ll} & \sigma((\lambda.2)[\overline{1} : 0[\varepsilon]], m+1) \equiv \uparrow_0^m (\sigma((\lambda.2)[\overline{1} : 0[\varepsilon]], 1)) \\ \iff & \quad \{\sigma \text{ and } \uparrow \text{ cross outer lambda (now } k' := 1)\} \\ & \lambda.\, \sigma(2[\overline{m+2} : \overline{1} : 0[\varepsilon]], m+2) \equiv \lambda.\, \uparrow_1^m (\sigma(2[\overline{2} : \overline{1} : 0[\varepsilon]], 2)) \\ \iff & \quad \{\text{environment lookup}\} \\ & \lambda.\, \sigma(0[\varepsilon], m+2) \equiv \lambda.\, \uparrow_1^m (\sigma(0[\varepsilon], 2)) \end{array}$$

The binding $0[\varepsilon]$ is retrieved from the environment. However, the current nesting level is $\ell + k' = 2$, instead of 0 at which the binding $0[\varepsilon]$ was pushed on the environment. Two refinements have to be made in order to adapt the induction hypothesis to the current goal.

The first refinement is to decouple ℓ_0 from the coefficient ℓ such that $wf(T[\rho], \ell_0)$, and to require that $\ell_0 \leq \ell$. Happily, $wf(T[\rho], \ell_0)$ guarantees that any binding $N[\rho']$ contained in ρ is ℓ'-well-formed for some $\ell' \leq \ell_0$, and before applying the induction hypothesis a new ℓ_0 is set to ℓ', i.e. $\ell_0 := \ell'$.

The second refinement involves the padding μ in the environments and the coefficient k'. We can only assume that the retrieved binding has an empty padding or we would lose generality. But, an empty padding does not match the $\mu^{k'}_{\ell+m}$ nor the $\mu^{k'}_{\ell}$ that we arrived at in the shape of the goal in the previous step. (By definition, the size of both paddings is k', and coefficient k' is incremented any time a lambda is crossed.) The solution is to split the k' into two (i.e. $k' = k_0 + k$) such that the excess k over k_0 corresponds with the size of the paddings in the goal. When a lambda is crossed, k is incremented. Before applying the induction hypothesis over a closure retrieved from the environment, the increment k is transferred to the accumulated increment k_0 (i.e. $k_0 := k_0 + k$) and the new k is reset to zero. In our example the coefficients are $\ell_0 = 1$, $\ell = 1$, $k_0 = 0$ and $k = 1$, and $wf(0[\varepsilon], \ell')$ with $\ell' = 0$. Before applying the induction hypothesis we let $\ell_0 := \ell'$, $\ell := 0$, $k_0 := 1$ and $k := 0$. Stripping off the outermost lambda, the goal has the shape

$$\sigma(0[\mu^k_{\ell+m+k_0} : \varepsilon], \ell + m + k_0 + k) \equiv \uparrow^m_{k_0+k} (\sigma(0[\mu^k_{\ell+k_0} : \varepsilon], \ell + k + k_0))$$

which now matches Lemma 6.4.

Now we prove that σ performs capture avoiding substitution in the pure lambda calculus. The proof also requires generalised lemmata, but a unique coefficient for the increment on the nesting level is enough (we call it k in Lemma 6.5, and m in Lemma 6.7).

Lemma 6.5 (Substitution index exceeded by level)**.** *For any $\ell_0, \ell, m \geq 0$ such that $\ell_0 \leq \ell$, and $T[\rho]$ a proper closure such that $wf(T[\rho], \ell_0)$, then $[S/m](\sigma(T[\rho], \ell + m + 1)) \equiv \sigma(T[\rho], \ell + m)$.*

In words, whenever the nesting level is strictly bigger than the substitution index m, the substitution function $[S/m]\_$ discards the subject of the substitution S and shifts the indices in the flattened closure $\sigma(T[\rho], \ell + m + 1)$ such that the result is equivalent to the flattened closure $\sigma(T[\rho], \ell + m)$.

Lemma 6.6 (Generalises Lemma 6.5)**.** *For any $\ell_0, \ell, m, k \geq 0$, if $wf(T[\rho], \ell_0)$ and $\ell_0 \leq \ell$, then*

$$[S/m + k](\sigma(T[\mu^k_{\ell+m+1} : \rho], \ell + m + 1 + k)) \equiv \sigma(T[\mu^k_{\ell+m} : \rho], \ell + m + k)$$

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Proof By induction on $T[\mu^k_{\ell+m} : \rho]$ generalising the property over $\ell_0, \ell, m, k \geq 0$. Since $T[\rho]$ is an ℓ_0-well-formed proper closure, we only need to analyse the following cases, which cursorily include the base case of the induction:

Case $T \equiv n$**:** We distinguish the sub-cases:

Case $n < k$**:** Then n points respectively to the formal parameters in each environment, i.e. $n^{\text{th}}(\mu^k_{\ell+m+1} : \rho) = \overline{\ell + m + 1 + k - n}$, and $n^{\text{th}}(\mu^k_{\ell+m} : \rho) = \overline{\ell + m + k - n}$. We need

$$[S/m + k](\sigma(\overline{\ell + m + 1 + k - n}, \ell + m + 1 + k)) \equiv \sigma(\overline{\ell + m + k - n}, \ell + m + k)$$

which simplifies to $[S/m + k]n \equiv n$. The latter holds by the definition of $[\_/\_]\_$ since $k > n$.

Case $k \leq n < k + |\rho|$**:** Then: $n^{\text{th}}(\mu^k_{\ell+m+1} : \rho) = n^{\text{th}}(\mu^k_{\ell+m} : \rho) = \mathsf{M}$. We distinguish the sub-cases:

Case $\mathsf{M} \equiv \overline{p}$: From the assumption $wf(T[\rho], \ell_0)$ we know that $p \leq \ell_0 \leq \ell$. We need

$$[S/m + k](\sigma(\overline{p}, \ell + m + 1 + k)) \equiv \sigma(\overline{p}, \ell + m + k)$$

which simplifies to $[S/m + k](q + 1) \equiv q$, where $q \geq m + k$. The latter holds by the definition of $[\_/\_]\_$.

Case $\mathsf{M} \equiv T'[\rho']$**:** We need $[S/m + k](\sigma(T'[\rho'], \ell + m + 1 + k)) \equiv \sigma(T'[\rho'], \ell + m + k)$ where $wf(T'[\rho+'], \ell')$ and $\ell' \leq \ell_0 \leq \ell$. The goal holds by the induction hypothesis replacing m by $m + k$, and letting $k = 0$.

Case $n \geq k + |\rho|$**:** We need

$$[S/m + k](n - |\mu^k_{\ell+m+1} : \rho| + \ell + m + 1 + k) \equiv n - |\mu^k_{\ell+m} : \rho| + \ell + m + k$$

which simplifies to $[S/m + k](q + 1) \equiv q$, where $q = n - |\rho| + \ell + m$. The latter holds by the definition of $[\_/\_]\_$, since $n - |\rho| \geq k$.

Case $T \equiv \lambda.B$**:** Both σ and $[\_/\_]\_$ cross the lambda, and we need

$$\lambda.[S/m + k + 1](\sigma(B[\mu^{k+1}_{\ell+m+1} : \rho], \ell + m + 1 + k + 1)) \equiv \lambda.\sigma(B[\mu^{k+1}_{\ell+m} : \rho], \ell + m + k + 1)$$

which holds by the induction hypothesis.

