Deriving the Full-Reducing Krivine Machine from the Small-Step Operational Semantics of Normal Order [*]

Álvaro García-Pérez
IMDEA Software Institute and
Universidad Politécnica de Madrid
agarcia@babel.ls.fi.upm.es

Pablo Nogueira
Universidad Politécnica de Madrid
pablo@babel.ls.fi.upm.es

Juan José Moreno-Navarro
IMDEA Software Institute and
Universidad Politécnica de Madrid
juanjose.moreno@imdea.org

Abstract

We derive by program transformation Pierre Crégut's full-reducing Krivine machine KN from the structural operational semantics of the normal order reduction strategy in a closure-converted pure lambda calculus. We thus establish the correspondence between the strategy and the machine, and showcase our technique for deriving full-reducing abstract machines. Actually, the machine we obtain is a slightly optimised version that can work with open terms and may be used in implementations of proof assistants.

1. Introduction

An operational semantics is a mathematical description of the reduction of program terms. An operational semantics is underlied by a *reduction strategy* that specifies the order in which reducible subterms ('redices' for short, singular 'redex') are to be reduced. An operational semantics can be implemented. A *reduction-based normaliser* is a program implementing a context-based small-step operational semantics, or 'reduction semantics' for short. (Such a semantics defines reduction as the iteration of three steps [15]: uniquely decomposing a term into a term with a hole (a context) with a redex within the hole, contracting (reducing) the redex within the hole, and recomposing the resulting term.) A *reduction-free normaliser* is a program implementing a big-step natural semantics. Finally, *abstract machines* are first-order, tail-recursive implementations of state transition functions that, unlike virtual machines, operate directly on terms, have no instruction set, and no need for a compiler.

There has been considerable research on inter-derivation by program transformation of such 'semantic artefacts', of which the works [3, 5, 12, 14] are noticeable exemplars. The transformations consist of equivalence-preserving steps (CPS transformation, inverse CPS, defunctionalisation, refunctionalisation, inlining, lightweight fusion by fixed-point promotion, etc.) which establish a formal correspondence between the artefacts: that all implement the same reduction strategy.

Research in inter-derivation is important not only for the correspondences and general framework it establishes in semantics. More pragmatically, it provides a semi-automatic way of proving the correspondence between small-step semantics, big-step semantics, and existing *contrived* abstract machines, some of which are used in real applications, e.g., [19]. Such proofs otherwise require external mathematical machinery (a famous case in point is [23] concerning call-by-value and the SECD machine.) Furthermore, research in inter-derivation extends the repertoire of equivalence-preserving program-transformation steps, and brings about the discovery of *new* calculi, *new* abstract machines, and *new versions* of known machines which might be easier to define, or be better suited for optimisation.

Surprisingly, inter-derivation techniques are not used as often as they should. A recent example is [25] in which several semantic artefacts are defined for the gradually-typed lambda calculus, but their correspondences are conjectured when they could have been shown by inter-derivation. On a related note, the operational semantics that have been considered in inter-derivations have been *weak-reducing*, e.g., call-by-value or call-by-name. *Full-reducing* strategies have received less attention. Full-reducing strategies reduce terms fully and deliver (full-)normal-forms.[1] Full-reducing strategies are important and useful [7]. Two applications are program optimisation by partial evaluation, and type checking in proof assistants which may have to reduce some terms fully, e.g. [19].

In a recent work [17, 18] we have refined the current inter-derivation techniques to inter-derive semantic artefacts for full-reducing strategies, and have showcased the inter-derivation of

[*] Research partially funded by the Spanish 'Ministerio de Economía y Competitividad' through projects DESAFIOS10 TIN2009-14599-C03-00 and STRONGSOFT TIN2012-39391-C04-02, and by 'Comunidad de Madrid' through programme PROMETIDOS P2009/TIC-1465. The first author is supported by Comunidad de Madrid grant CPI/0622/2008 and by IMDEA Software Institute.

[1] We prefer 'full-reducing' to 'strong-' or 'strongly-reducing', as used by some authors, because the latter can be confused with 'strongly-normalising' which means something different. Strong normalisation is a property of a calculus. A full-reducing strategy in a strongly-normalising calculus always terminates, but it may diverge in a non-strongly-normalising calculus.

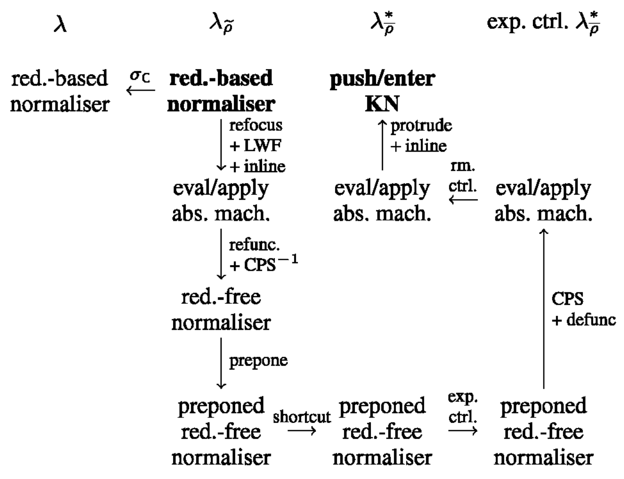

Figure 1. Derivation path of KN.

substitution-based artefacts for *normal order*, the standard, full-reducing strategy of the *pure* (untyped) lambda calculus. Normal order delivers the normal form of a term if it exists or diverges otherwise. As explained in Section 3, it is a *hybrid* strategy, i.e., it relies on an ancillary *subsidiary* strategy, call-by-name.

In the derivation we obtained a substitution-based, eval/apply, open-terms abstract machine that resembles Pierre Crégut's KN machine, a well-known machine that fully reduces pure lambda calculus terms [8]. However, KN is an environment-based, push/enter, closed-terms machine that uses de Bruijn indices and levels for representing terms. It has been proven (mathematically) that KN finds the normal form of a *closed* term when the normal form exists [8], but the actual correspondence between KN and normal order (that KN realises normal order) has remained unproven.

Contributions. In this paper we derive the original KN from a small-step operational semantics of normal order in a new calculus of closures, thus proving by means of program-transformation the correspondence between the strategy and the machine. Actually, what we obtain is a slightly optimised version of KN that can also work with *open* terms, and is therefore suitable for use in implementations of proof assistants [7, 19, 20].

There are four points to stress about our derivation. Our operational semantics are 'single-stage' [17, 18], i.e., they define a single but hybrid normal order strategy that relies on a subsidiary call-by-name strategy (Section 3). Consequently, we can use *single-layer* CPS without control delimiters, as opposed to a two-layer CPS or a single-layer CPS with control delimiters, as found in other works (Section 10). Second, following [14] we derive the reduction-based normaliser (and so the reduction semantics) from *search functions* that implement the compatibility rules of the structural operational semantics of normal order in our calculus of closures. Third, we introduce a non-standard but straightforward 'preponing' step and show that it is equivalence-preserving (Section 8). Last, we construct the grammar of continuation stacks as in [18]. In that work we showed that the stack is easily obtained from the grammar of the reduction semantics, but we only used the stack grammar to recover shallow inspection whereas in this paper we also use it to remove explicit control. This paper thus illustrates once more the importance of that step in derivations of full-reducing hybrid strategies.

Figure 1 illustrates the derivation path. The start and end points are shown in boldface. The figure extends the derivational taxonomy of [5, p.24] and summarises the contents of Sections 7.2 to 9.4 of this paper.

Here is a more detailed list of the contributions:

- We introduce the $\lambda_{\widehat{\rho}}$ calculus which naturally extends the $\lambda_{\widehat{\rho}}$ calculus[2] of Biernacka and Danvy [5] with de Bruijn levels (present in KN), closure abstractions, and absolute indices. The latter two are required for full-reduction. Closure abstractions are required to represent closures where the redex may occur under lambda, and absolute indices are required to represent 'neutral closures', i.e., non-redex closure applications. We formalise the translation from $\lambda_{\widehat{\rho}}$ to the pure lambda calculus λ by providing a substitution function σ_{C} that simulates capture-avoiding substitution in the pure lambda calculus.
- We define the small-step structural operational semantics of normal order in $\lambda_{\widehat{\rho}}$ and derive from the search functions that implement the compatibility rules the trampolined reduction-based normaliser, and from it the reduction-free normalisers. In other words, we derive the context-based and the big-step natural semantics of normal order in $\lambda_{\widehat{\rho}}$.
- In the reduction-free normalisers we can better justify the required *preponing* of certain normal order reduction steps to call-by-name, and justify the correctness of this non-standard but straightforward step (Section 8).
- After shortcut optimisation, which takes us to a version of $\lambda_{\widehat{\rho}}$ without ephemerals that we call $\lambda^{*}_{\widehat{\rho}}$, we have to introduce explicit control to combine two reduction-free normalisers into one. Using the correlation between explicit control and the continuation stack [17, 18], we finally obtain the open-terms version of KN by applying further standard derivation steps.

We have written all the code of the derivation in Standard ML, the traditional programming language of derivation papers. Though semi-automatically obtained, the code is rather long (we include all steps in detail) and language-specific. Therefore we present the semantic artefacts in mathematical notation and simply name in the paper the functions implementing the artefacts in the code, which is available online.[3] We have tested the code. We have not verified the transformations using a proof assistant for several reasons. First, most transformation steps are standard and easy to check by readers familiar with derivation papers in Standard ML. Second, involving a proof assistant or a dependently-typed language would result in a different paper for a different readership. We would have to explain additional proof techniques and the particulars of the assistant. See for instance [26] where several techniques (logical relations, etc) have to be introduced to obtain the weak-reducing KAM machine for the simply typed lambda calculus. And last, our calculus is untyped and our normalisers are partial functions which may diverge, and whether they do or not, is undecidable.

2. Preliminaries

We consider the pure untyped lambda calculus with de Bruijn indices [4], hereafter λ for short, whose terms are defined by the grammar $\Lambda \ ::= \ n \mid (\lambda.\Lambda) \mid (\Lambda\,\Lambda)$. A natural number n represents a variable bound to the nth lambda starting from 0, or to a free variable when n is greater than or equal to the nesting level. For example, the abstraction $\lambda.0$ is the identity function whereas $\lambda.1$ is a constant function delivering free 0 when the function is applied to an operand. Uppercase, often primed, letters M, M', N,

[2] The $\lambda_{\widehat{\rho}}$ calculus is itself an extension of Curien's lambda calculus of closures λ_{ρ} [9], which is an extension of the pure lambda calculus that adds closures for handling explicit substitutions [1, 20].

[3] `http://babel.ls.fi.upm.es/~agarcia/papers/KrivineFull`

B, etc, will range over elements of Λ. We use the standard precedence and association convention: applications associate to the left and abstraction binds tighter than application. The reader must be familiar with the usual lambda calculus notions of bound and free variables, redices $(\lambda.B)M$, syntactic equivalence $\equiv$, capture-avoiding substitution $[N/n]M$, β-contraction, and relations $\rightarrow_\beta$, $\rightarrow_\beta^*$, and $=_\beta$. A reduction strategy s of λ is a partial function that is a sub-relation of $\rightarrow_\beta^*$. We write $\rightarrow_s$ and $\Downarrow_s$ for the small-step and big-step definitions of s.

We define terms and calculi using EBNF grammars with regular expressions $\{\alpha\}^*$ (denoting zero or more occurrences of sentential form α) and $\{\alpha\}^?$ (denoting zero or one occurrence of α). For example, NF ::= λ.NF | $n\,\{\text{NF}\}^*$ defines the set of (full-)normal-forms. Some sentential forms of the second production are n, $n\,\text{NF}$, $n\,\text{NF}\,\text{NF}$, etc., which respectively associate as n, $(n\,\text{NF})$, $((n\,\text{NF})\,\text{NF})$, etc, according to the convention. In the text we use 'whnf' and 'nf' to abbreviate 'weak-head-normal-form' and '(full)-normal-form' respectively. The set of whnfs and nfs is defined in Figure 2 (bottom left). Notice that a term in nf is also in whnf.

3. Normal order in all substitution-based styles

Normal order is typically defined by the slogan 'contract the leftmost redex first', understanding 'leftmost' as in [11] or 'leftmost-outermost' when referring to the redex's position in the abstract syntax tree of the term. Normal order is hybrid, it relies on subsidiary call-by-name (which reduces terms to whnf) to avoid going prematurely 'under lambda' so as to discard unneeded potentially divergent subterms. Given an application MN, when call-by-name reduces the operator M to an abstraction $\lambda.B$ (a whnf) then the leftmost (outermost) redex $(\lambda.B)N$ is contracted next instead of the redices in B. For example, given the term $(\lambda.0\,\Omega)(\lambda.1)$ where Ω is a divergent subterm, since the operator is in whnf, normal order reduces that leftmost outermost redex to $(\lambda.1)\Omega$, and this term to 0, discarding Ω.

Figure 2 shows the structural (left) and natural (right) operational semantics of normal order. In the structural small-step there is no rule for variables because these are in normal-form. There are four rules for applications. The first (β) is well-known. The second ($\mu1$) says the redex must be searched for in the operator if the operator is not in whnf. These two rules make up subsidiary call-by-name. The third rule ($\mu2$) says the redex must be searched for in the operator if it is a whnf but not a nf nor an abstraction (if an abstraction then (β) is applicable). Finally, (ν) says the redex must be searched for in the operand if the operator is a nf but not an abstraction. The outermost application does not reduce to a redex and so the redex must be searched for in the operand. (Although a nf is also a whnf, rules ($\mu2$) and (ν) are non-overlapping because the third premiss in ($\mu2$) is not the case when $M \in$ NF.) Hereafter we shall refer to variables and non-redex applications as *neutral terms*, defined by the regular expression $n\,\{\Lambda\}^*$. Last, rule (ξ) provides structural compatibility with abstractions, that is, 'go under lambda'.

In the big-step, normal order $\Downarrow_{no}$ relies on subsidiary and uniform call-by-name $\Downarrow_{bn}$ to reduce operators to whnf (first premiss of rules RED_{no} and NEU_{no}), and then fully reduces the resulting redex (rule RED_{no}) or the resulting neutral (rule NEU_{no}). Finally, LAM_{no} says that normal order goes under lambda and VAR_{no} says normal order is an identity on variables. In contrast, call-by-name does not go under lambda and does not reduce operands in neutral terms.[4]

[4] Call-by-name in the pure untyped lambda calculus differs from call-by-name in the applied and implicitly typed calculus of [23] (which also assumes closed input terms) precisely in its treatment of neutral terms [24, p.421].

The structural and natural semantics in Figure 2 are *single-stage*. There is an alternative two-stage *eval-readback* [19] approach that defines reduction as the *composition* of two partial functions (i.e., two single-stage natural semantics), namely, an 'eval' function that delivers *intermediate* results, and a 'readback' function that distributes reduction over the subterms of the intermediate result. The eval-readback approach is a degenerate case of normalisation-by-evaluation [2] in which the value domain is the set of terms, and readback is 'reify' without the translation from domain values to terms. (The two-stage nature of eval-readback definitions is also present in their corresponding small-step semantics, where a reduction sequence consists of nested concatenations of eval and readback sequences.) Normal order is defined in eval-readback style as the composition $\Downarrow_{rn} \circ \Downarrow_{bn}$ where $\Downarrow_{bn}$ is eval and $\Downarrow_{rn}$ is readback:

$$\frac{}{n \Downarrow_{rn} n}\,\text{VAR}_{rn} \qquad \frac{B \Downarrow_{bn} B' \quad B' \Downarrow_{rn} B''}{\lambda.B \Downarrow_{rn} \lambda.B''}\,\text{LAM}_{rn}$$

$$\frac{M \Downarrow_{rn} M' \quad N \Downarrow_{bn} N' \quad N' \Downarrow_{rn} N''}{M\,N \Downarrow_{rn} M'\,N''}\,\text{APP}_{rn}$$

Readback takes input terms in whnf (no redex at the outermost level) which explains the lack of a contraction rule for it. The equivalence between single-stage and eval-readback, namely $\Downarrow_{no} = \Downarrow_{rn} \circ \Downarrow_{bn}$, can be proven by induction on derivations, or by program transformation using lightweight fusion by fixed-point promotion [21] (Section 8).

Now to the context-based reduction semantics. In addition to the grammar of terms and normal forms there is a grammar for reduction contexts and a contraction rule that applies (β) within the context hole.

$$\begin{array}{llcl}
\text{Red. context:} & \mathbf{C}_{no}[\,] & ::= & [\,] \mid \mathbf{C}_{bn}[\,]\,\Lambda \mid \lambda.\mathbf{C}_{no}[\,] \mid \mathbf{C}_{ne}[\,] \\
 & \mathbf{C}_{bn}[\,] & ::= & [\,] \mid \mathbf{C}_{bn}[\,]\,\Lambda \\
 & \mathbf{C}_{ne}[\,] & ::= & n\{\text{NF}\}^*\,\mathbf{C}_{no}[\,] \mid \mathbf{C}_{ne}[\,]\,\Lambda \\
\text{Contraction:} & \mathbf{C}_{no}[(\lambda.B)N] & \rightarrow_{no} & \mathbf{C}_{no}[[N/0]B]
\end{array}$$

Given a term M, it is either in nf or is uniquely decomposed into a context, derived from non-terminal $\mathbf{C}_{no}[\,]$, and a redex within the hole. The unique decomposition of $\mathbf{C}_{no}[\,]$ is proven by induction on terms [17, 18]. For example, the term $\lambda.(\lambda.0)0$ is decomposed into $\lambda.[(\lambda.0)0]$ with the context $\lambda.[\,]$ grammatically derived as follows: $\mathbf{C}_{no}[\,] \Rightarrow \lambda.\mathbf{C}_{no}[\,] \Rightarrow \lambda.[\,]$. Hybridisation is signalled by the presence of call-by-name subcontexts $\mathbf{C}_{bn}[\,]$.

4. Closures and environment machines

The operational semantics defined in Section 3 are substitution-based, i.e., rely on the traditional meta-level substitution function $[N/n]B$. But the full-reducing Krivine machine is an environment-based machine that works with *closures* $M[\rho]$ consisting of a term M with an environment ρ that maps M's variables to corresponding bindings. Usually, the environment is a list of closures, such that de Bruijn indices act as lexical offsets (starting with 0) that point to the appropriate binding in the environment.

The $\lambda_{\widehat{\rho}}$ calculus of Biernacka and Danvy [5] extends the pure lambda calculus with definitions for closures C and environments ρ. This calculus is itself an extension of Curien's calculus of closures λ_ρ. Here is their respective syntax for terms, adapted from [5] to our own notation explained below.

$$\begin{array}{lcl@{\qquad}lcl}
 & \lambda_\rho & & & \lambda_{\widehat{\rho}} & \\
\mathsf{C} & ::= & \Lambda[\rho] & \mathsf{C} & ::= & \Lambda[\rho] \mid \mathsf{C}\cdot\mathsf{C} \\
\rho & ::= & \epsilon \mid \mathsf{C} : \rho & \rho & ::= & \epsilon \mid \mathsf{C} : \rho
\end{array}$$

In the $\lambda_{\widehat{\rho}}$ calculus we have *proper closures* $\Lambda[\rho]$ and *closure applications* $\mathsf{C}\cdot\mathsf{C}$. We use a left-associative explicit closure ap-

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

$$\frac{M \notin \text{WHNF} \quad M \rightarrow_{no} M'}{M\,N \rightarrow_{no} M'\,N}\ (\mu 1)$$

$$\frac{M \in \text{WHNF} \quad M \not\equiv \lambda.B \quad M \rightarrow_{no} M'}{M\,N \rightarrow_{no} M'\,N}\ (\mu 2)$$

$$\frac{M \in \text{NF} \quad M \not\equiv \lambda.B \quad N \rightarrow_{no} N'}{M\,N \rightarrow_{no} M\,N'}\ (\nu)$$

$$\frac{B \rightarrow_{no} B'}{\lambda.B \rightarrow_{no} \lambda.B'}\ (\xi)$$

$$\begin{array}{lcl@{\qquad}lcl} \text{WHNF} & ::= & \lambda.\Lambda & \text{NF} & ::= & \lambda.\text{NF} \\ & \mid & n\{\Lambda\}^* & & \mid & n\{\text{NF}\}^* \end{array}$$

$$\frac{}{n \Downarrow_{bn} n}\ (\textsc{Var}_{bn}) \qquad \frac{}{\lambda.B \Downarrow_{bn} \lambda.B}\ (\textsc{Lam}_{bn})$$

$$\frac{M \Downarrow_{bn} M' \quad M' \equiv \lambda.B \quad [N/0]B \Downarrow_{bn} B'}{M\,N \Downarrow_{bn} B'}\ (\textsc{Red}_{bn})$$

$$\frac{M \Downarrow_{bn} M' \quad M' \not\equiv \lambda.B}{M\,N \Downarrow_{bn} M'\,N}\ (\textsc{Neu}_{bn})$$

$$\frac{}{n \Downarrow_{no} n}\ (\textsc{Var}_{no}) \qquad \frac{B \Downarrow_{no} B'}{\lambda.B \Downarrow_{no} \lambda.B'}\ (\textsc{Lam}_{no})$$

$$\frac{M \Downarrow_{bn} M' \quad M' \equiv \lambda.B \quad [N/0]B \Downarrow_{no} B'}{M\,N \Downarrow_{no} B'}\ (\textsc{Red}_{no})$$

$$\frac{M \Downarrow_{bn} M' \quad M' \not\equiv \lambda.B \quad M' \Downarrow_{no} M'' \quad N \Downarrow_{no} N'}{M\,N \Downarrow_{no} M''\,N'}\ (\textsc{Neu}_{no})$$

Figure 2. Structural (left) and natural (right) operational semantics of normal order [17].

plication operator $\cdot$ which is elided in [5]. In both calculi, an environment ρ is either empty, which we denote by ϵ, or a list of colon-separated closures. In Curien's calculus the β-rule is $((\lambda.B)N)[\rho] \rightarrow B[N[\rho] : \rho]$ which pushes the substitution on the environment. A rule for variables is introduced $n[\rho] \rightarrow n^{\text{th}}(\rho)$ to deliver the nth binding in the environment. Both λ_ρ and $\lambda_{\widehat{\rho}}$ assume *closures without free variables*, i.e., the term is closed by the environment and a binding is always found. As noted in [5], small-step reduction relations cannot be expressed in λ_ρ and a natural solution is to extend λ_ρ with closure application together with an *ephemeral expansion* rule $(M\,N)[\rho] \rightarrow M[\rho] \cdot N[\rho]$ that distributes the environment by constructing an ephemeral closure application.[5] The β-rule now operates on closure applications: $(\lambda.B)[\rho] \cdot \mathsf{N} \rightarrow B[\mathsf{N} : \rho]$. (We use uppercase sans-serif letters M, N, etc, for closure terms to save us from contriving environment symbols. For instance, N above stands for some closure $N[\rho']$ with some environment ρ'.)

To illustrate, consider a closed term $(\lambda.0)N$. It is initialised to a closure $((\lambda.0)N)[\epsilon]$ and reduced as follows: $((\lambda.0)N)[\epsilon] \rightarrow (\lambda.0)[\epsilon] \cdot N[\epsilon] \rightarrow 0[N[\epsilon] : \epsilon] \rightarrow 0^{\text{th}}([N[\epsilon] : \epsilon]) \rightarrow N[\epsilon] \rightarrow \ldots$ The simulation of λ_ρ reductions in $\lambda_{\widehat{\rho}}$, which is easy to see, is proven in [5].

4.1 Call-by-name semantics and environment-based machine

The structural and context-based small-step operational semantics of call-by-name in $\lambda_{\widehat{\rho}}$ are shown in Figure 3 (adapted from [5] to our notation). Observe in the reduction semantics that closure application enables the definition of reduction contexts for closures. The redex can now occur in the operator side of a closure application. In [5], a reduction-based normaliser for this reduction semantics is implemented, and an environment-machine derived. The ephemeral expansion step is shortcut, getting rid of the closure application. The machine obtained is the call-by-name KAM machine [7].