Case $T \equiv MN$**:** By the induction hypothesis. □

Lemma 6.7 (Interaction between σ and $[\_/\_]\_$)**.** *For any $\ell \geq 0$ and $B[N[\rho'] : \rho]$ a proper closure such that $wf(B[N[\rho'] : \rho], \ell)$, then*

$$\sigma(B[N[\rho'] : \rho], \ell) \equiv [\sigma(N[\rho'], \ell)/0](\sigma(B[\overline{\ell + 1} : \rho], \ell + 1))$$

In words, for any nesting level ℓ, flattening a closure $B[N[\rho'] : \rho]$ (i.e. the body of a closure abstraction $\lambda.B[\overline{\ell + 1} : \rho]$ where a closure subject $N[\rho']$ is pushed on the position pointed by index 0) yields the same term than the substitution of flattened subject $\sigma(N[\rho'], \ell)$ for 0 in flattened body $\sigma(B[\overline{\ell + 1} : \rho], \ell + 1)$.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Lemma 6.8 (Generalises Lemma 6.7). *For any* $\ell_0, \ell, m \geq 0$*, if* $wf(B[N[\rho'] : \rho], \ell_0)$ *and* $\ell_0 \leq \ell$*, then*

$$\sigma(B[\mu_\ell^m : N[\rho'] : \rho], \ell + m) \equiv [\sigma(N[\rho'], \ell)/m](\sigma(B[\mu_{\ell+1}^m : \overline{\ell+1} : \rho], \ell + m + 1))$$

Proof By induction on $B[\mu_{\ell+1}^m : \overline{\ell+1} : \rho]$ generalising the property over $\ell_0, \ell, m \geq 0$. Since $wf(B[N[\rho'] : \rho], \ell_0)$ we only need to analyse the following cases, which cursorily include the base case of the induction:

Case $B \equiv n$**:** We distinguish the sub-cases:

Case $n = m$**:** Then n points respectively to the closure subject $N[\rho']$ and to the formal parameter $\overline{\ell+1}$, i.e. $n^{\text{th}}(\mu_\ell^m : N[\rho'] : \rho) = N[\rho']$, and $n^{\text{th}}(\mu_{\ell+1}^m : \overline{\ell+1} : \rho) = \overline{\ell+1}$. By definition of $[\_/\_]\_$ we need $\sigma(N[\rho'], \ell + m) \equiv \uparrow_0^m \sigma(N[\rho'], \ell)$, which holds by Lemma 6.3 where $k_0, k, \ell_0 = 0$.

Case $n < m$**:** Then n points respectively to the formal parameters in each environment, i.e. $n^{\text{th}}(\mu_\ell^m : N[\rho'] : \rho) = \overline{\ell + (m-n)}$, and $n^{\text{th}}(\mu_{\ell+1}^m : \overline{\ell+1} : \rho) = \overline{\ell + (m-n) + 1}$. We need $\sigma(\overline{\ell+(m-n)}, \ell+m) \equiv [\sigma(N[\rho'], \ell)/m](\sigma(\overline{\ell + (m-n) + 1}, \ell + m + 1))$, which simplifies to $n \equiv [\sigma(N[\rho'], \ell)/m]n$. The lemma holds by definition of $[\_/\_]\_$ since $m > n$.

Case $m < n \leq m + |\rho|$**:** Then: $n^{\text{th}}(\mu_\ell^m : N[\rho'] : \rho) = n^{\text{th}}(\mu_{\ell+1}^m : \overline{\ell+1} : \rho) = \mathsf{M}$. We distinguish the sub-cases:

Case $\mathsf{M} \equiv \overline{p}$**:** From the assumption $wf(B[N[\rho'] : \rho], \ell_0)$, we know that $p \leq \ell_0 \leq \ell$. We need $\sigma(\overline{p}, \ell + m) \equiv [\sigma(N[\rho'], \ell)/m](\sigma(\overline{p}, \ell + m + 1))$, which simplifies to $q \equiv [\sigma(N[\rho'], \ell)/m](q + 1)$, where $q \geq m$. The lemma holds because $[\_/\_]\_$ decrements by one every index which is greater than m.

Case $\mathsf{M} \equiv T[\rho'']$**:** We need

$$\sigma(T[\rho''], \ell + m) \equiv [\sigma(N[\rho'], \ell)/m](\sigma(T[\rho''], \ell + m + 1))$$

where $wf(T[\rho''], \ell')$. The lemma holds by Lemma 6.5 since $\ell' \leq \ell$.

Case $n > m + |\rho|$**:** We need

$$n - ((m + 1 + |\rho|) - (\ell + m)) \equiv [\sigma(N[\rho'], \ell)/m](n - ((m + 1 + |\rho|) - (\ell + m + 1)))$$

which simplifies to $q \equiv [\sigma(N[\rho'], \ell)/m](q + 1)$ where $q = n - 1 - |\rho| + \ell$. The latter holds by the definition of $[\_/\_]\_$ because $q + 1 > m$.

Case $B \equiv \lambda.M$**:** Both σ and $[\_/\_]\_$ cross the lambda, and we need

$$\lambda.\sigma(M[\mu_\ell^{m+1} : N[\rho'] : \rho], \ell + m + 1) \equiv \lambda.[\sigma(N[\rho'], \ell)/m + 1](\sigma(M[\mu_{\ell+1}^{m+1} : \rho], \ell + 1 + m + 1))$$

which holds by the induction hypothesis.

Case $B \equiv MN$**:** By the induction hypothesis. □

The axioms in Figure 10 except $(\beta\widetilde{\rho})$ are the notions of reduction for ephemeral expansion, lookup and result index calculation, which perform computations similar to function σ. We consider the steps whose derivations end in those axioms as administrative, and split the $\widetilde{no}$ strategy into the following strategies.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Definition 6.5 ($\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ and $\xrightarrow{\ell}_{\widetilde{\rho}}$)**.** *Reduction* $\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ *is the strategy defined by rules* $(\beta\widetilde{\rho})$*,* $(\mu 1\widetilde{\rho})$*,* $(\mu 2\widetilde{\rho})$*,* $(\nu\widetilde{\rho})$ *and* $(\xi\widetilde{\rho})$ *in Figure 10. Reduction* $\xrightarrow{\ell}_{\widetilde{\rho}}$ *is the strategy defined by rules* $(\textsc{App}\widetilde{\rho})$*,* $(\textsc{Lam}\widetilde{\rho})$*,* $(\textsc{Var}\widetilde{\rho})$*,* $(\textsc{Fre}\widetilde{\rho})$*,* $(\textsc{Par}\widetilde{\rho})$*,* $(\mu 1\widetilde{\rho})$*,* $(\mu 2\widetilde{\rho})$*,* $(\nu\widetilde{\rho})$ *and* $(\xi\widetilde{\rho})$ *in Figure 10.*

Lemma 6.9 ($\xrightarrow{\ell}_{\widetilde{no}}$ splits into $\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ and $\xrightarrow{\ell}_{\widetilde{\rho}}$)**.** *Let* $\mathsf{M}_0, \ldots, \mathsf{M}_n$ *be a* $\widetilde{no}$ *reduction sequence at level* ℓ*, and let* $\mathsf{M} \xrightarrow{\ell}_{\beta\widetilde{\rho}} \mathsf{M}_0$ *and* $\mathsf{M}' \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{M}_0$*. Then both* $\mathsf{M}, \mathsf{M}_0, \ldots, \mathsf{M}_n$ *and* $\mathsf{M}', \mathsf{M}_0, \ldots, \mathsf{M}_n$ *are* $\widetilde{no}$ *reduction sequences at level* ℓ*.*

Proof The semantics of $\xrightarrow{\ell}_{\widetilde{no}}$ in Figure 10 is syntax-directed and deterministic. Since it defines a one-step strategy, a derivation's upward-branching-tree consists of a single branch with a unique axiom in its summit. Partitioning the axioms in disjoint sets, and considering each set together with all the compatibility rules, results in sub-strategies of $\widetilde{no}$. □

We distinguish the shape of a closure either at the beginning of a reduction sequence or after a $(\beta\widetilde{\rho})$ step, which we call a *start closure*, and immediately before a $(\beta\widetilde{\rho})$ step, which we call an *expanded closure*.

Definition 6.6 (Start closure and expanded closure)**.** *A start closure has the following shape:*

$$\begin{array}{rcl}
\mathrm{ST}_{\mathsf{C}} & ::= & \lfloor n \rfloor \\
& \mid & \Lambda[\rho]\,\{\cdot\,\Lambda[\rho]\}^* \\
& \mid & \overline{n}\,\{\cdot\,\Lambda[\rho]\}^* \\
& \mid & \lfloor n \rfloor\,\{\cdot\,\mathrm{NF}_{\mathsf{C}}\}^* \cdot \mathrm{ST}_{\mathsf{C}}\,\{\cdot\,\Lambda[\rho]\}^* \\
& \mid & \lambda.\mathrm{ST}_{\mathsf{C}}
\end{array}$$

An expanded closure at level ℓ *has the following shape:*

$$\begin{array}{rcl}
\mathrm{EX}^{\ell}_{\mathsf{C}} & ::= & (\lambda.\Lambda[\overline{\ell+1} : \rho])\{\cdot\,\Lambda[\rho]\}^+ \\
& \mid & \lfloor n \rfloor\,\{\cdot\,\mathrm{NF}_{\mathsf{C}}\}^* \cdot \mathrm{EX}^{\ell}_{\mathsf{C}}\,\{\cdot\,\Lambda[\rho]\}^* \\
& \mid & \lambda.\mathrm{EX}^{\ell+1}_{\mathsf{C}}
\end{array}$$

We have defined start closures such that they include the closure normal forms. This is a technical requirement that eases the proofs of Lemmas 6.10, 6.11 and 6.12 below. The expanded closures are indexed by the de Bruijn level at which they occur, such that if the $\beta\widetilde{\rho}$ redex signalled by the expanded closure occurs at level ℓ, then the formal parameter in its body is $\overline{\ell+1}$.