The connection between $\lambda_{\widehat{\rho}}$ and λ is established by substitution function σ [5, p.9] (Figure 4) that forces all the delayed substitutions and simulates capture-avoiding substitution in λ. The function carries a *lexical adjustment* parameter k that is incremented when

[5] 'Ephemeral' in the sense that closure applications are shortcut when deriving big-step artefacts [5].

$$\frac{}{(\lambda.B)[\rho] \cdot \mathsf{N} \rightarrow_{\widehat{bn}} B[\mathsf{N} : \rho]}\ (\beta_{\widehat{\rho}}) \qquad \frac{\mathsf{M} \rightarrow_{\widehat{bn}} \mathsf{M}'}{\mathsf{M} \cdot \mathsf{N} \rightarrow_{\widehat{bn}} \mathsf{M}' \cdot \mathsf{N}}\ (\mu_{\widehat{\rho}})$$

$$\frac{}{n[\rho] \rightarrow_{\widehat{bn}} n^{\text{th}}(\rho)}\ (\textsc{Var}_{\widehat{\rho}})$$

$$\frac{}{(M\,N)[\rho] \rightarrow_{\widehat{bn}} M[\rho] \cdot N[\rho]}\ (\textsc{App}_{\widehat{\rho}})$$

$$\begin{array}{ll} \text{Red. context:} & \mathbf{C}_{\widehat{bn}}[\,] \quad ::= \quad [\,] \mid \mathbf{C}_{\widehat{bn}}[\,] \cdot \mathsf{C} \\ \text{Contraction:} & \mathbf{C}_{\widehat{bn}}[(M\,N)[\rho]] \ \rightarrow_{\widehat{bn}} \ \mathbf{C}_{\widehat{bn}}[M[\rho] \cdot N[\rho]] \\ & \mathbf{C}_{\widehat{bn}}[n[\rho]] \ \rightarrow_{\widehat{bn}} \ \mathbf{C}_{\widehat{bn}}[n^{\text{th}}(\rho)] \\ & \mathbf{C}_{\widehat{bn}}[(\lambda.B)[\rho] \cdot \mathsf{N}] \ \rightarrow_{\widehat{bn}} \ \mathbf{C}_{\widehat{bn}}[B[\mathsf{N} : \rho]] \end{array}$$

Figure 3. Structural and reduction semantics of call-by-name in $\lambda_{\widehat{\rho}}$ (adapted from [5]).

$$\begin{array}{rcl} \sigma(\mathsf{C},\mathbb{N}) & \rightarrow & \Lambda \\ \sigma(n[\rho],k) & = & \begin{cases} n & \text{if } n \leq k \\ \sigma((n-k)^{\text{th}}(\rho),0) & \text{if } n > k \end{cases} \\ \sigma((\lambda.B)[\rho],k) & = & \lambda.\sigma(B[\rho],k+1) \\ \sigma((M\,N)[\rho],k) & = & (\sigma(M[\rho],k))(\sigma(N[\rho],k)) \\ \sigma(\mathsf{M}\cdot\mathsf{N},k) & = & (\sigma(\mathsf{M},k))(\sigma(\mathsf{N},k)) \end{array}$$

Figure 4. Substitution function σ connects $\lambda_{\widehat{\rho}}$ and λ.

going under lambda (second clause). Integers $n \leq k$ stand for occurrences of formal parameters of abstractions that *have not* been applied to an operand. Integers $n > k$ are occurrences of formal parameters of abstractions that *have* been applied to an operand and thus have a binding in the environment (recall $\lambda_{\widehat{\rho}}$ assumes closures without free variables). For these variables the index is adjusted to $n - k$, and substitution is applied on the binding with the lexical adjustment reset to zero. The environment and the lexical adjustment are duplicated for application closures and closure applications (third and fourth clauses). The lexical adjustment discipline faithfully implements substitution for closures without free variables.

$$
\begin{array}{lcl}
 & \lambda_{\overline{\rho}} & \\
\mathsf{C} & ::= & \Lambda[\rho] \mid \overline{n} \mid \lfloor \Lambda, l \rfloor \\
\rho & ::= & \epsilon \mid \mathsf{C} : \rho
\end{array}
\qquad
\begin{array}{lcl}
S & ::= & \epsilon \\
 & \mid & \Lambda[\rho] : S \\
 & \mid & \lambda : S \\
 & \mid & \lfloor \Lambda, l \rfloor : S
\end{array}
$$

(1)	T	$\to$	$(T[\epsilon], \epsilon, 0)$
(2)	$((n+1)[\mathsf{C} : \rho], S, l)$	$\to$	$(n[\rho], S, l)$
(3)	$(0[\mathsf{C} : \rho], S, l)$	$\to$	(C, S, l)
(4)	$((M\,N)[\rho], S, l)$	$\to$	$(M[\rho], N[\rho] : S, l)$
(5)	$((\lambda.B)[\rho], N[\rho'] : S, l)$	$\to$	$(B[N[\rho'] : \rho], S, l)$
(6)	$((\lambda.B)[\rho], S, l)$	$\to$	$(B[\overline{l+1} : \rho], \lambda : S, l+1)$
(7)	$(\overline{n}, S, l)$	$\to$	$(\lfloor l - n, l \rfloor, S, l)$
(8)	$(\lfloor M, l \rfloor, N[\rho] : S, l')$	$\to$	$(N[\rho], \lfloor M, l \rfloor : S, l)$
(9)	$(\lfloor B, l \rfloor, \lambda : S, l')$	$\to$	$(\lfloor \lambda.B, l \rfloor, S, l')$
(10)	$(\lfloor N, l \rfloor, \lfloor M, l' \rfloor : S, l'')$	$\to$	$(\lfloor M\,N, l' \rfloor, S, l'')$
(11)	$(\lfloor T, l \rfloor, \epsilon, l')$	$\to$	T

Figure 5. Crégut's full-reducing KN with its calculus of closures $\lambda_{\overline{\rho}}$ and continuation stack S (adapted from [8]).

5. Cregut's full-reducing Krivine machine

The full-reducing machine KN, shown in Figure 5 (adapted from [8] to our notation), is the target of our derivation. KN is a first-order transition function which operates on a triple consisting of a closure, a continuation stack, and a de Bruijn level l (lambda level for short). Closures C now include de Bruijn indices (n coming from Λ) and lambda levels for encoding the nesting of formal parameters that are pushed on the environment (written $\overline{n}$). Lambda levels realise what we shall refer to as the *parameters-as-levels* technique. Closures also include an embedding of ground terms with a level $\lfloor \Lambda, l \rfloor$ whose meaning is explained below. The syntax suggests an implicit calculus which we name $\lambda_{\overline{\rho}}$. The continuation stack S can be empty (same symbol as empty environments), store operands, store the control character λ which indicates that the current scope is under an abstraction, or store embedded ground terms.

The execution example in Figure 6 shows the rules in Figure 5 at work. We explain each rule in detail. The first init rule constructs a triple for a *closed* term T. The next two rules are for looking up variables by peeling off the environment while decrementing indices. The binding at the top of the environment is delivered when the index is 0. The 4th rule pushes on the stack the operand in closure form. The 5th rule embodies a contraction: the operand closure is retrieved from the stack and pushed on the abstraction body's environment. The 6th rule is for unapplied abstractions (there is no closure operand on the top of the stack). The control character λ is pushed on the stack to signal that the machine is going under lambda, and the level l is incremented and also pushed on the abstraction body's environment. The level pushed on the environment $\overline{l+1}$ encodes the nesting of the abstraction's formal parameter. In the 7th rule, the appropriate de Bruijn index is computed by subtracting $\overline{n}$ from the level in the current scope, and the computed index is embedded in a ground term with the current level. The subtraction is reminiscent of the lexical adjustment technique in σ (Section 4) although in KN level l is not reset to zero and no adjustment is needed when looking up in the environment, for it grows as formal parameters are pushed onto it. This guarantees an *index alignment* property, i.e., every index points to a binding on the environment.

The remaining rules are for neutral terms and illuminate the reason for embedded ground terms with levels. We shall explain them with an example. Consider the abstraction $\lambda.0(\lambda.M)N$ which has a neutral term as body. Subterm N has to be reduced with the same level as the head variable 0. The head variable is embedded in a ground term with its level (7th rule, already explained), and the embedding pushed on the stack (8th rule). The machine increments the level when going under lambda in $\lambda.M$ (6th rule, already explained), but it does not decrement the level when scoping out of it (9th rule). However, the appropriate level for N can be recovered from the ground term on the top of the stack (10th rule).

$$
\begin{array}{ll}
 & \lambda.0((\lambda.0)0) \\
 & \{\text{init}\} \\
\to & ((\lambda.0((\lambda.0)0))[\epsilon], \epsilon, 0) \\
 & \{\text{unapplied abs., push } \overline{1} \text{ onto env. and } \lambda \text{ onto stack}\} \\
\to & (0((\lambda.0)0)[\overline{1} : \epsilon], \lambda : \epsilon, 1) \\
 & \{\text{application, push operand onto stack}\} \\
\to & (0[\overline{1} : \epsilon], ((\lambda.0)0)[\overline{1} : \epsilon] : \lambda : \epsilon, 1) \\
 & \{\text{retrieve top binding}\} \\
\to & (\overline{1}, ((\lambda.0)0)[\overline{1} : \epsilon] : \lambda : \epsilon, 1) \\
 & \{\text{embed abs. index } \overline{n} - l \text{ into ground term with level}\} \\
\to & (\lfloor 0, 1 \rfloor, ((\lambda.0)0)[\overline{1} : \epsilon] : \lambda : \epsilon, 1) \\
 & \{\text{reduce operand of ground term, store ground in the stack}\} \\
\to & (((\lambda.0)0)[\overline{1} : \epsilon], \lfloor 0, 1 \rfloor : \lambda : \epsilon, 1) \\
 & \{\text{application, push operand onto stack}\} \\
\to & ((\lambda.0)[\overline{1} : \epsilon], 0[\overline{1} : \epsilon] : \lfloor 0, 1 \rfloor : \lambda : \epsilon, 1) \\
 & \{\beta\text{-redex, push operand in body's env.}\} \\
\to & (0[0[\overline{1} : \epsilon] : \overline{1} : \epsilon], \lfloor 0, 1 \rfloor : \lambda : \epsilon, 1) \\
 & \{\text{retrieve top binding}\} \\
\to & (0[\overline{1} : \epsilon], \lfloor 0, 1 \rfloor : \lambda : \epsilon, 1) \\
 & \{\text{retrieve top binding}\} \\
\to & (\overline{1}, \lfloor 0, 1 \rfloor : \lambda : \epsilon, 1) \\
 & \{\text{embed abs. index } \overline{n} - l \text{ into ground term with level}\} \\
\to & (\lfloor 0, 1 \rfloor, \lfloor 0, 1 \rfloor : \lambda : \epsilon, 1) \\
 & \{\text{accumulate neutral in nf into ground term}\} \\
\to & (\lfloor 0\,0, 1 \rfloor, \lambda : \epsilon, 1) \\
 & \{\text{out-of-lambda}\} \\
\to & (\lfloor \lambda.0\,0, 1 \rfloor, \epsilon, 1) \\
 & \{\text{done}\} \\
\to & \lambda.0\,0
\end{array}
$$

Figure 6. Execution example of KN on term $\lambda.0((\lambda.0)0)$.