The set of ℓ-well-formed closures irreducible by $\xrightarrow{\ell}_{\widetilde{\rho}}$ is made up of closure normal forms and expanded closures at level ℓ.

Now we show that administrative reduction is akin to substitution for closures that flatten into a closure normal form.

Lemma 6.10 (Administrative reduction to weak head normal form)**.** *Let* $\ell \geq 0$ *and* $\mathsf{S} \in \mathrm{ST}_{\mathsf{C}}$ *such that* $\mathsf{S} \notin \mathrm{WHNF}_{\mathsf{C}}$*, and* $T \in \mathrm{WHNF}$*. If* $wf(\mathsf{S}, \ell)$ *and* $\sigma(\mathsf{S}, \ell) = T$*, then* $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}' \in \mathrm{ST}_{\mathsf{C}}$ *and either* $\mathsf{S}' \notin \mathrm{NF}_{\mathsf{C}}$*, or otherwise* $\mathsf{S}' \in \mathrm{NF}_{\mathsf{C}}$ *and* $\mathsf{S}' \cong T$*.*

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Proof The proof goes by induction on S. We distinguish the following cases:

Case $\mathsf{S} \equiv \lfloor n \rfloor$**:** Then $\overline{n} \xrightarrow{\ell}_{\widetilde{\rho}} \lfloor \ell - n \rfloor$ and the lemma holds.
Case $\mathsf{S} \equiv \overline{n}$**:** Then $\overline{n} \xrightarrow{\ell}_{\widetilde{\rho}} \lfloor \ell - n \rfloor$ and the lemma holds.
Case $\mathsf{S} \equiv n[\rho]$**:** If $n \geq |\rho|$, then $n[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \lfloor n - (|\rho| - \ell) \rfloor$ and the lemma holds. If $n < |\rho|$, since S is well formed, then $n^{\text{th}}(\rho)$ can be either $\overline{m}$ or $M[\rho']$ and the lemma holds.
Case $\mathsf{S} \equiv (MN)[\rho]$**:** Then $(MN)[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} M[\rho] \cdot N[\rho]$ and the lemma holds.
Case $\mathsf{S} \equiv (\lambda.B)[\rho]$**:** Then $(\lambda.B)[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \lambda.B[\overline{\ell + 1} : \rho]$ and the lemma holds.
Case $\mathsf{S} \equiv \mathsf{S}_1 \cdot N[\rho]$ **where** $\mathsf{S}_1 \in \mathrm{ST}_{\mathsf{C}}$ **and** $\mathsf{S}_1 \notin \mathrm{WHNF}_{\mathsf{C}}$**:** We have that $T = M'\,N'$ such that $M' \equiv \sigma(\mathsf{S}_1, \ell)$ and $N' \equiv \sigma(N[\rho], \ell)$, and where M' is in WHNF. We know that S_1 cannot be an abstraction closure, or otherwise M would not be in WHNF. Since S_1 is a start closure in $\mathrm{WHNF}_{\mathsf{C}}$ that is not an abstraction closure, then S_1 can only be an index closure, an application closure, a nested application of proper closures or a nested application of proper closures with a formal parameter as the leftmost operator. Since S_1 is ℓ-well-formed, neither of these reduces by $\xrightarrow{\ell}_{\widetilde{\rho}}$ to a closure abstraction, and thus we know that S'_1 is not a closure abstraction. By the induction hypothesis, $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}}{}^{*} \mathsf{S}'_1 \in \mathrm{ST}_{\mathsf{C}}$. By rule $(\mu 1\widetilde{\rho})$ then $\mathsf{S}_1 \cdot N[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}'_1 \cdot N[\rho] \equiv \mathsf{S}' \in \mathrm{ST}_{\mathsf{C}}$ and we are done since S' is not in NF_{C}. □

Lemma 6.11 (Administrative reduction to normal form)**.** *Let* $\ell \geq 0$ *and* $\mathsf{S} \in \mathrm{NF}_{\mathsf{C}}$ *such that* $\mathsf{S} \in \mathrm{WHNF}_{\mathsf{C}}$ *and* $\mathsf{S} \notin \mathrm{NF}_{\mathsf{C}}$*, and* $T \in \mathrm{NF}$*. If wf*(S, ℓ) *and* $\sigma(\mathsf{S}, \ell) = T$*, then* $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}' \in \mathrm{ST}_{\mathsf{C}}$ *and either* $\mathsf{S}' \in \mathrm{WHNF}_{\mathsf{C}}$ *and* $\mathsf{S}' \notin \mathrm{NF}_{\mathsf{C}}$*, or otherwise* $\mathsf{S}' \in \mathrm{NF}_{\mathsf{C}}$ *and* $\mathsf{S}' \cong T$*.*

Proof The proof goes by induction on S, generalising the property over ℓ. The cases where $\mathsf{S} \equiv \lfloor n \rfloor$, $\mathsf{S} \equiv \overline{n}\,\{\cdot N_i[\rho_i]\}^*$, and $\mathsf{S} \equiv M[\rho]\,\{\cdot N_i[\rho_i]\}^*$ are impossible because we know that S is in $\mathrm{WHNF}_{\mathsf{C}}$ and not in NF_{C}. We consider the following cases:

Case $\mathsf{S} \equiv \mathsf{S}_1 \cdot N[\rho]$ **where** $\mathsf{S}_1 \in \mathrm{ST}_{\mathsf{C}}$**,** $\mathsf{S}_1 \in \mathrm{WHNF}_{\mathsf{C}}$ **and** $\mathsf{S}_1 \notin \mathrm{NF}_{\mathsf{C}}$**:** By the induction hypothesis we know that $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}'_1 \in \mathrm{ST}_{\mathsf{C}}$. We also know that S_1 is not a closure abstraction because otherwise S would not be in $\mathrm{WHNF}_{\mathsf{C}}$. Therefore, by rule $(\mu 2\widetilde{\rho})$, $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}'_1 \cdot N[\rho]$, which is a start closure in $\mathrm{WHNF}_{\mathsf{C}}$ and not in NF_{C}, and the lemma holds.
Case $\mathsf{S} \equiv \mathsf{S}_1 \cdot \mathsf{S}_2$ **where** $\mathsf{S}_1 \equiv \lfloor n \rfloor\,\{\cdot \mathsf{N}_i\}^*$ **with** $\mathsf{N}_i \in \mathrm{NF}_{\mathsf{C}}$**, and** $\mathsf{S}_2 \in \mathrm{ST}_{\mathsf{C}}$ **not in** NF_{C}**:** We have that $T \equiv MN \in \mathrm{NF}$ where $\sigma(\mathsf{S}_1, \ell) = M$ and $\sigma(\mathsf{S}_2, \ell) = N$. We distinguish the following subcases:

> **Subcase** $\mathsf{S}_2 \notin \mathrm{WHNF}_{\mathsf{C}}$**:** Then by Lemma 6.10 we know that $\mathsf{S}_2 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}'_2 \in \mathrm{ST}_{\mathsf{C}}$. By rule $(\nu\widetilde{\rho})$, we have that $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1 \cdot \mathsf{S}'_2 \equiv \mathsf{S}'$. If S'_2 is not in NF_{C}, then S' is not in NF_{C} either and the lemma holds. If S'_2 is in NF_{C}, then S' is in NF_{C} and we know by Lemma 6.10 that $\mathsf{S}'_2 \cong N$, and thus $\mathsf{S}' \equiv \mathsf{S}_1 \cdot \mathsf{S}'_2 \cong MN \equiv T$ and the lemma holds.
>
> **Subcase** $\mathsf{S}_2 \in \mathrm{WHNF}_{\mathsf{C}}$**:** Then by the induction hypothesis we have that $\mathsf{S}_2 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}'_2 \in \mathrm{ST}_{\mathsf{C}}$. By rule $(\nu\widetilde{\rho})$, we have that $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1 \cdot \mathsf{S}'_2 \equiv \mathsf{S}'$. If S'_2 is not in NF_{C}, then S' is not in NF_{C} either and the lemma holds. If S'_2 is in NF_{C}, then S' is in NF_{C} and we

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

know by the induction hypothesis that $\mathsf{S}_2' \cong N$, and thus $\mathsf{S}' \equiv \mathsf{S}_1 \cdot \mathsf{S}_2' \cong MN \equiv T$ and the lemma holds.

Case $\mathsf{S} \equiv \lambda.\mathsf{S}_1$**:** We have that $\sigma(\mathsf{S}, \ell) = \lambda.B$ with $B \in \mathrm{NF}$. We distinguish the following subcases:

Subcase $\mathsf{S}_1 \notin \mathrm{WHNF}_\mathsf{C}$**:** Then by Lemma 6.10 we know that $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1' \in \mathrm{ST}_\mathsf{C}$. By rule $(\xi\widetilde{\rho})$, we have that $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \lambda.\mathsf{S}_1' \equiv \mathsf{S}'$. If S_1' is not in NF_C, then S' is not in NF_C either and the lemma holds. If S_1' is in NF_C, then S' is in NF_C and we know by Lemma 6.10 that $\mathsf{S}_1' \cong B$, and thus $\mathsf{S}' \equiv \lambda.\mathsf{S}_1' \cong \lambda.B \equiv T$ and the lemma holds.

Subcase $\mathsf{S}_1 \in \mathrm{WHNF}_\mathsf{C}$ **but** $\mathsf{S}_1 \notin \mathrm{NF}_\mathsf{C}$**:** Then by the induction hypothesis we have that $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1' \in \mathrm{ST}_\mathsf{C}$. By rule $(\xi\widetilde{\rho})$, we have that $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}} \lambda.\mathsf{S}_1' \equiv \mathsf{S}'$. If S_1' is not in NF_C, then S' is not in NF_C either and the lemma holds. If S_1' is in NF_C, then S' is in NF_C and we know by the induction hypothesis that $\mathsf{S}_1' \cong B$, and thus $\mathsf{S}' \equiv \mathsf{S}_1 \cdot \mathsf{S}_2' \cong MN \equiv T$ and we are done. □

Our goal now is to show that administrative reduction on a well-formed start closure either delivers a closure normal form, or it expands the closure until it locates the next $\beta\widetilde{\rho}$-redex to be contracted.

Lemma 6.12 (Administrative reduction locates next beta redex)**.** *Let* $\ell \geq 0$, $\mathsf{S} \in \mathrm{ST}_\mathsf{C}$ *and* $M \in \Lambda$ *such that* $M \longrightarrow_{no} M'$. *If* $wf(\mathsf{S}, \ell)$ *and* $\sigma(\mathsf{S}, \ell) = M$, *then there exists* $\mathsf{X} \in \mathrm{EX}_\mathsf{C}^\ell$ *such that* $wf(\mathsf{X}, \ell)$ *and* $\mathsf{S} \xrightarrow{\ell}_{\widetilde{\rho}}^{*} \mathsf{X}$.

Proof The proof goes by induction on $h(\mathsf{S})$, generalising the property over ℓ. The cases where $\mathsf{S} \equiv \lfloor n \rfloor$ and $\mathsf{S} \equiv \overline{n}$ are impossible because we know that M is not in NF. We first assume that $\mathsf{S} \notin \mathrm{WHNF}_\mathsf{C}$ and consider the following cases:

Case $\mathsf{S} \equiv n[\rho]$**:** Since $\sigma(n[\rho], \ell) = M$ and $M \longrightarrow_{no} M'$, then $n < |\rho|$ and $n[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} n^{\mathrm{th}}(\rho)$ where $n^{\mathrm{th}}(\rho) \not\equiv \overline{m}$, and the lemma holds by applying the induction hypothesis to $n^{\mathrm{th}}(\rho)$.

Case $\mathsf{S} \equiv (MN)[\rho]$**:** Then $(MN)[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} M[\rho] \cdot N[\rho]$ and by applying the induction hypothesis to $M[\rho] \cdot N[\rho]$.

Case $\mathsf{S} \equiv (\lambda.B)[\rho]$**:** Then $(\lambda.B)[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \lambda.B[\overline{\ell+1} : \rho]$ and by applying the induction hypothesis to $\lambda.B[\overline{\ell+1} : \rho]$.

Case $\mathsf{S} \equiv \mathsf{S}_1 \cdot N[\rho]$ **where** $\mathsf{S}_1 \in \mathrm{ST}_\mathsf{C}$ **and** $\mathsf{S}_1 \notin \mathrm{WHNF}_\mathsf{C}$**:** We distinguish the following subcases:

Subcase $\mathsf{S}_1 \equiv (\lambda.B)[\rho']$**:** By rule $(\mu 1\widetilde{\rho})$ then $\mathsf{S}_1 \cdot N[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} (\lambda.B[\overline{\ell+1} : \rho]) \cdot N[\rho'] \equiv \mathsf{X}$ where X is an ℓ-well-formed expanded closure at level ℓ, and the lemma holds.

Subcase $\mathsf{S}_1 \not\equiv (\lambda.B)[\rho']$**:** By Lemma 6.10 we know that $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1' \in \mathrm{ST}_\mathsf{C}$. Since $\mathsf{S}_1 \notin \mathrm{WHNF}_\mathsf{C}$ is a start closure that is not an abstraction closure, then S_1 can only be an index closure, an application closure, a nested application of proper closures or a nested application of proper closures with a formal parameter as the leftmost operator. Since S_1 is ℓ-well-formed, neither of these reduces in one step to a closure abstraction, and thus we know that S_1' is not a closure abstraction. By rule $(\mu 1\widetilde{\rho})$

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

we have that $\mathsf{S} \equiv \mathsf{S}_1 \cdot N[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1' \cdot N[\rho] \equiv \mathsf{S}'$ and the lemma holds by applying the induction hypothesis to S'.

Now we assume that $\mathsf{S} \in \mathrm{WHNF}_{\mathsf{C}}$ and consider the following cases:

Case $\mathsf{S} \equiv \mathsf{S}_1 \cdot N[\rho]$ **where** $\mathsf{S}_1 \in \mathrm{ST}_{\mathsf{C}}$**,** $\mathsf{S}_1 \in \mathrm{WHNF}_{\mathsf{C}}$ **and** $\mathsf{S}_1 \notin \mathrm{NF}_{\mathsf{C}}$**:** We know that S_1 is not a closure abstraction because then S would not be in $\mathrm{WHNF}_{\mathsf{C}}$. We distinguish whether $\sigma(\mathsf{S}_1, \ell) \in \mathrm{NF}$ or not. If $\sigma(\mathsf{S}_1, \ell) \in \mathrm{NF}$, then we apply Lemma 6.11 to S_1, and we obtain $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1'$. Since $\mathsf{S}_1 \in \mathrm{WHNF}_{\mathsf{C}}$ but S_1 is not an abstraction, by rule $(\mu\widetilde{\rho}2)$ then $\mathsf{S}_1 \cdot N[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{S}_1' \cdot N[\rho]$, and the lemma holds by applying the induction hypothesis to $\mathsf{S}_1' \cdot N[\rho]$. If $\sigma(\mathsf{S}_1, \ell) \notin \mathrm{NF}$, then we apply the induction hypothesis to S_1 and get $\mathsf{S}_1 \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{X}_1$. By rule $(\mu\widetilde{\rho}2)$ then $\mathsf{S}_1 \cdot N[\rho] \xrightarrow{\ell}_{\widetilde{\rho}} \mathsf{X}_1 \cdot N[\rho]$, and the lemma holds with $\mathsf{X} \equiv \mathsf{X}_1 \cdot N[\rho]$.
Case $\mathsf{S} \equiv \mathsf{S}_1 \cdot \mathsf{S}_2$ **where** $\mathsf{S}_1 \equiv \lfloor n \rfloor \{ \cdot \mathsf{N}_i\}^*$ **with** $\mathsf{N}_i \in \mathrm{NF}_{\mathsf{C}}$**, and** $\mathsf{S}_2 \in \mathrm{ST}_{\mathsf{C}}$ **and not in** NF_{C}**:** We have that $T \equiv MN \in \mathrm{NF}$ where $\sigma(\mathsf{S}_1, \ell) = M$ and $\sigma(\mathsf{S}_2, \ell) = N$. By rule (ν) we know that $N \longrightarrow_{no} N'$, and by the induction hypothesis $\mathsf{S}_2 \xrightarrow{\ell}{}^*_{\widetilde{\rho}} \mathsf{X}_2 \in \mathrm{EX}^{\ell}_{\mathsf{C}}$. By rule $(\nu\widetilde{\rho})$ then $\mathsf{S} \equiv \mathsf{S}_1 \cdot \mathsf{S}_2 \xrightarrow{\ell}{}^*_{\widetilde{\rho}} \mathsf{S}_1 \cdot \mathsf{X}_2 \equiv \mathsf{X}$ and the lemma holds.
Case $\mathsf{S} \equiv \lambda.\mathsf{S}'$**:** Straightforward by the induction hypothesis and rules (ξ) and $(\xi\widetilde{\rho})$. □

Now we show our main result. We first present a commutation theorem that states that normal-order (pure and $\lambda\widetilde{\rho}$'s) commutes with the substitution function σ. Lockstep simulation follows directly from the commutation theorem since the start term in a *no* reduction sequence uniquely determines the start closure in the corresponding $\widetilde{no}$ reduction sequence, and both *no* and $\widetilde{no}$ are reduction strategies.