6. Introducing the calculus of closures $\lambda_{\widetilde{\rho}}$

We introduce the $\lambda_{\widetilde{\rho}}$ calculus as the natural extension of $\lambda_{\widehat{\rho}}$ that subsumes $\lambda_{\overline{\rho}}$.

$$
\begin{array}{lcl}
 & \lambda_{\widetilde{\rho}} & \\
\mathsf{C} & ::= & \Lambda[\rho] \mid \overline{n} \mid \lfloor n \rfloor \mid \lambda\!\!\lambda.\mathsf{C} \mid \mathsf{C} \cdot \mathsf{C} \\
\rho & ::= & \epsilon \mid \mathsf{C} : \rho
\end{array}
$$

The calculus only adds two ephemeral constructors which are required for full-reduction, namely, *absolute indices* $\lfloor n \rfloor$ and *closure abstractions* $\lambda\!\!\lambda.\mathsf{C}$. Absolute indices are de Bruijn indices that are not relative to an environment. Absolute indices are different from closures $n[\epsilon]$. The latter stand for free variables (as well as $n[\rho]$ with $n > |\rho|$) and, as we will see below, they trigger index calculations. The reader can deduce from the previous sentence that $\lambda_{\widetilde{\rho}}$ *assumes closures with free variables* (open terms). Absolute indices are required to represent neutral closures which are closure applications of an absolute index to other closures (for an advance, see the irreducible forms at the bottom of Figure 9). Closure abstractions are required to represent closures where the redex may occur under lambda. There is an obvious isomorphism between Λ and all the ephemeral closure constructions (hereafter 'ephemeral closures'), gathered in $\mathsf{E} ::= \lfloor n \rfloor \mid \lambda\!\!\lambda.\mathsf{E} \mid \mathsf{E} \cdot \mathsf{E}$. As was the case

$$
\begin{array}{rcl}
\sigma_{\mathsf{C}}(\mathsf{C},\mathbb{N}) & \rightarrow & \mathsf{E} \\
\sigma_{\mathsf{C}}(\lfloor n \rfloor, l) & = & \lfloor n \rfloor \\
\sigma_{\mathsf{C}}(n[\rho], l) & = & \begin{cases} \sigma_{\mathsf{C}}(n^{\mathrm{th}}(\rho), l) & \text{if } n < |\rho| \\ \lfloor n - (|\rho| - l) \rfloor & \text{if } n \geq |\rho| \end{cases} \\
\sigma_{\mathsf{C}}((\lambda.B)[\rho], l) & = & \sigma_{\mathsf{C}}(\lambda\!\!\lambda.B[\overline{l+1} : \rho], l) \\
\sigma_{\mathsf{C}}(\lambda\!\!\lambda.\mathsf{B}, l) & = & \lambda\!\!\lambda.\sigma_{\mathsf{C}}(\mathsf{B}, l+1) \\
\sigma_{\mathsf{C}}(\overline{n}, l) & = & \lfloor l - n \rfloor \\
\sigma_{\mathsf{C}}((M\,N)[\rho], l) & = & \sigma_{\mathsf{C}}(M[\rho] \cdot N[\rho], l) \\
\sigma_{\mathsf{C}}(\mathsf{M} \cdot \mathsf{N}, l) & = & \sigma_{\mathsf{C}}(\mathsf{M}, l) \cdot \sigma_{\mathsf{C}}(\mathsf{N}, l)
\end{array}
$$

Figure 7. Substitution function σ_{C}.

$$
\begin{array}{rcl}
h(\mathsf{C}) & \rightarrow & \mathbb{N} \\
h(n[\rho]) & = & \begin{cases} h(n^{\mathrm{th}}(\rho)) & \text{if } n < |\rho| \\ 0 & \text{if } n \geq |\rho| \end{cases} \\
h((\lambda.B)[\rho]) & = & 1 + h(B[\overline{n} : \rho]) \\
h((M\,N)[\rho]) & = & \max\{h(M[\rho]), h(N[\rho])\} \\
h(\overline{n}) & = & 0 \\
h(\lfloor n \rfloor) & = & 0 \\
h(\lambda\!\!\lambda.\mathsf{B}) & = & 1 + h(\mathsf{B}) \\
h(\mathsf{M} \cdot \mathsf{N}) & = & \max\{h(\mathsf{M}), h(\mathsf{N})\}
\end{array}
$$

Figure 8. Height of a closure.

with $\lambda_{\widehat{\rho}}$ (Section 4), the ephemeral closures of $\lambda_{\widetilde{\rho}}$ are required to define reduction contexts for closures (Section 6.1).

The connection between $\lambda_{\widetilde{\rho}}$ and λ is established by substitution function σ_{C} (Figure 7, top) which is the analogous of function σ in $\lambda_{\widehat{\rho}}$ and simulates capture-avoiding substitution in λ. Function σ_{C} now carries a lambda level parameter l and enforces index alignment like KN (Section 5). Observe that in the 3rd clause, σ_{C} increments the level encoding the scope of the formal parameter that is pushed on the environment, namely $\overline{l+1}$, but does not increment the level parameter l. It is in the 4th rule, when going under closure abstraction, that the lambda level l is incremented but the environment is not touched. The remaining clauses are unsurprising. Absolute indices are simply returned (1st clause), bound variables are looked up in the environment (2nd clause, case $n < |\rho|$), free variables are given their absolute indices (2nd clause, case $n \geq |\rho|$) which are calculated by subtracting to the current index n the number of *proper bindings* in the environment, i.e., bindings other than levels encoding the nesting of formal parameters. This number coincides with the length of the environment $|\rho|$ minus the current lambda level l. Finally, σ_{C} calculates the absolute index of formal parameters retrieved from the environment (5th clause), lifts application closures to closure applications (6th clause), and distributes over closure applications (7th clause).

Function σ_{C} simulates capture-avoiding substitution in λ, that is, $\sigma_{\mathsf{C}}(B[N[\rho] : \rho'], l) \equiv \sigma_{\mathsf{C}}(([\sigma_{\mathsf{C}}(N[\rho], l)/0]B)[\rho'], l)$, provided that $N[\rho]$ is a proper closure. This simulation property is proven by induction on the height of $B[N[\rho] : \rho']$ which is calculated by function h shown in Figure 8. As we shall see in the next section, the structural operational semantics of normal order will guarantee that $N[\rho]$ is always a proper closure.

6.1 Structural operational semantics of normal order in $\lambda_{\widetilde{\rho}}$

Figure 9 shows the structural operational semantics of normal order in $\lambda_{\widetilde{\rho}}$ which we have obtained from the structural version in λ (Figure 2) by adding ephemerals and KN's parameters-as-levels (Section 5). The lambda level l has to be carried along and thus $\rightarrow_{\widetilde{no}}$ operates on pairs $\langle \mathsf{C}, \mathbb{N} \rangle$ rather than just closures. The rules on the left of Figure 9 are notions of reduction for the new constructs and come naturally from σ_{C}. $\mathrm{V{\scriptsize AR}}_{\widetilde{\rho}}$ is the rule for bound variables, $\mathrm{A{\scriptsize PP}}_{\widetilde{\rho}}$ for lifting to closure application, $\mathrm{P{\scriptsize AR}}_{\widetilde{\rho}}$ for formal parameters, $\mathrm{F{\scriptsize RE}}_{\widetilde{\rho}}$ for free variables, and $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$ for lifting to closure abstraction where the formal parameter (the incremented lambda level) is pushed on the environment. The first rule on the right $(\beta_{\widetilde{\rho}})$ contracts $\beta_{\widetilde{\rho}}$-redices $(\lambda\!\!\lambda.B[\overline{n} : \rho]) \cdot \mathsf{N}$, where the formal parameter $\overline{n}$ that was pushed on the top of the environment by an immediately preceding ephemeral expansion $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$ is discarded and replaced by the operand N. The other compatibility rules $(\mu 1_{\widetilde{\rho}})$, $(\mu 2_{\widetilde{\rho}})$, $(\nu_{\widetilde{\rho}})$, and $(\xi_{\widetilde{\rho}})$ are obtained by adapting to closure-level pairs the corresponding rules in Figure 2. A pair's lambda level is incremented in $(\xi_{\widetilde{\rho}})$, for it 'goes under closure abstraction', leaving B's environment untouched.

Hybridisation can be observed by noticing that rules $\mathrm{V{\scriptsize AR}}_{\widetilde{\rho}}$, $\mathrm{A{\scriptsize PP}}_{\widetilde{\rho}}$, $\mathrm{F{\scriptsize RE}}_{\widetilde{\rho}}$, $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$, $(\beta_{\widetilde{\rho}})$ and $(\mu 1_{\widetilde{\rho}})$ define call-by-name in $\lambda_{\widetilde{\rho}}$, which coincides with call-by-name in $\lambda_{\widehat{\rho}}$ (Section 4.1, Figure 3), save for the addition of $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$ and $\mathrm{F{\scriptsize RE}}_{\widetilde{\rho}}$, and for the omission in $(\mu_{\widehat{\rho}})$ of the premiss $\mathsf{M} \notin \mathrm{WHNF}_{\mathsf{C}}$ which is present in $(\mu 1_{\widetilde{\rho}})$. $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$ is the immediately preceding ephemeral expansion required for $(\beta_{\widetilde{\rho}})$, and $\mathrm{F{\scriptsize RE}}_{\widetilde{\rho}}$ is required for free variables. Finally, the premiss $\mathsf{M} \notin \mathrm{WHNF}_{\mathsf{C}}$ is not present in $(\mu_{\widehat{\rho}})$ because that rule applies only when M is not in whnf. The premiss is required in $(\mu 1_{\widetilde{\rho}})$ to control the rule's applicability as part of a larger set of rules that specify the whole hybrid strategy.

Observe that derivations are *balanced*, i.e., a pair's lambda level remains constant in the left- and right-hand sides of judgements in derivation trees. This makes reasoning by structural induction easier and guarantees that the lambda level of a proper closure $\Lambda[\rho]$ is compatible with its environment, i.e., the lambda level matches the level of the bindings in ρ. Consequently, lambda levels don't have to be carried along with closures in environments and, unlike KN, levels don't have to be recovered from the environment when reducing operands of neutral closures. This suggest an optimisation of KN that we discuss in Section 9.4.

The syntax for closure whnfs (hereafter whnf$_{\mathsf{C}}$) and closure nfs (hereafter nf$_{\mathsf{C}}$) is shown at the bottom of Figure 9. The nf$_{\mathsf{C}}$s are included in ephemeral closures E but are not included in whnf$_{\mathsf{C}}$s because abstraction bodies in whnf$_{\mathsf{C}}$ are proper closures with delayed substitutions in their environments. These environments may be enlarged by the combination of $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$ and $(\beta_{\widetilde{\rho}})$, and their closures can only be removed when demanded by $\mathrm{V{\scriptsize AR}}_{\widetilde{\rho}}$.

As discussed in Section 6, the substitution function σ_{C} (Figure 7) connects $\lambda_{\widetilde{\rho}}$ with λ. Moreover, the connection can be established at the step-by-step level between $\rightarrow_{\widetilde{no}}$ in $\lambda_{\widetilde{\rho}}$ and $\rightarrow_{no}$ in λ, as illustrated by the following diagram.

$$\begin{tikzcd}
\langle \mathsf{M}, 0 \rangle \arrow[r, "*", "\widetilde{\rho}"'] & \langle \mathsf{M}', 0 \rangle \arrow[r, "\beta_{\widetilde{\rho}}"'] & \langle \mathsf{M}_1, 0 \rangle \arrow[r, "*", "\widetilde{\rho}"'] \arrow[d, "\sigma_{\mathsf{C}}"'] & \langle \mathsf{M}'_1, 0 \rangle \arrow[r, "\beta_{\widetilde{\rho}}"'] & \langle \mathsf{M}_2, 0 \rangle \arrow[d, "\sigma_{\mathsf{C}}"'] \\
M \arrow[u, hook] \arrow[rr, "\beta"'] & & M_1 \arrow[rr, "\beta"'] & & M_2
\end{tikzcd}$$

The input term M is injected into the closure $M[\epsilon]$ abbreviated M. The closures M_i map via σ_{C} to ground terms M_i which are the result of step-by-step normal order in λ. The reduction relation $\rightarrow_{\widetilde{\rho}}$ is that induced by all the rules in Figure 10 except $(\beta_{\widetilde{\rho}})$. The reduction relation $\rightarrow_{\beta_{\widetilde{\rho}}}$ is that induced by all the rules on the right column. *Due to the compatibility rules*, which are exactly those taken from λ, the relation $\rightarrow_{\widetilde{\rho}}$ ephemerally expands $\langle \mathsf{M}, 0 \rangle$ using notions of reduction $\mathrm{A{\scriptsize PP}}_{\widetilde{\rho}}$ and $\mathrm{L{\scriptsize AM}}_{\widetilde{\rho}}$ until finding either the leftmost $\beta_{\widetilde{\rho}}$-redex R or an index closure $n[\rho]$ in head position. In the first case, $\sigma_{\mathsf{C}}(\mathsf{R}, l)$ (where l is R's lambda level) is the leftmost β-redex in M. In the second case, n is either bound to a proper closure, or to a formal parameter $\overline{m}$, or is out of bounds. The binding for n is retrieved ($\mathrm{V{\scriptsize AR}}_{\widetilde{\rho}}$), or its proper absolute index calculated ($\mathrm{P{\scriptsize AR}}_{\widetilde{\rho}}$, $\mathrm{F{\scriptsize RE}}_{\widetilde{\rho}}$), and $\rightarrow_{\widetilde{\rho}}$ continues looking for the leftmost $\beta_{\widetilde{\rho}}$-redex.

$$\frac{n < |\rho|}{\langle n[\rho], l\rangle \to_{\widetilde{no}} \langle n^{\mathrm{th}}(\rho), l\rangle}\ (\textsc{Var}_{\widetilde{\rho}})
\qquad
\frac{}{\langle((\lambda\!\!\lambda.B[\overline{n}:\rho])\cdot \mathsf{N}, l\rangle \to_{\widetilde{no}} \langle B[\mathsf{N}:\rho], l\rangle}\ (\beta_{\widetilde{\rho}})$$

$$\frac{}{\langle (M\,N)[\rho], l\rangle \to_{\widetilde{no}} \langle M[\rho]\cdot N[\rho], l\rangle}\ (\textsc{App}_{\widetilde{\rho}})
\qquad
\frac{\mathsf{M}\notin \mathrm{WHNF}_{\mathsf{C}} \quad \langle \mathsf{M}, l\rangle \to_{\widetilde{no}} \langle \mathsf{M}', l\rangle}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \to_{\widetilde{no}} \langle \mathsf{M}'\cdot\mathsf{N}, l\rangle}\ (\mu1_{\widetilde{\rho}})$$

$$\frac{}{\langle \overline{n}, l\rangle \to_{\widetilde{no}} \langle \lfloor l-n\rfloor, l\rangle}\ (\textsc{Par}_{\widetilde{\rho}})
\qquad
\frac{\mathsf{M}\in \mathrm{WHNF}_{\mathsf{C}} \quad \mathsf{M}\not\equiv \lambda\!\!\lambda.\mathsf{B} \quad \langle \mathsf{M}, l\rangle \to_{\widetilde{no}} \langle \mathsf{M}', l\rangle}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \to_{\widetilde{no}} \langle \mathsf{M}'\cdot\mathsf{N}, l\rangle}\ (\mu2_{\widetilde{\rho}})$$

$$\frac{n \geq |\rho|}{\langle n[\rho], l\rangle \to_{\widetilde{no}} \langle \lfloor n-(|\rho|-l)\rfloor, l\rangle}\ (\textsc{Fre}_{\widetilde{\rho}})
\qquad
\frac{\mathsf{M}\in \mathrm{NF}_{\mathsf{C}} \quad \mathsf{M}\not\equiv \lambda\!\!\lambda.\mathsf{B} \quad \langle \mathsf{N}, l\rangle \to_{\widetilde{no}} \langle \mathsf{N}', l\rangle}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \to_{\widetilde{no}} \langle \mathsf{M}\cdot\mathsf{N}', l\rangle}\ (\nu_{\widetilde{\rho}})$$

$$\frac{}{\langle (\lambda.B)[\rho], l\rangle \to_{\widetilde{no}} \langle \lambda\!\!\lambda.B[\overline{l+1}:\rho], l\rangle}\ (\textsc{Lam}_{\widetilde{\rho}})
\qquad
\frac{\langle \mathsf{B}, l+1\rangle \to_{\widetilde{no}} \langle \mathsf{B}', l+1\rangle}{\langle \lambda\!\!\lambda.\mathsf{B}, l\rangle \to_{\widetilde{no}} \langle \lambda\!\!\lambda.\mathsf{B}', l\rangle}\ (\xi_{\widetilde{\rho}})$$

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

Figure 9. Parameters-as-levels and closure-converted structural operational semantics of normal order in $\lambda_{\widetilde{\rho}}$.

The notions of reduction $\textsc{Var}_{\widetilde{\rho}}$, $\textsc{Par}_{\widetilde{\rho}}$, and $\textsc{Fre}_{\widetilde{\rho}}$ merely implement substitution on demand and do not interfere with $(\beta_{\widetilde{\rho}})$.

The step-by-step connection rests on the property that σ_{C} simulates capture-avoiding substitution in λ (Section 6) and that normal order guarantees that bindings on environments are always proper closures or formal parameters. Since all the notions of reduction but $(\beta_{\widetilde{\rho}})$ come from σ_{C}, and since σ_{C} and $\to_{\widetilde{\rho}}$ do not go under environments, then σ_{C} commutes with $\to_{\widetilde{\rho}}$, i.e., given $\langle P, 0\rangle \to_{\widetilde{\rho}} \langle Q, 0\rangle$ and $\langle R, 0\rangle \to_{\widetilde{\rho}} \langle S, 0\rangle$, it is the case that $\sigma_{\mathsf{C}}(P,0) \equiv \sigma_{\mathsf{C}}(R,0)$ iff $\sigma_{\mathsf{C}}(Q,0) \equiv \sigma_{\mathsf{C}}(S,0)$.

7. From structural operational semantics to reduction-free normaliser

7.1 From structural to reduction semantics

The search functions `search_whnf` and `search_nf` implement the compatibility rules of the structural operational semantics of normal order in $\lambda_{\widetilde{\rho}}$ (Figure 9). The search functions deliver for an input term the (normal order or call-by-name) redex subterm to be contracted or the input term back if the input term is irreducible. The entry function `search` invokes `search_nf`. (From now on we omit for brevity the entry functions of all our semantics.) Function `search_nf` searches for a nf_{C} or for the next redex to be contracted. It relies on `search_whnf` to check if operators in applications are in $\mathrm{whnf}_{\mathsf{C}}$. Function `search_whnf` searches for a $\mathrm{whnf}_{\mathsf{C}}$ or for the next redex in the call-by-name subreduction to be contracted. The use of two functions explicitly reflects the inclusion of the subsidiary in the hybrid whereas an alternative equivalent implementation using a single search function with a boolean check for $\mathrm{whnf}_{\mathsf{C}}$-ness would only reflect it implicitly. The derivation tree above a second premiss of $(\mu1_{\widetilde{\rho}})$ will only contain call-by-name rules because $(\mu2_{\widetilde{\rho}})$, $(\nu_{\widetilde{\rho}})$ and $(\xi_{\widetilde{\rho}})$ are only applicable when the operator is in $\mathrm{whnf}_{\mathsf{C}}$ or in nf_{C}.

We apply standard derivation steps (CPS transformation, simplification, defunctionalisation, decomposition) and obtain three decomposition functions `decompose_whnf`, `decompose_nf`, and `decompose_cont` (the latter the continuation dispatcher that inevitably pops up). By adding the necessary `contract`, `recompose`, and trampoline `iterate` functions [12] we obtain the trampolined reduction-based normaliser `normalise` that implements the reduction semantics of Figure 10. This reduction-based normaliser is the starting point of the derivation path shown in Figure 1.

$$\begin{array}{lrcl}
\text{Red. context:} & \mathbf{C}^{l}_{\widetilde{no}}[\,] & ::= & [\,]^{l} \mid \mathbf{C}^{l}_{\widetilde{bn}}[\,]\cdot\mathsf{C} \mid \lambda\!\!\lambda.\mathbf{C}^{l+1}_{\widetilde{no}}[\,] \mid \mathbf{C}^{l}_{\widetilde{ne}}[\,] \\
 & \mathbf{C}^{l}_{\widetilde{bn}}[\,] & ::= & [\,]^{l} \mid \mathbf{C}^{l}_{\widetilde{bn}}[\,]\cdot\mathsf{C} \\
 & \mathbf{C}^{l}_{\widetilde{ne}}[\,] & ::= & \lfloor n\rfloor\{\cdot\mathrm{NF}_{\mathsf{C}}\}^{*}\cdot\mathbf{C}^{l}_{\widetilde{no}}[\,] \mid \mathbf{C}^{l}_{\widetilde{ne}}[\,]\cdot\mathsf{C}
\end{array}$$

$$\begin{array}{ll}
\text{Contraction:} & \langle \mathbf{C}^{0}_{\widetilde{no}}[n[\rho]], l\rangle \ \to_{\widetilde{no}} \\
 & \qquad\begin{cases} \langle \mathbf{C}^{0}_{\widetilde{no}}[n^{\mathrm{th}}(\rho)], l\rangle & \text{if } n < |\rho| \\ \langle \mathbf{C}^{0}_{\widetilde{no}}[\lfloor n-(|\rho|-l)\rfloor], l\rangle & \text{if } n \geq |\rho| \end{cases} \\
 & \langle \mathbf{C}^{0}_{\widetilde{no}}[(M\,N)[\rho]], l\rangle \ \to_{\widetilde{no}} \ \langle \mathbf{C}^{0}_{\widetilde{no}}[M[\rho]\cdot N[\rho]], l\rangle \\
 & \langle \mathbf{C}^{0}_{\widetilde{no}}[\overline{n}], l\rangle \ \to_{\widetilde{no}} \ \langle \mathbf{C}^{0}_{\widetilde{no}}[\lfloor l-n\rfloor], l\rangle \\
 & \langle \mathbf{C}^{0}_{\widetilde{no}}[(\lambda.B)[\rho]], l\rangle \ \to_{\widetilde{no}} \ \langle \mathbf{C}^{0}_{\widetilde{no}}[\lambda\!\!\lambda.B[\overline{l+1}:\rho]], l\rangle \\
 & \langle \mathbf{C}^{0}_{\widetilde{no}}[(\lambda\!\!\lambda.B[\overline{n}:\rho])\cdot\mathsf{N}], l\rangle \ \to_{\widetilde{no}} \ \langle \mathbf{C}^{0}_{\widetilde{no}}[B[\mathsf{N}:\rho]], l\rangle
\end{array}$$

Figure 10. Reduction semantics for normal order in $\lambda_{\widetilde{\rho}}$.

The reduction relation $\to_{\widetilde{no}}$ is defined on pairs $\langle \mathbf{C}^{0}_{\widetilde{no}}[\mathsf{R}], l\rangle$ consisting of a top-level context (with the closure redex R within the hole) and the lambda level l at which the redex occurs. Reduction contexts keep track of the lambda level (superscripts), starting with level zero (top scope) and incrementing it when entering a $\lambda\!\!\lambda$ scope. Thus, the lambda level l in $\langle \mathbf{C}^{0}_{\widetilde{no}}[\mathsf{R}], l\rangle$ is such that $\mathbf{C}^{0}_{\widetilde{no}}[\mathsf{R}] \equiv \ldots[\mathsf{R}]^{l}\ldots$

Observe that the reduction semantics of call-by-name in $\lambda_{\widetilde{\rho}}$ (Section 4.1) coincides with the reduction semantics defined by considering $\mathbf{C}^{0}_{\widetilde{bn}}[\,]$ to be the top-level context, by removing the contraction cases for free variables and for formal parameters, and by shortcutting closure abstractions. Free variables are not considered in $\lambda_{\widetilde{\rho}}$ which assumes closures without free variables. Formal parameters $\overline{n}$ are not considered in $\lambda_{\widetilde{\rho}}$ because call-by-name does not go under lambda. Finally, the last contraction case for call-by-name in Figure 3 is obtained by shortcutting the last two contraction cases in Figure 10. The lambda level is never incremented by $\mathbf{C}^{0}_{\widetilde{bn}}[\,]$ and can be dropped.

7.2 Syntactic correspondence

The correspondence between the reduction semantics of Figure 10 and the environment-based eval/apply abstract machine of Fig-

$$
\begin{array}{rcl}
S & ::= & \mathbf{C}_0 \mid \mathbf{C}_1(\mathsf{C}) : S \mid \mathbf{C}_2 : S \mid \mathbf{C}_3(\mathsf{C}) \\
 & \mid & \mathbf{C}_4(\mathsf{C}) : S \mid \mathbf{C}_5(\mathsf{C}) : S
\end{array}
$$