Theorem 6.13 (Normal-order commutes with the substitution function)**.** *Let $\ell \geq 0$, S_0 be a start closure, and M_0 be a term. If $wf(\mathsf{S}_0, \ell)$ and $\sigma(\mathsf{S}_0, \ell) \equiv M_0$, then either $M_0 \in \mathrm{NF}$ and there exists $\mathsf{N} \in \mathrm{NF}_{\mathsf{C}}$ such that $\mathsf{S}_0 \xrightarrow{\ell}{}^*_{\widetilde{\rho}} \mathsf{N}$ and $\sigma(\mathsf{N}, \ell) \equiv M_0$, or otherwise there exist $M_1 \in \Lambda$, $\mathsf{X} \in \mathrm{EX}^{\ell}_{\mathsf{C}}$, and $\mathsf{S}_1 \in \mathrm{ST}_{\mathsf{C}}$ such that $wf(\mathsf{X}, \ell)$ and $wf(\mathsf{S}_1, \ell)$, and the following diagram commutes:*

$$\begin{array}{ccccc} \mathsf{S}_0 & & \xrightarrow{\ell}{}^*_{\widetilde{\rho}} \mathsf{X} & & \xrightarrow[\beta\widetilde{\rho}]{\ell} \mathsf{S}_1 \\ \Big\downarrow{\scriptstyle \sigma(\mathsf{S}_0, \ell)} & & & & \Big\downarrow{\scriptstyle \sigma(\mathsf{S}_1, \ell)} \\ M_0 & & \xrightarrow[no]{} & & M_1 \end{array}$$

In words, given a term M_0 and an ℓ-well-formed start closure S_0 such that $\sigma(\mathsf{S}_0, \ell) \equiv M_0$, either M_0 is in normal form and S_0 reduces via administrative reductions to a closure normal form isomorphic to M_0, or otherwise S_0 reduces to an expanded closure X at level ℓ via administrative reductions and the β-reduction step $M_0 \longrightarrow_{no} M_1$ corresponds to the $\widetilde{no}$ reduction sequence $\mathsf{S}_0 \xrightarrow{\ell}{}^*_{\widetilde{\rho}} \mathsf{X} \xrightarrow{\ell}_{\beta\widetilde{\rho}} \mathsf{S}_1$, which comprises one $\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ step after zero or more $\xrightarrow{\ell}_{\widetilde{\rho}}$ steps.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Proof If $M_0 \in \mathrm{NF}$, then by successively applying Lemmas 6.10 and 6.11 there exists $\mathsf{N} \in \mathrm{NF}_{\mathsf{C}}$ such that $\mathsf{N} \cong M_0$ and $\mathsf{S}_0 \xrightarrow{\ell}{}^{*}_{\widetilde{\rho}} \mathsf{N}$. Otherwise, $M_0 \notin \mathrm{NF}$ and then there exists M_1 such that $M_0 \longrightarrow_{no} M_1$ and by Lemma 6.12 there also exists $\mathsf{X} \in \mathrm{EX}^{\ell}_{\mathsf{C}}$ such that $wf(\mathsf{X}, \ell)$ and $\mathsf{S}_0 \xrightarrow{\ell}{}^{*}_{\widetilde{\rho}} \mathsf{X}$. By Lemma 6.9 we know that the reduction sequence on the top row of the commutation diagram is a $\widetilde{no}$ reduction sequence. From $\mathsf{S}_0 \xrightarrow{\ell}{}^{*}_{\widetilde{\rho}} \mathsf{X}$ we have $\sigma(\mathsf{S}_0, \ell) \equiv \sigma(\mathsf{X}, \ell)$. And from $\mathsf{X} \in \mathrm{EX}^{\ell}_{\mathsf{C}}$ we know that X reduces by a $\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ step to the start closure S_1. Hence, it is sufficient to prove the theorem when no $\widetilde{\rho}$-reduction occurs. For that, we do an induction over X (as an expanded closure at level ℓ), generalising the property over ℓ. Three cases arise:

Case $\mathsf{X} \equiv (\lambda.B[\overline{\ell+1} : \rho]) \cdot N[\rho']\,\{\cdot\, T_i[\rho_i]\}^*$**:** We have

$$M_0 \equiv (\lambda.\sigma(B[\overline{\ell+1} : \rho], \ell+1))(\sigma(N[\rho'], \ell))\{\sigma(T_i[\rho_i], \ell)\}^*,$$

which reduces by a $\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ step to the start closure $\mathsf{S}_1 \equiv B[N[\rho'] : \rho]\,\{\cdot\, T_i[\rho_i]\}^*$. The commuting condition $\sigma(B[N[\rho'] : \rho], \ell) \equiv [\sigma(N[\rho'], \ell)/0](\sigma(B[\overline{\ell+1} : \rho], \ell+1))$ holds by Lemma 6.7, the antecedent of Lemma 6.7 being a direct consequence of $wf(\mathsf{X}, \ell)$.

Case $\mathsf{X} \equiv \lambda.\mathsf{X}'$**:** We have that S_1 is necessarily of the form $\lambda.\mathsf{S}'_1$. From $wf(\mathsf{X}, \ell)$, we have $wf(\mathsf{X}', \ell+1)$. By the induction hypothesis, we know that $\sigma(\mathsf{X}', \ell+1) \longrightarrow_{no} \sigma(\mathsf{S}_1, \ell+1)$ and that $\mathsf{X}' \xrightarrow{\ell+1}_{\beta\widetilde{\rho}} \mathsf{S}'_1$. Hence, $\sigma(\mathsf{X}, \ell) \equiv \sigma(\lambda.\mathsf{X}', \ell) \longrightarrow_{no} \sigma(\lambda.\mathsf{S}'_1, \ell) \equiv \sigma(\mathsf{S}_1, \ell)$ and $\lambda.\mathsf{X} \xrightarrow{\ell}_{\beta\widetilde{\rho}} \lambda.\mathsf{S}'_1$.

Case $\mathsf{X} \equiv \lfloor n \rfloor\,\{\cdot\, \mathsf{N}_i\}^* \cdot \mathsf{X}'\,\{\cdot\, T_j[\rho_j]\}^*$ **where all** $\mathsf{N}_i \in \mathrm{NF}_{\mathsf{C}}$**:** In this case, the $\beta\widetilde{\rho}$-redex of X is necessarily in X', implying that $\mathsf{S}_1 \equiv \lfloor n \rfloor\,\{\cdot\, \mathsf{N}_i\}^* \cdot \mathsf{S}'_1\,\{\cdot\, T_j[\rho_j]\}^*$. From $wf(\mathsf{X}, \ell)$, we have $wf(\mathsf{X}', \ell)$. By the induction hypothesis we know that $\sigma(\mathsf{X}', \ell) \longrightarrow_{no} \sigma(\mathsf{S}'_1, \ell)$ and that $\mathsf{X}' \xrightarrow{\ell}_{\beta\widetilde{\rho}} \mathsf{S}'_1$. Hence,

$$\begin{aligned}\sigma(\mathsf{X}, \ell) &\equiv \sigma(\lfloor n \rfloor\,\{\cdot\, \mathsf{N}_i\}^* \cdot \mathsf{X}'\,\{\cdot\, T_j[\rho_j]\}^*,\ \ell)\\ &\longrightarrow_{no} \sigma(\lfloor n \rfloor\,\{\cdot\, \mathsf{N}_i\}^* \cdot \mathsf{S}'_1\,\{\cdot\, T_j[\rho_j]\}^*,\ \ell) \equiv \sigma(\mathsf{S}_1, \ell)\end{aligned}$$

and $\lfloor n \rfloor\,\{\cdot\, \mathsf{N}_i\}^* \cdot \mathsf{X}'\,\{\cdot\, T_j[\rho_j]\}^*,\ \ell) \xrightarrow{\ell}_{\beta\widetilde{\rho}} \lfloor n \rfloor\,\{\cdot\, \mathsf{N}_i\}^* \cdot \mathsf{S}'_1\,\{\cdot\, T_j[\rho_j]\}^*$. □