$$
\begin{array}{|rrcl|}
\hline
 & T & \to & (T[\epsilon], \mathbf{C}_0, 0)_n \\
\hline
(\text{if } n < |\rho|) & (n[\rho], S, l)_w & \to & (n^{\text{th}}(\rho), S, l)_w \\
(\text{if } n \geq |\rho|) & (n[\rho], S, l)_w & \to & (\lfloor n - (|\rho| - l) \rfloor, S, l)_a \\
 & (\overline{n}, S, l)_w & \to & (\lfloor l - n \rfloor, S, l)_a \\
 & (\lfloor n \rfloor, S, l)_w & \to & (\lfloor n \rfloor, S, l)_a \\
 & ((\lambda.B)[\rho], S, l)_w & \to & (\boldsymbol{\lambda}.B[\overline{l+1} : \rho], S, l)_a \\
 & ((M\,N)[\rho], S, l)_w & \to & (M[\rho] \cdot N[\rho], S, l)_w \\
 & (\mathsf{M} \cdot \mathsf{N}, S, l)_w & \to & (\mathsf{M}, \mathbf{C}_1(\mathsf{N}) : S, l)_w \\
\hline
(\text{if } n < |\rho|) & (n[\rho], S, l)_n & \to & (n^{\text{th}}(\rho), S, l)_n \\
(\text{if } n \geq |\rho|) & (n[\rho], S, l)_n & \to & (\lfloor n - (|\rho| - l) \rfloor, S, l)_a \\
 & (\overline{n}, S, l)_n & \to & (\lfloor l - n \rfloor, S, l)_a \\
 & (\lfloor n \rfloor, S, l)_n & \to & (\lfloor n \rfloor, S, l)_a \\
 & ((\lambda.B)[\rho], S, l)_n & \to & (B[\overline{l+1} : \rho], \mathbf{C}_2 : S, l+1)_n \\
 & ((M\,N)[\rho], S, l)_n & \to & (M[\rho] \cdot N[\rho] : S, l)_n \\
 & (\mathsf{M} \cdot \mathsf{N}, S, l)_n & \to & (\mathsf{M}, \mathbf{C}_3(\mathsf{N}) : S, l)_w \\
\hline
 & (\boldsymbol{\lambda}.B[\overline{n} : \rho], \mathbf{C}_1(\mathsf{N}) : S, l)_a & \to & (B[\mathsf{N} : \rho], S, l)_w \\
 & (\mathsf{M}, \mathbf{C}_1(\mathsf{N}) : S, l)_a & \to & (\mathsf{M} \cdot \mathsf{N}, S, l)_a \\
 & (\mathsf{B}, \mathbf{C}_2 : S, l)_a & \to & (\boldsymbol{\lambda}.\mathsf{B}, S, l-1)_a \\
 & (\boldsymbol{\lambda}.B[\overline{n} : \rho], \mathbf{C}_3(\mathsf{N}) : S, l)_a & \to & (B[\mathsf{N} : \rho], S, l)_n \\
 & (\mathsf{M}, \mathbf{C}_3(\mathsf{N}) : S, l)_a & \to & (\mathsf{M}, \mathbf{C}_4(\mathsf{N}) : S, l)_n \\
 & (\mathsf{M}, \mathbf{C}_4(\mathsf{N}) : S, l)_a & \to & (\mathsf{N}, \mathbf{C}_5(\mathsf{M}) : S, l)_n \\
 & (\mathsf{N}, \mathbf{C}_5(\mathsf{M}) : S, l)_a & \to & (\mathsf{M} \cdot \mathsf{N}, S, l)_a \\
\hline
 & (\mathsf{E}, \mathbf{C}_0, l)_a & \to & \mathsf{E} \\
\hline
\end{array}
$$

Figure 11. Normal order environment-based eval/apply machine with continuation stack S.

ure 11 is obtained by performing refocusing, inlining-of-iterate-function, and transition compression steps. We apply the refined inlining-of-iterate-function step in [17, 18] that exploits the shape invariant of the continuation stack, and enables the derivation of the machine with the *shallow inspection* property. The machine has three states, normalise to whnf, normalise to nf, and apply. The configurations for each state are type-annotated by subscripts w, n, and a respectively. The machine decrements the level l (6th rule from the bottom) when leaving a $\boldsymbol{\lambda}$ scope, thus mirroring rule ($\xi_{\widetilde{\rho}}$) in Figure 9. This machine is the closure-converted version of the substitution-based, eval/apply machine in [17]. The functions `normalise4_whnf`, `normalise4_nf`, and `normalise4_cont` in the code make up the big-step tail-recursive implementation of the machine.

7.3 Functional correspondence

We apply refunctionalisation and inverse CPS to the big-step tail-recursive implementation of the machine in Figure 11 and obtain the reduction-free normalisers `normalise6_whnf` and `normalise6_nf` that implement the big-step natural semantics in Figure 12. Notice that levels are unneeded and hence dropped from *final* results.

8. Towards shortcutting ephemeral expansion

To shortcut ephemeral expansion in the natural semantics of Figure 12 we have to introduce a *preponing* step which we explain in this section. Consider these two examples of call-by-name normalisation that deliver a whnf$_{\mathsf{C}}$:

$$
\begin{array}{rcll}
\langle (\lambda.B)[\rho], 0 \rangle & \Downarrow_{\widetilde{bn}} & \boldsymbol{\lambda}.B[\overline{1} : \rho] & (1) \\
\langle (0(\lambda.M))[\rho], 0 \rangle & \Downarrow_{\widetilde{bn}} & \lfloor 0 \rfloor \cdot ((\lambda.M)[\rho]) & (2)
\end{array}
$$

Abstraction bodies in whnf$_{\mathsf{C}}$ are not ephemeral closures by the definition of whnf$_{\mathsf{C}}$ (Figure 9, bottom), and neither are operands in neutral terms by the definition of $\Downarrow_{\widetilde{bn}}$. In the examples above, neither $B[\overline{1} : \rho]$ nor $(\lambda.M)[\rho]$ can be embedded into a ground term without further processing, e.g., expanding and substituting like σ_{C}. Consequently, the ephemeral expansion steps cannot be coalesced in call-by-name because ephemeral constructors must appear in the rules of $\Downarrow_{\widetilde{bn}}$ and in the definition of whnf$_{\mathsf{C}}$. More precisely, $\Downarrow_{\widetilde{bn}}$ is used in the premiss $\langle \mathsf{M}, l \rangle \Downarrow_{\widetilde{bn}} \mathsf{M}'$ of rules $\textsc{Red}_{\widetilde{no}}$ and $\textsc{Neu}_{\widetilde{no}}$, with M' later used by their respective second premiss. Operationally, $\langle \mathsf{M}, l \rangle \Downarrow_{\widetilde{bn}} \mathsf{M}'$ is computed first and then the appropriate rule (either $\textsc{Red}_{\widetilde{no}}$ or $\textsc{Neu}_{\widetilde{no}}$) is selected depending on whether M' is a closure abstraction $\boldsymbol{\lambda}.B[\overline{n} : \rho]$ or a neutral closure $\lfloor n \rfloor \{ \cdot \mathsf{C} \}^*$ as in the case of examples (1) and (2) above.

Coalescing ephemeral expansion for closure abstraction is straightforward. First, we have to change rule $\textsc{Lam}_{\widetilde{bn}}$ so that it delivers the body $B[\overline{l+1} : \rho]$, a proper closure. Second, we have to modify the second premiss of $\textsc{Red}_{\widetilde{bn}}$, $\textsc{Neu}_{\widetilde{bn}}$, $\textsc{Red}_{\widetilde{no}}$, and $\textsc{Neu}_{\widetilde{no}}$, to a check on whether M' is a proper closure. We shall come back to this in Section 8.1.

Coalescing ephemeral expansion for neutral closures would be possible if its operands where in nf$_{\mathsf{C}}$ after call-by-name. That is, if the reduction steps within the third premiss $\langle \mathsf{M}', l \rangle \Downarrow_{\widetilde{no}} \mathsf{M}''$ of $\textsc{Neu}_{\widetilde{no}}$ that normalise the operands of neutral closures were preponed to the call-by-name steps of the first premiss $\langle \mathsf{M}, l \rangle \Downarrow_{\widetilde{bn}} \mathsf{M}'$ of that same rule. Fortunately, this can be easily achieved by copying the last premiss $\langle \mathsf{N}, l \rangle \Downarrow_{\widetilde{no}} \mathsf{N}'$ in $\textsc{Neu}_{\widetilde{no}}$ and pasting it as the last premiss in $\textsc{Neu}_{\widetilde{bn}}$, and by removing the third premiss $\langle \mathsf{M}', l \rangle \Downarrow_{\widetilde{no}} \mathsf{M}''$ in $\textsc{Neu}_{\widetilde{no}}$ which is no longer needed because M' would now be in nf$_{\mathsf{C}}$. Figure 13 shows the resulting $\textsc{Neu}^{p}_{\widetilde{bn}}$ and $\textsc{Neu}^{p}_{\widetilde{no}}$ rules, relabelled with a p superscript for readability. The remaining rules stay as in Figure 12 save for the addition of the superscript.

Due to this transformation both $\Downarrow^{p}_{\widetilde{bn}}$ and $\Downarrow^{p}_{\widetilde{no}}$ are mutually recursive hybrid strategies. The resulting preponed normaliser is implemented by functions `normalise15_whnf` and `normalise15_nf` in the code.

The preponing step is intuitive and its correctness is proven by induction on derivations. The correctness can also be proven in the alternative eval-readback version of the reduction-free normaliser. The single-stage and eval-readback normalisers are inter-derivable by inverse and direct lightweight fusion by fixed-point promotion [21]. For the sake of completeness we have included their detailed inter-derivation in the code (entry functions `normalise7` to `normalise14`).

Recall from Section 3 that $\Downarrow_{no} = \Downarrow_{rn} \circ \Downarrow_{bn}$. Preponing here consists of moving to the call-by-name stage the first reduction steps of $\Downarrow_{rn}$ for applications, namely those of the first premiss $M \Downarrow_{rn} M'$ of $\textsc{App}_{rn}$. In other words, it consists of shifting the point at which the eval ends and the readback begins when reducing neutrals. This is easily achieved by removing the first premiss $M \Downarrow_{rn} M'$ of $\textsc{App}_{rn}$, and then copying the last two premisses $N \Downarrow_{bn} N'$ and $N' \Downarrow_{rn} N''$ of the same rule, and pasting them as the last two premisses of rule $\textsc{Neu}_{bn}$ in Figure 2.

8.1 Shortcut normaliser

The preponed reduction-free normaliser can now be shortcut [5] resulting in the reduction-free normaliser implementing the natural semantics in Figure 14. The normaliser is implemented by functions `normalise16_whnf` and `normalise16_nf` in the code.

Shortcutting removes ephemeral constructors and rules $\textsc{Abs}^{p}_{\widetilde{bn}}$, $\textsc{App}^{p}_{\widetilde{bn}}$, $\textsc{Abs}^{p}_{\widetilde{no}}$, and $\textsc{App}^{p}_{\widetilde{no}}$. The resulting rules in Figure 14 now deliver ground terms except for $\textsc{Var}_{\overline{bn}}$, $\textsc{Red}_{\overline{bn}}$, and $\textsc{Lam}_{\overline{bn}}$. As explained in Section 8, instead of an abstraction closure, rule $\textsc{Lam}_{\overline{bn}}$ now delivers the body $B[\overline{l+1} : \rho]$, a proper closure. Rules $\textsc{Var}_{\overline{bn}}$ and $\textsc{Red}_{\overline{bn}}$ simply propagate proper closures. Consequently, the second premiss of $\textsc{Red}_{\overline{bn}}$ and $\textsc{Red}_{\overline{no}}$ checks if M' is a proper clos-

$$\frac{n < |\rho| \quad \langle n^{\mathrm{th}}(\rho), l\rangle \Downarrow_{\widetilde{bn}} \mathsf{N}}{\langle n[\rho], l\rangle \Downarrow_{\widetilde{bn}} \mathsf{N}}\ (\textsc{Var}_{\widetilde{bn}}) \qquad \frac{n \geq |\rho|}{\langle n[\rho], l\rangle \Downarrow_{\widetilde{bn}} \lfloor n - (|\rho| - l)\rfloor}\ (\textsc{Fre}_{\widetilde{bn}}) \qquad \frac{}{\langle \overline{n}, l\rangle \Downarrow_{\widetilde{bn}} \lfloor l - n\rfloor}\ (\textsc{Par}_{\widetilde{bn}})$$

$$\frac{}{\langle (\lambda.B)[\rho], l\rangle \Downarrow_{\widetilde{bn}} \lambda\!\!\lambda.B[\overline{l+1} : \rho]}\ (\textsc{Lam}_{\widetilde{bn}}) \qquad \frac{}{\langle \lfloor n\rfloor, l\rangle \Downarrow_{\widetilde{bn}} \lfloor n\rfloor}\ (\textsc{Abs}_{\widetilde{bn}}) \qquad \frac{\langle M[\rho]\cdot N[\rho], l\rangle \Downarrow_{\widetilde{bn}} \mathsf{C}}{\langle (M\,N)[\rho], l\rangle \Downarrow_{\widetilde{bn}} \mathsf{C}}\ (\textsc{App}_{\widetilde{bn}})$$

$$\frac{\langle \mathsf{M}, l\rangle \Downarrow_{\widetilde{bn}} \mathsf{M}' \quad \mathsf{M}' \equiv \lambda\!\!\lambda.B[\overline{n} : \rho] \quad \langle B[\mathsf{N} : \rho], l\rangle \Downarrow_{\widetilde{bn}} \mathsf{B}'}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \Downarrow_{\widetilde{bn}} \mathsf{B}'}\ (\textsc{Red}_{\widetilde{bn}}) \qquad \frac{\langle \mathsf{M}, l\rangle \Downarrow_{\widetilde{bn}} \mathsf{M}' \quad \mathsf{M}' \not\equiv \lambda\!\!\lambda.B[\overline{n} : \rho]}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \Downarrow_{\widetilde{bn}} \mathsf{M}'\cdot\mathsf{N}}\ (\textsc{Neu}_{\widetilde{bn}})$$