Corollary 6.14 (Lockstep simulation)**.** *Let M be a term. Then the no reduction sequence starting at M and the $\widetilde{no}$ reduction sequence starting at the closure $M[\varepsilon]$ and at the nesting level 0 correspond to each other in lockstep. That is, each $\longrightarrow_{no}$ step in the no reduction sequence corresponds to a segment of the $\widetilde{no}$ reduction sequence that comprises one $\xrightarrow{0}_{\beta\widetilde{\rho}}$ step and zero or more $\xrightarrow{0}_{\widetilde{\rho}}$ steps that surround the $\xrightarrow{0}_{\beta\widetilde{\rho}}$ step, and vice versa. In words, there is a one-to-one correspondence between the $\longrightarrow_{no}$ steps and the non-administrative $\xrightarrow{0}_{\widetilde{no}}$ steps.*

Proof Straightforward from Theorem 6.13, since $M[\varepsilon]$ is a 0-well-formed start closure and both *no* and $\widetilde{no}$ are reduction strategies. □

Consider the *no* reduction sequence from Example 3.1 (page 10). It corresponds in lockstep with the $\widetilde{no}$ reduction in Figure 9, as shown by the following commuting diagrams

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

where the closure and term on the right in the first diagram are respectively the closure and term on the left in the second diagram.

$$\begin{array}{ccc}
(\lambda.0((\lambda.0)((\lambda.0)0)))[\varepsilon] & \xrightarrow[\widetilde{\rho}]{0\,*} & \lambda.\lfloor 0\rfloor\cdot((\lambda.0[\overline{2}{:}\overline{1}])\cdot((\lambda.0)0)[\overline{1}]) \\
\sigma(_,0)\downarrow & & \\
\lambda.0((\lambda.0)((\lambda.0)0)) & \xrightarrow[no]{} & \lambda.0((\lambda.0)0)
\end{array}
\qquad
\begin{array}{ccc}
\xrightarrow[\beta\widetilde{\rho}]{0} \lambda.\lfloor 0\rfloor\cdot 0[((\lambda.0)0)[\overline{1}]{:}\overline{1}] & & \\
& & \sigma(_,0)\downarrow \\
& & \lambda.0((\lambda.0)0)
\end{array}$$

$$\begin{array}{ccc}
\lambda.\lfloor 0\rfloor\cdot 0[((\lambda.0)0)[\overline{1}]:\overline{1}] & \xrightarrow[\widetilde{\rho}]{0\,*} & \lambda.\lfloor 0\rfloor\cdot((\lambda.0[\overline{2}:\overline{1}])\cdot 0[\overline{1}]) \\
\sigma(_,0)\downarrow & & \\
\lambda.0((\lambda.0)0) & \xrightarrow[no]{} & \lambda.0\,0
\end{array}
\qquad
\begin{array}{ccc}
\xrightarrow[\beta\widetilde{\rho}]{0} \lambda.\lfloor 0\rfloor\cdot 0[0[\overline{1}]:\overline{1}] & & \\
& & \sigma(_,0)\downarrow \\
& & \lambda.0\,0
\end{array}$$

All the closures above are 0-well formed. Additionally, $(\lambda.0((\lambda.0)((\lambda.0)0)))[\varepsilon]$ and $\lambda.\lfloor 0\rfloor\cdot 0[((\lambda.0)0)[\overline{1}]:\overline{1}]$ are start closures in ST_{C}, and $\lambda.\lfloor 0\rfloor\cdot((\lambda.0[\overline{2}:\overline{1}])\cdot((\lambda.0)0)[\overline{1}])$ and $\lambda.\lfloor 0\rfloor\cdot((\lambda.0[\overline{2}:\overline{1}])\cdot 0[\overline{1}])$ are expanded closures in $\mathrm{EX}^0_{\mathsf{C}}$. The expanded sub-closures $(\lambda.0[\overline{2}:\overline{1}])\cdot((\lambda.0)0)[\overline{1}]$ and $(\lambda.0[\overline{2}:\overline{1}])\cdot 0[\overline{1}]$ in $\mathrm{EX}^1_{\mathsf{C}}$ correspond respectively to redexes $(\lambda.0)((\lambda.0)0)$ and $(\lambda.0)0$ in the *no* reduction sequence from Example 3.1.

7 Related and future work

Derivations of full-reducing abstract machines. The derivation of improved KN from $\lambda\widetilde{\rho}$'s structural operational semantics given in García-Pérez *et al.* (2013) is fully implemented and differs significantly from the sketched derivation of Munk (2008). For a detailed comparison and criticism, see pp. 169–174 of García-Pérez (2014). The derivation also differs from Danvy *et al.* (2013) who present a derivation for a different full-reducing machine of Curien (1986).

Full-reduction and hybridisation. In García-Pérez *et al.* (2013, p. 90), there is an inaccurate statement of lockstep simulation. In that paper $\lambda\widetilde{\rho}$-normal-order is defined in several styles of operational semantics (structural, context-based and natural), with implementations provided for each of them. Of particular interest is the use of hybrid style for the last two. Normal-order is a hybrid strategy in nature (contains a weaker subsidiary strategy, namely, call-by-name) and it is thus better defined in hybrid style. See García-Pérez (2014, Section 5.3) and García-Pérez & Nogueira (2014) for more on hybrid style and hybrid nature.

Calculi with lexical adjustment. In Section 1.2 we have already discussed the framework for abstract machines of Biernacka & Danvy (2007). Their substitution function from $\lambda\widehat{\rho}$-terms to pure terms carries along a lexical offset that is used to adjust indices when retrieving bindings from the environment. The offset is incremented when crossing a lambda and is reset to zero after retrieving a binding from the environment. Similarly, the environments of the full-reducing machines of Munk (2008) and Danvy *et al.* (2013)

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

carry along an offset that is used at binding lookup. In Section 4.1 we have referred to the use of this extra lexical offset as *lexical adjustment*.

Lexical adjustment is also present in the $\lambda\chi$ calculus of Lescanne & Rouyer-Degli (1995) which, unlike most calculi, uses a de-Bruijn-levels representation of terms. Substitutions are decorated with a depth that is incremented when crossing a lambda. The depth is employed to recalculate the de Bruijn levels of the names in a binding when the latter is retrieved from the environment. This resembles the lexical adjustment of Biernacka & Danvy (2007) but for de Bruijn levels instead of de Bruijn indices.

All of Lescanne & Rouyer-Degli (1995), Biernacka & Danvy (2007), Munk (2008) and Danvy *et al.* (2013) recalculate levels in bindings retrieved from the environment, or adjust indices at binding lookup.

In contrast, the use of index alignment and parameters-as-levels do away with lexical adjustment. Index calculations take place at the point where reduction encounters a free variable or a formal parameter.

Calculi of explicit substitutions. Calculi of explicit substitutions have been extensively studied, e.g. Abadi *et al.* (1991); Curien *et al.* (1996); Kesner (2007, 2009). Our $\lambda\widetilde{\rho}$ calculus uses environments and is not a calculus of explicit substitutions in the tradition of the seminal $\lambda\sigma$ calculus (Abadi *et al.*, 1991). Environments are instances of explicit substitutions but not the opposite. Environments are not composable like explicit substitutions, and terms and closures (terms with environments) are separated sorts.

The usual concerns for calculi of explicit substitutions (confluence, strong normalisation, preservation of strong normalisation, simulation, composition of substitutions, etc.) do not apply to $\lambda\widetilde{\rho}$ which is untyped and not strongly normalising. Instead of preservation of strong normalisation (i.e. if a pure term is strongly normalising, then so is its corresponding closure), because of lockstep simulation $\lambda\widetilde{\rho}$ upholds preservation of normalisation (i.e. if the pure term has a normal form, then so does its corresponding closure, and the former is obtained from the latter via σ). Moreover, the typed term of $\lambda\sigma$ which does not terminate (Melliès, 1995) is impossible to write in $\lambda\widetilde{\rho}$ not for lack of types, which may well be added to $\lambda\widetilde{\rho}$, but because of environments, index alignment and parameters-as-levels. The reader may try to reduce the typed term by hand using Figure 10, or by machine using the normalisers implemented in García-Pérez *et al.* (2013).

The linear substitution calculus (hereafter LSC) is a calculus of explicit substitutions which takes into account permutations of substitutions and structural equivalences which can be optimised away (distilled) when they do not lead to redex contraction. In particular, the LSC's structural equivalences are between machine transitions of the full-reducing Milner Abstract Machine (hereafter MAM) (Accattoli *et al.*, 2015; Accattoli, 2016). However, the LSC 'silently works modulo α-equivalence' (Accattoli *et al.*, 2015, p. 5; Accattoli, 2016, p. 4) or, put similarly, performs 'on-the-fly α-equivalence' (Accattoli *et al.*, 2015, p. 4). That is, it assumes a plain term representation and performs α-conversion. On the other hand, the MAM employs an underspecified so-called 'well-namedness' convention for terms where every bound and free variable differ (Accattoli *et al.*, p. 11; Accattoli, 2016, p. 6). This convention is underspecified because at face value it is the same as the *Barendregt convention*, (Barendregt, 1990) which is unstable under full-reduction due to the copying of operators in abstraction bodies,

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

e.g. (Pierce, 2002, p. 75). Witness the counter-example $(\lambda z.zz)(\lambda x.\lambda y.xy)$ which requires α-conversion. Similarly with explicit substitution: $(zz)[z \mapsto \lambda x.\lambda y.xy]$. The LSC and MAM are disconnected at the underspecified representation of terms. A proper representation has to be chosen for the machine such that (i) it corresponds with the calculus's term representation, (ii) it facilitates inter-derivation by program transformation and (iii) it is usable in mechanised proofs, which is the case for terms in de Bruijn notation but not for terms up to α-equivalence (Aydemir *et al.*, 2005).

Naturally, distilling structural equivalences is of great value for any calculus. There are, indeed, structural equivalences among KN's transitions to exploit as optimisation steps in the spirit of the LSC. Such equivalences must take into account index alignment and parameters-as-levels. For instance, consider a closure $((\lambda.B)N)[\rho]$ in some $\widetilde{no}$ reduction sequence at level ℓ. The closure reduces by a single $\xrightarrow{\ell}_{\beta\widetilde{\rho}}$ step (plus administrative steps) to $B[N[\rho] : \rho]$. This corresponds to the following single *no* step:

$$(\lambda.\sigma(B[\overline{\ell+1} : \rho], \ell+1))(\sigma(N[\rho], \ell)) \longrightarrow_{no} [\sigma(N[\rho], \ell)/0](\sigma(B[\overline{\ell+1} : \rho, \ell+1)))$$

By Lemma 6.7, the following holds:

$$\sigma(([N/0]B)[\rho], \ell) = [\sigma(N[\rho], \ell)/0](\sigma(B[\overline{\ell+1} : \rho, \ell+1)))$$

Therefore, by Lemma 6.7 and Theorem 6.13 the $\widetilde{no}$ reduction sequences of $([N/0]B)[\rho]$ and $B[N[\rho] : \rho]$ have a common reduct at some reduction step. This can be construed as 'the composition of implicit substitution $[N/0]$ with explicit substitution $[\rho]$ is equivalent to explicit substitution $[N[\rho] : \rho]$'. This structural equivalence is reminiscent of the one given in Accattoli & Kesner (2012, p.19) for the λj calculus that is a precursor of the LSC.

Calculi with locally nameless representation for terms. Such term representation uses de Bruijn indices to represent bound variables and fresh names to represent variables that are free in a scope (Pollack, 1994; Aydemir *et al.*, 2008; Charguéraud, 2012). Names are employed, among other things, to avoid de Bruijn index shifting. The operation of *variable opening* on an abstraction $\lambda.B$ recursively traverses B and maps the applied occurrences of the bound variable (the appropriate de Bruijn index in B) to a fresh name. This operation requires, like shifting, a cutoff parameter that keeps track of the nesting level during recursive traversal. It also requires an operation for fresh name generation, typically implemented using natural numbers for names and increment for freshness. The inverse operation of *variable closing* builds an abstraction $\lambda.B$ from a term B and a name n by replacing all occurrences (no shadowing possible in this representation) of the name n in B by the appropriate de Bruijn index, also using a cutoff parameter to keep track of the nesting level.

Index-aligned environments and parameters-as-levels are a form of locally nameless representation that, instead of recursive traversals, uses an environment and calculations to perform variable opening and closing. As shown in Figure 10, rules (LAM$\widetilde{\rho}$) and ($\xi_{\widetilde{\rho}}$) correspond to variable opening in which a fresh name (incremented nesting level) is given to the formal parameter (and pushed on the environment). Symmetrically, rules (PAR$\widetilde{\rho}$) and ($\xi_{\widetilde{\rho}}$) correspond to variable closing. The result index calculations of the first rule recovers the index. The second rule leaves the abstraction scope 'at return time'. The distinction

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

between free out-of-bounds variables and free in-bounds variables (applied occurrences) is taken care of by axiom (FRE$\widetilde{\rho}$).

Charguéraud's 2nd solution[8] to the POPLMark Challenge resembles Crégut's parameters-as-levels, but the environment stores the types of formal parameters, not their levels. We think index-aligned environments, parameters-as-levels and balanced derivations deserve to be explored as a possible solution to the POPLMark Challenge.

Weak lazy leftless systems. A mechanism reminiscent of index alignment can be found in the 'weak lazy leftless' type system of Kiselyov (2018). The system keeps track of 'ignored context terms' (akin to parameters of unapplied lambda abstractions) by pushing on the judgement's environment an 'any type' dummy placeholder. This type system is shown to be equivalent to the type system without the placeholder. A denotational semantics is given from terms of the weak lazy leftless type system to SKI combinators. It contains a weakening rule (EW) that ignores the dummy placeholder, and an (EAbs1) rule that performs the so-called K-optimisation (Peyton-Jones, 1987) which replaces an abstraction over an ignored variable by the constant combinator. All the de Bruijn indices of terms of the type system point to a binding on the (judgement's) environment.

Future work. The $\overline{\lambda}\mu\tilde{\mu}\mathtt{r}\uparrow$ calculus (Ariola *et al.*, 2009) extends the $\overline{\lambda}\mu\tilde{\mu}$ sequent calculus (Curien & Herbelin, 2000) by adding de Bruijn indices and explicit substitutions. In Ariola *et al*. (2009), they show that $\overline{\lambda}\mu\tilde{\mu}\mathtt{r}\uparrow$ simulates both KAM and CEK. In sequent calculi, reduction occurs at a bounded depth from the root of the term, such that it is possible to define in the calculus a tail-recursive evaluator that captures the dispatch function carried out by an abstract machine. Sequent calculi were designed to encode a particular deductive structure and hence they are typed and strongly normalising. Our calculus is untyped and our approach concerns KN's ingredients for full reduction. Exploring whether those ingredients can be carried to sequent calculi and whether this results in capturing the dispatch function of KN are interesting questions that constitute future work.

As we have said many times by now, the one-step reduction relation of $\lambda\widetilde{\rho}$ is a reduction strategy. A one-step relation proper (like one-step $\longrightarrow_\beta$ in the pure lambda calculus) may be given which reduces the operands and/or the bindings in environments. This relation would stage reduction in a way reminiscent of normalisation by evaluation: ephemeral expansion does not proceed under lambda until a weak head normal form of the closure/binding is reached, $\beta\widetilde{\rho}$-contraction and lookup take place anywhere, and result index calculation only takes place at the end of the process, in the 'frozen' normal-form context in which the 'active components' of the input closure occur (see Definition 2.3 in Barendregt *et al.* (1987) for a definition of active component in the pure lambda calculus). This reduction would be reminiscent of needed reduction (Barendregt *et al.*, 1987) but lifted to closures. The staged character of the reduction relation can be expressed in hybrid style (García-Pérez, 2014, Section 5.3); (García-Pérez & Nogueira, 2014). With this reduction relation, it could be possible to simulate other reduction strategies, in particular call-by-value and related. We conjecture this reduction relation would simulate in lockstep a subrelation of needed reduction mentioned above, modulo the administrative

[8] http://www.chargueraud.org/research/2006/poplmark.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

steps of expansion, lookup and index calculation. We think the proof would proceed by induction given the machinery presented in this paper. This constitutes future work.