$$\frac{n < |\rho| \quad \langle n^{\mathrm{th}}(\rho), l\rangle \Downarrow_{\widetilde{no}} \mathsf{N}}{\langle n[\rho], l\rangle \Downarrow_{\widetilde{no}} \mathsf{N}}\ (\textsc{Var}_{\widetilde{no}}) \qquad \frac{n \geq |\rho|}{\langle n[\rho], l\rangle \Downarrow_{\widetilde{no}} \lfloor n - (|\rho| - l)\rfloor}\ (\textsc{Fre}_{\widetilde{no}}) \qquad \frac{}{\langle \overline{n}, l\rangle \Downarrow_{\widetilde{no}} \lfloor l - n\rfloor}\ (\textsc{Par}_{\widetilde{no}})$$

$$\frac{\langle B[\overline{l+1} : \rho], l+1\rangle \Downarrow_{\widetilde{no}} \mathsf{B}'}{\langle (\lambda.B)[\rho], l\rangle \Downarrow_{\widetilde{no}} \lambda\!\!\lambda.\mathsf{B}'}\ (\textsc{Lam}_{\widetilde{no}}) \qquad \frac{}{\langle \lfloor n\rfloor, l\rangle \Downarrow_{\widetilde{no}} \lfloor n\rfloor}\ (\textsc{Abs}_{\widetilde{no}}) \qquad \frac{\langle M[\rho]\cdot N[\rho], l\rangle \Downarrow_{\widetilde{no}} \mathsf{C}}{\langle (M\,N)[\rho], l\rangle \Downarrow_{\widetilde{bn}} \mathsf{C}}\ (\textsc{App}_{\widetilde{no}})$$

$$\frac{\langle \mathsf{M}, l\rangle \Downarrow_{\widetilde{bn}} \mathsf{M}' \quad \mathsf{M}' \equiv \lambda\!\!\lambda.B[\overline{n} : \rho] \quad \langle B[\mathsf{N} : \rho], l\rangle \Downarrow_{\widetilde{no}} \mathsf{B}'}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \Downarrow_{\widetilde{no}} \mathsf{B}'}\ (\textsc{Red}_{\widetilde{no}})$$

$$\frac{\langle \mathsf{M}, l\rangle \Downarrow_{\widetilde{bn}} \mathsf{M}' \quad \mathsf{M}' \not\equiv \lambda\!\!\lambda.B[\overline{n} : \rho] \quad \langle \mathsf{M}', l\rangle \Downarrow_{\widetilde{no}} \mathsf{M}'' \quad \langle \mathsf{N}, l\rangle \Downarrow_{\widetilde{no}} \mathsf{N}'}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \Downarrow_{\widetilde{no}} \mathsf{M}''\cdot\mathsf{N}'}\ (\textsc{Neu}_{\widetilde{no}})$$

Figure 12. Natural semantics of normal order in $\lambda_{\widetilde{\rho}}$.

$$\frac{\langle \mathsf{M}, l\rangle \Downarrow^{p}_{\widetilde{bn}} \mathsf{M}' \quad \mathsf{M}' \not\equiv \lambda\!\!\lambda.B[\overline{n} : \rho] \quad \langle \mathsf{N}, l\rangle \Downarrow^{p}_{\widetilde{no}} \mathsf{N}'}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \Downarrow^{p}_{\widetilde{bn}} \mathsf{M}'\cdot\mathsf{N}'}\ (\textsc{Neu}^{p}_{\widetilde{bn}}) \qquad \frac{\langle \mathsf{M}, l\rangle \Downarrow^{p}_{\widetilde{bn}} \mathsf{M}' \quad \mathsf{M}' \not\equiv \lambda\!\!\lambda.B[\overline{n} : \rho] \quad \langle \mathsf{N}, l\rangle \Downarrow^{p}_{\widetilde{no}} \mathsf{N}'}{\langle \mathsf{M}\cdot\mathsf{N}, l\rangle \Downarrow^{p}_{\widetilde{no}} \mathsf{M}'\cdot\mathsf{N}'}\ (\textsc{Neu}^{p}_{\widetilde{no}})$$

Figure 13. Rules changed by preponing. The remaining rules are the same as in Figure 12 save for the addition of the p superscript.

$$\frac{n < |\rho| \quad \langle n^{\mathrm{th}}(\rho), l\rangle \Downarrow_{\overline{bn}} \mathsf{N}}{\langle n[\rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{N}}\ (\textsc{Var}_{\overline{bn}}) \qquad \frac{n \geq |\rho|}{\langle n[\rho], l\rangle \Downarrow_{\overline{bn}} \lfloor n - (|\rho| - l)\rfloor}\ (\textsc{Fre}_{\overline{bn}}) \qquad \frac{}{\langle \overline{n}, l\rangle \Downarrow_{\overline{bn}} \lfloor l - n\rfloor}\ (\textsc{Par}_{\overline{bn}})$$

$$\frac{}{\langle (\lambda.B)[\rho], l\rangle \Downarrow_{\overline{bn}} B[\overline{l+1} : \rho]}\ (\textsc{Lam}_{\overline{bn}}) \qquad \frac{\langle M[\rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{M}' \quad \mathsf{M}' \equiv B[\overline{n} : \rho] \quad \langle B[N[\rho] : \rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{B}'}{\langle (M\,N)[\rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{B}'}\ (\textsc{Red}_{\overline{bn}})$$

$$\frac{\langle M[\rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{M}' \quad \mathsf{M}' \equiv \lfloor M'\rfloor \quad \langle N[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor N'\rfloor}{\langle (M\,N)[\rho], l\rangle \Downarrow_{\overline{bn}} \lfloor M'\,N'\rfloor}\ (\textsc{Neu}_{\overline{bn}})$$

$$\frac{n < |\rho| \quad \langle n^{\mathrm{th}}(\rho), l\rangle \Downarrow_{\overline{no}} \lfloor N\rfloor}{\langle n[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor N\rfloor}\ (\textsc{Var}_{\overline{no}}) \qquad \frac{n \geq |\rho|}{\langle n[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor n - (|\rho| - l)\rfloor}\ (\textsc{Fre}_{\overline{no}}) \qquad \frac{}{\langle \overline{n}, l\rangle \Downarrow_{\overline{no}} \lfloor l - n\rfloor}\ (\textsc{Par}_{\overline{no}})$$

$$\frac{\langle B[\overline{l+1} : \rho], l+1\rangle \Downarrow_{\overline{no}} \lfloor B'\rfloor}{\langle (\lambda.B)[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor \lambda.B'\rfloor}\ (\textsc{Lam}_{\overline{no}}) \qquad \frac{\langle M[\rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{M}' \quad \mathsf{M}' \equiv B[\overline{n} : \rho] \quad \langle B[N[\rho] : \rho], l\rangle \Downarrow_{\overline{no}} \lfloor B'\rfloor}{\langle (M\,N)[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor B'\rfloor}\ (\textsc{Red}_{\overline{no}})$$

$$\frac{\langle M[\rho], l\rangle \Downarrow_{\overline{bn}} \mathsf{M}' \quad \mathsf{M}' \equiv \lfloor M'\rfloor \quad \langle N[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor N'\rfloor}{\langle (M\,N)[\rho], l\rangle \Downarrow_{\overline{no}} \lfloor M'\,N'\rfloor}\ (\textsc{Neu}_{\overline{no}})$$

Figure 14. Shortcut natural semantics of normal order in $\lambda_{\widetilde{\rho}}$.

$$\mathbf{c} \ ::= \ \mathbf{w} \mid \mathbf{n}$$

$$\frac{n < |\rho| \quad \langle n^{\mathrm{th}}(\rho), l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \mathsf{N}}{\langle n[\rho], l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \mathsf{N}}\ (\textsc{Var}_{\overline{ctl}}) \qquad \frac{n \geq |\rho|}{\langle n[\rho], l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \lfloor n - (|\rho| - l)\rfloor}\ (\textsc{Fre}_{\overline{ctl}}) \qquad \frac{}{\langle \overline{n}, l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \lfloor l - n\rfloor}\ (\textsc{Par}_{\overline{ctl}})$$

$$\frac{}{\langle (\lambda.B)[\rho], l, \mathbf{w}\rangle \Downarrow_{\overline{ctl}} B[\overline{l+1} : \rho]}\ (\textsc{Lam1}_{\overline{ctl}}) \qquad \frac{\langle M[\rho], l, \mathbf{w}\rangle \Downarrow_{\overline{ctl}} \mathsf{M}' \quad \mathsf{M}' \equiv B[\overline{n} : \rho] \quad \langle B[N[\rho] : \rho], l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \mathsf{B}'}{\langle (M\,N)[\rho], l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \mathsf{B}'}\ (\textsc{Red}_{\overline{ctl}})$$

$$\frac{\langle B[\overline{l+1} : \rho], l+1, \mathbf{n}\rangle \Downarrow_{\overline{ctl}} \lfloor B'\rfloor}{\langle (\lambda.B)[\rho], l, \mathbf{n}\rangle \Downarrow_{\overline{ctl}} \lfloor \lambda.B'\rfloor}\ (\textsc{Lam2}_{\overline{ctl}}) \qquad \frac{\langle M[\rho], l, \mathbf{w}\rangle \Downarrow_{\overline{ctl}} \mathsf{M}' \quad \mathsf{M}' \equiv \lfloor M'\rfloor \quad \langle N[\rho], l, \mathbf{n}\rangle \Downarrow_{\overline{ctl}} \lfloor N'\rfloor}{\langle (M\,N)[\rho], l, \mathbf{c}\rangle \Downarrow_{\overline{ctl}} \lfloor M'\,N'\rfloor}\ (\textsc{Neu}_{\overline{ctl}})$$

Figure 15. Natural semantics of normal order in $\lambda_{\widetilde{\rho}}$ with explicit control.

ure, and the second premiss of $\textsc{Neu}_{\overline{bn}}$ and of $\textsc{Neu}_{\overline{no}}$ checks if M' is a ground term.

The underlying calculus, which we call $\lambda^*_{\widetilde{\rho}}$, is an optimised variant of $\lambda_{\widetilde{\rho}}$ that omits the levels in ground terms $\lfloor \Lambda \rfloor$:

$$\begin{array}{rcl} & \lambda^*_{\widetilde{\rho}} & \\ \mathsf{C} & ::= & \Lambda[\rho] \mid \overline{n} \mid \lfloor \Lambda \rfloor \\ \rho & ::= & \epsilon \mid \mathsf{C} : \rho \end{array}$$

9. From reduction-free normaliser to push/enter abstract machine

9.1 A reduction-free normaliser with explicit control

The mutually recursive $\Downarrow_{\overline{bn}}$ and $\Downarrow_{\overline{no}}$ of the shortcut natural semantics in Figure 14 differ in the treatment of abstractions. Rule $\textsc{Lam}_{\overline{bn}}$ takes place when the abstraction is applied to an operand whereas $\textsc{Lam}_{\overline{no}}$ takes place when the abstraction is unapplied. We transform the shortcut normalisers `normalise16_whnf` and `normalise16_nf` into a single `normalise_ctl` normaliser with explicit control that encodes the different treatment of abstractions. We introduce the control characters $\mathbf{w}$ and $\mathbf{n}$ that respectively encode a subderivation with $\textsc{Lam}_{\overline{bn}}$ and a subderivation with $\textsc{Lam}_{\overline{no}}$. The normaliser with explicit control implements the natural semantics of Figure 15. The control character $\mathbf{w}$ is used for operators in applications, and $\mathbf{n}$ for operands of neutral closures.

9.2 From reduction-free normaliser to eval/apply abstract machine

We apply defunctionalisation and CPS transformation to the normaliser with explicit control to obtain the eval/apply machine with explicit control shown in Figure 16. The machine is implemented by functions `normalise_ctl_cont` and `apply_ctl_cont` in the code. The horizontal bar in the middle separates the eval configuration from the apply configuration. The eval configuration pattern-matches on the control character $\mathbf{c}$ to decide whether to go under lambda. The apply configuration does not use the control character. The occurrence of the control character discriminates both configurations and there is no need for type annotations. Observe the use of $\mathbf{w}$ when reducing operators in applications and the use of $\mathbf{n}$ when reducing operands in neutral closures. Observe that continuation $\mathbf{C}_1(\mathsf{C}, \mathbf{c})$ carries along the control character which is restored after contraction (first rule of the apply configuration).