Acknowledgments

We are deeply grateful to Pierre-Yves Strub for mechanising a large portion of the lock-step simulation proof in the Coq proof assistant. Pierre-Yves helped us sanitise the paper proof and improved several definitions and lemmata. We are also deeply grateful to our JFP editor, Jeremy Gibbons, for his excellent editorial support. Finally, we are deeply grateful to all the anonymous reviewers for their patience and very pertinent suggestions which have helped us improve the original submission substantially in terms of correctness, presentation and scope. In particular, a comment by one reviewer led us to state and prove Theorem 6.13 as a proper commuting diagram rather than as a conditional diagram. The authors have carried out the research presented in this paper part-time at different institutions: the Babel Research Group of Universidad Politécnica de Madrid (both authors), Reykjavik University and The IMDEA Software Institute (Álvaro), and ESNE, University School of Design, Innovation and Technology (Pablo). This work 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, by the Icelandic Research Fund through project NOSOS 141558-053 and by the European Research Council through project RACCOON H2020-EU 714729.

References

Abadi, M., Cardelli, L., Curien, P.-L. & Lévy, J.-J. (1991) Explicit substitutions. *J. Funct. Program.* **1**(4), 375–416.

Accattoli, B. (2016) The useful MAM, a reasonable implementation of the strong lambda-calculus. In Proceedings of the 23rd Workshop on Logic, Language, Information and Computation. LNCS, vol. 9803. Springer, pp. 1–21.

Accattoli, B. & Kesner, D. (2012) Preservation of strong normalisation modulo permutations for the structural lambda-calculus. *Log. Methods Comput. Sci.* **8**(1), 1–44.

Accattoli, B., Barenbaum, P. & Mazza, D. (2015) A strong distillery. In Proceedings of the 13th Asian Symposium on Programming and Systems. LNCS, vol. 9458. Springer, pp. 231–250.

Ariola, Z. M., Bohannon, A. & Sabry, A. (2009) Sequent calculi and abstract machines. *ACM Trans. Program. Lang. Syst.* **31**(4), 13:1–13:48.

Aydemir, B., Charguéraud, A., Pierce, B. C., Pollack, R. & Weirich, S. (2008) Engineering formal metatheory. In Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM Press, pp. 3–15.

Aydemir, B. E., Bohannon, A., Fairbairn, M., Foster, J. N., Pierce, B. C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S. & Zdancewic, S. (2005) Mechanized metatheory for the masses: The POPLmark challenge. In Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics. LNCS, vol. 3603. Springer, pp. 50–65.

Barendregt, H. P. (1984) *The Lambda Calculus, Its Syntax and Semantics*. North Holland.

Barendregt, H. P. (1990) Functional programming and lambda calculus. In Chapter 7: Handbook of Theoretical Computer Science, vol. B. Elsevier / MIT Press, pp. 321–364.

Barendregt, H. P., Kennaway, J. R., Klop, J. W. & Sleep, M. R. (1987) Needed reduction and spine strategies for the lambda calculus. *Inf. Comput.* **75**, 191–231.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Biernacka, M. & Danvy, O. (2007) A concrete framework for environment machines. *ACM Trans. Comput. Log.* **9**(1), 6:1–6:30.

Charguéraud, A. (2012) The locally nameless representation. *J. Autom. Reas.* **49**(3), 363–408.

Crégut, P. (2007) Strongly reducing variants of the Krivine abstract machine. *Higher-Order Symb. Comput.* **20**(3), 209–230.

Curien, P.-L. (1986) *Categorical Combinators, Sequential Algorithms and Functional Programming*. John Wiley & Sons.

Curien, P.-L. (1991) An abstract framework for environment machines. *Theor. Comput. Sci.* **82**(2), 389–402.

Curien, P.-L. & Herbelin, H. (2000) The duality of computation. In Proceedings of the 5th International Conference on Functional Programming. SIGPLAN Notices, vol. 35, issue no.9. ACM Press, pp. 233–243.

Curien, P.-L., Hardin, T. & Lévy, J.-J. (1996) Confluence properties of weak and strong calculi of explicit substitutions. *J. ACM* **43**(2), 362–397.

Curry, H. B. & Feys, R. (1958) *Combinatory Logic*, vol. 1. North-Holland.

Danvy, O. (2009) From reduction-based to reduction-free normalization. In 6th International School on Advanced Functional Programming, Revised Lectures. LNCS. Springer, pp. 66–164.

Danvy, O., Johannsen, J. & Zerny, I. (2011) A walk in the semantic park. In Proceedings of the 20th ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. ACM Press, pp. 1–12.

Danvy, O., Milikin, K. & Munk, J. (2013) *A correspondence between full normalization by reduction and full normalization by evaluation*. Talk presented at *A scientific meeting in honor of Pierre-Louis Curien*, Venice, 9–11 September.

de Bruijn, N. G. (1972) Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Math.* **34**(5), 381–392.

Diehl, S., Hartel, P. & Sestoft, P. (2000) Abstract machines for programming language implementation. *Future Gener. Comput. Syst.* **16**(7), 739 – 751.

Felleisen, M. (1987) *The Calculi of Lambda-v-cs Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. Ph.D. thesis, Department of Computer Science, Indiana University.

García-Pérez, Á. (2014) *Operational Aspects of Full Reduction in Lambda Calculi*. Ph.D. thesis, ETSI Informáticos, Universidad Politécnica de Madrid.

García-Pérez, Á. & Nogueira, P. (2014) On the syntactic and functional correspondence between hybrid (or layered) normalisers and abstract machines. *Sci. Comput. Program.* **95**(Part 2), 176–199.

García-Pérez, Á., Nogueira, P. & Moreno-Navarro, J. J. (2013) Deriving the full-reducing Krivine machine from the small-step operational semantics of normal order. In Proceedings of the 15th International Symposium on Principles and Practice of Declarative Programming. ACM Press, pp. 85–96.

Grégoire, B. & Leroy, X. (2002) A compiled implementation of strong reduction. In Proceedings of the 7th International Conference on Functional Programming, vol. 37, issue no. (9), pp. 235–246.

Kahn, G. (1987) Natural semantics. In Proceedings of Symposium on Theoretical Aspects of Computer Science. LNCS, vol. 247. Springer, pp. 22–39.

Keller, R. M. (1976) Formal verification of parallel programs. *Commun. ACM* **19**(7), 371–384.

Kesner, D. (2007) The theory of calculi with explicit substitutions revisited. In Proceedings of the 21st International Workshop on Computer Science Logic. LNCS, vol. 4646. Springer, pp. 238–252.

Kesner, D. (2009) A theory of explicit substitutions with safe and full composition. *Log. Methods Comput. Sci.* **5**(3), 1–29.

Kiselyov, O. (2018) λ to SKI, semantically - declarative pearl. In Proceedings of the 14th International Symposium on Functional and Logic Programming. LNCS. Springer, pp. 33–50.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press

Krivine, J.-L. (2007) A call-by-name lambda-calculus machine. *Higher-Order Symb. Comput.* **20**(3), 199–207.

Lescanne, P. & Rouyer-Degli, J. (1995) Explicit substitutions with de Bruijn's levels. In Proceedings of the 6th International Conference on Rewriting Techniques and Applications. LNCS, vol. 914. Springer, pp. 294–308.

Melliès, P.-A. (1995) Typed lambda-calculi with explicit substitutions may not terminate. In Proceedings of the 2nd International Conference on Typed Lambda Calculi and Applications. LNCS, vol. 902. Springer, pp. 328–334.

Munk, J. (2008) *A Study of Syntactic and Semantic Artifacts and its Application to Lambda Definability, Strong Normalization, and Weak Normalization in the Presence of State*. M.Phil. thesis, BRICS, Aarhus University.

Paulson, L.C. (1996) *ML For the Working Programmer*. 2nd ed. New York, NY: Cambridge University Press.

Peyton-Jones, S. (1987) *The Implementation of Functional Programming Languages*. Prentice-Hall.

Pierce, B. (2002) *Types and Programming Languages*. The MIT Press.

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

Plotkin, G. (1981) *A structural approach to operational semantics*. Technical Report DAIMI FN-19. Department of Computer Science, Aarhus University, Denmark.

Pollack, R. (1994) Closure under alpha-conversion. In Proceedings of the 1993 International Workshop on Types for Proofs and Programs. LNCS, vol. 806. Springer, pp. 313–332.

Ronchi Della Rocca, S. & Paolini, L. (2004) *The Parametric Lambda Calculus*. Springer.

Scherer, G. & Rémy, D. (2015) Full reduction in the face of absurdity. *Proceedings of Programming Languages and Systems - 24th European Symposium on Programming*. LNCS, vol. 9032. Springer, pp. 685–709.

Sestoft, P. (2002) Demonstrating lambda calculus reduction. In The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones. LNCS, vol. 2566. Springer, pp. 420–435.

https://doi.org/10.1017/S0956796819000017 Published online by Cambridge University Press