9.3 Removing explicit control

Once the normaliser is in defunctionalised CPS, the correlation between explicit control and the continuation stack can be observed. The machine uses $\mathbf{w}$ when continuation $\mathbf{C}_1$ is pushed on the stack. The remaining transitions just preserve the current control, except for the transition dealing with operands in neutral closures,

$$\begin{array}{rcl} S & ::= & \mathbf{C}_0 \mid \mathbf{C}_1(\mathsf{C}, \mathbf{c}) : S \mid \mathbf{C}_2 : S \mid \mathbf{C}_3(\mathsf{C}) : S \\ \mathbf{c} & ::= & \mathbf{w} \mid \mathbf{n} \end{array}$$

	T	$\rightarrow$	$(T[\epsilon], \mathbf{C}_0, \mathbf{0}, \mathbf{n})$
(if $n < \|\rho\|$)	$(n[\rho], S, l, \mathbf{c})$	$\rightarrow$	$(n^{\mathrm{th}}(\rho), S, l, \mathbf{c})$
(if $n \geq \|\rho\|$)	$(n[\rho], S, l, \mathbf{c})$	$\rightarrow$	$(\lfloor n - (\|\rho\| - l)\rfloor, S, l)$
	$(\overline{n}, S, l, \mathbf{c})$	$\rightarrow$	$(\lfloor l - n\rfloor, S, l)$
	$((\lambda.B)[\rho], S, l, \mathbf{w})$	$\rightarrow$	$(B[\overline{l+1} : \rho], S, l)$
	$((\lambda.B)[\rho], S, l, \mathbf{n})$	$\rightarrow$	$(B[\overline{l+1} : \rho], \mathbf{C}_2 : S, l+1, \mathbf{n})$
	$((M\,N)[\rho], S, l, \mathbf{c})$	$\rightarrow$	$(M[\rho], \mathbf{C}_1(N[\rho], \mathbf{c}) : S, l, \mathbf{w})$
	$(B[\overline{n} : \rho], \mathbf{C}_1(\mathsf{N}, \mathbf{c}) : S, l)$	$\rightarrow$	$(B[\mathsf{N} : \rho], S, l, \mathbf{c})$
	$(\lfloor M\rfloor, \mathbf{C}_1(\mathsf{N}, \mathbf{c}) : S, l)$	$\rightarrow$	$(\mathsf{N}, \mathbf{C}_3(\lfloor M\rfloor) : S, l, \mathbf{n})$
	$(\lfloor B\rfloor, \mathbf{C}_2 : S, l)$	$\rightarrow$	$(\lfloor \lambda.B\rfloor, S, l-1)$
	$(\lfloor N\rfloor, \mathbf{C}_3(\lfloor M\rfloor) : S, l)$	$\rightarrow$	$(\lfloor M\,N\rfloor, S, l)$
	$(\lfloor T\rfloor, \mathbf{C}_0, l)$	$\rightarrow$	T

Figure 16. Eval/apply machine with explicit control and continuation stack S.

where the machine uses $\mathbf{n}$ and pushes $\mathbf{C}_3$ on the stack, signalling the point at which call-by-name ends and normal order resumes. Consequently, control character $\mathbf{w}$ can be replaced by checking for the occurrence of $\mathbf{C}_1$ on the top of the stack, and control character $\mathbf{n}$ can be dropped because it is used only when the machine resumes normal order. This fact can be proven more rigorously by constructing the following grammar of well-formed stack values. The grammar can be obtained from the reduction semantics of Figure 10 in similar fashion as in [18]:

$$\begin{array}{rcl} S & ::= & \{D\}^? \mathbf{C}_0 \\ D & ::= & \{\{\mathbf{C}_1(\mathsf{C}, \mathbf{c}) :\}^* \mathbf{C}_3(\mathsf{C}) :\}^? \{\mathbf{C}_2 :\}^* \\ & \mid & \{D\}^? \mathbf{C}_5(\lfloor n\rfloor\{\,\cdot\,\mathrm{NF}_{\mathsf{C}}\}^*) : \{\mathbf{C}_4(\mathsf{C}) :\}^* \{\mathbf{C}_2 :\}^* \end{array}$$

Pattern-matching on the stack breaks the shallow inspection required to refunctionalise the machine, but this context-dependency is present in KN and has to be introduced at some point in order to derive the machine.

The resulting machine with implicit control in Figure 17 is implemented by functions `normalise20_cont` and `apply20_cont` in the code. Type annotations are required again to distinguish the eval and apply configurations.

9.4 From eval/apply to push/enter machine

To turn the machine into push/enter, the apply function has to be inlined in eval. There are three eval transitions going to apply, namely the 2nd, 3th, and 4th. The last can be inlined ('compressed') with the first transition of apply. To inline the other two we first 'protrude' (inverse inline) them into a new eval transition for ground

$$S \quad ::= \quad \mathbf{C}_0 \mid \mathbf{C}_1(\mathsf{C}) : S \mid \mathbf{C}_2 : S \mid \mathbf{C}_3(\mathsf{C}) : S$$

$$\begin{array}{|rcl|}
\hline
T & \rightarrow & (T[\epsilon], \mathbf{C}_0, 0)_e \\
\hline
(\text{if } n < |\rho|) \quad (n[\rho], S, l)_e & \rightarrow & (n^{\text{th}}(\rho), S, l)_e \\
(\text{if } n \geq |\rho|) \quad (n[\rho], S, l)_e & \rightarrow & (\lfloor n - (|\rho| - l) \rfloor, S, l)_a \\
(\overline{n}, S, l)_e & \rightarrow & (\lfloor l - n \rfloor, S, l)_a \\
((\lambda.B)[\rho], \mathbf{C}_1(\mathsf{N}) : S, l)_e & \rightarrow & (B[\overline{l+1} : \rho], \mathbf{C}_1(\mathsf{N}) : S, l)_a \\
((\lambda.B)[\rho], S, l)_e & \rightarrow & (B[\overline{l+1} : \rho], \mathbf{C}_2 : S, l+1)_e \\
((M\,N)[\rho], S, l)_e & \rightarrow & (M[\rho], \mathbf{C}_1(N[\rho]) : S, l)_e \\
\hline
(B[\overline{n} : \rho], \mathbf{C}_1(\mathsf{N}) : S, l)_a & \rightarrow & (B[\mathsf{N} : \rho], S, l)_e \\
(\lfloor M \rfloor, \mathbf{C}_1(\mathsf{N}) : S, l)_a & \rightarrow & (\mathsf{N}, \mathbf{C}_3(\lfloor M \rfloor) : S, l)_e \\
(\lfloor B \rfloor, \mathbf{C}_2 : S, l)_a & \rightarrow & (\lfloor \lambda.B \rfloor, S, l-1)_a \\
(\lfloor N \rfloor, \mathbf{C}_3(\lfloor M \rfloor) : S, l)_a & \rightarrow & (\lfloor M\,N \rfloor, S, l)_a \\
\hline
(\lfloor T \rfloor, \mathbf{C}_0, l)_a & \rightarrow & T \\
\hline
\end{array}$$

Figure 17. Eval/apply machine with implicit control.

$$\begin{array}{|rcl|}
\hline
T & \rightarrow & (T[\epsilon], \mathbf{C}_0, 0) \\
\hline
(\text{if } n < |\rho|) \quad (n[\rho], S, l) & \rightarrow & (n^{\text{th}}(\rho), S, l) \\
(\text{if } n \geq |\rho|) \quad (n[\rho], S, l) & \rightarrow & (\lfloor n - (|\rho| - l) \rfloor, S, l) \\
(\overline{n}, S, l) & \rightarrow & (\lfloor l - n \rfloor, S, l) \\
((\lambda.B)[\rho], \mathbf{C}_1(\mathsf{N}) : S, l) & \rightarrow & (B[\mathsf{N} : \rho], S, l) \\
((\lambda.B)[\rho], S, l) & \rightarrow & (B[\overline{l+1} : \rho], \mathbf{C}_2 : S, l+1) \\
((M\,N)[\rho], S, l) & \rightarrow & (M[\rho], \mathbf{C}_1(N[\rho]) : S, l) \\
(\lfloor M \rfloor, \mathbf{C}_1(\mathsf{N}) : S, l) & \rightarrow & (\mathsf{N}, \mathbf{C}_3(\lfloor M \rfloor) : S, l) \\
(\lfloor B \rfloor, \mathbf{C}_2 : S, l) & \rightarrow & (\lfloor \lambda.B \rfloor, S, l-1) \\
(\lfloor N \rfloor, \mathbf{C}_3(\lfloor M \rfloor) : S, l) & \rightarrow & (\lfloor M\,N \rfloor, S, l) \\
\hline
(\lfloor T \rfloor, \mathbf{C}_0, l) & \rightarrow & T \\
\hline
\end{array}$$

Figure 18. Push/enter machine (optimised KN).

$$S \ ::= \ \epsilon \mid \Lambda[\rho] : S \mid \lambda : S \mid \lfloor \Lambda \rfloor : S$$

$$\begin{array}{|rcl|}
\hline
T & \rightarrow & (T[\epsilon], \epsilon, 0) \\
\hline
((n+1)[\mathsf{C} : \rho], S, l) & \rightarrow & (n[\rho], S, l) \\
(0[\mathsf{C} : \rho], S, l) & \rightarrow & (\mathsf{C}, S, l) \\
(n[\epsilon], S, l) & \rightarrow & (\lfloor n + l \rfloor, S, l) \\
((M\,N)[\rho], S, l) & \rightarrow & (M[\rho], N[\rho] : S, l) \\
((\lambda.B)[\rho], N[\rho'] : S, l) & \rightarrow & (B[N[\rho'] : \rho], S, l) \\
((\lambda.B)[\rho], S, l) & \rightarrow & (B[\overline{l+1} : \rho], \lambda : S, l+1) \\
(\overline{n}, S, l) & \rightarrow & (\lfloor l - n \rfloor, S, l) \\
(\lfloor M \rfloor, N[\rho] : S, l) & \rightarrow & (N[\rho], \lfloor M \rfloor : S, l) \\
(\lfloor B \rfloor, \lambda : S, l) & \rightarrow & (\lfloor \lambda.B \rfloor, S, l-1) \\
(\lfloor N \rfloor, \lfloor M \rfloor : S, l) & \rightarrow & (\lfloor M\,N \rfloor, S, l) \\
\hline
(\lfloor T \rfloor, \epsilon, l) & \rightarrow & T \\
\hline
\end{array}$$

Figure 19. Push/enter machine (optimised KN) in format closer to original.

terms going to apply:

$$\begin{array}{rcl}
(\text{if } n \geq |\rho|) \quad (n[\rho], S, l)_e & \rightarrow & (\lfloor n - (|\rho| - l) \rfloor, S, l)_e \\
(\overline{n}, S, l)_e & \rightarrow & (\lfloor l - n \rfloor, S, l)_e \\
(\lfloor n \rfloor, S, l)_e & \rightarrow & (\lfloor n \rfloor, S, l)_a
\end{array}$$

The rest of the transitions remain the same and are omitted. The protruded machine is implemented in the code by functions `normalise21_cont` and `apply21_cont`. We inline the transitions of apply for ground terms into the new transition in the protruded machine and obtain the push/enter machine shown in Figure 18, which is implemented in the code by function `normalise22_push`.

Save for two minor *visual* differences that we discuss in the next paragraph, this machine is an optimised version of the original KN that can work with open terms. The optimisation is minor: embedded ground terms do not carry a level, so such levels need not be recovered from the environment when reducing operands of neutral closures, because the machine decrements the current level when leaving a lambda scope, as specified by rule ($\xi_{\overline{\rho}}$) in the structural operational semantics of normal order in Figure 9. Naturally, the machine can take closed terms as input. The clause for free variables would simply not be used.

The visual differences with the original KN of Figure 5 are the following. First, the use of n^{th} for lookup instead of a recursive peeling-off of the environment (n^{th} can be implemented by recursive peel-off, but also by random access). Second, the presence of defunctionalised continuations coming from the stack S defined in Figure 17.

We remove the visual differences in Figure 19. The n^{th} function for lookup is replaced by a peeling-off definition (consequently, the transition for free variables $n[\epsilon]$ has to be adapted). And defunctionalised continuations in S are replaced by the constructors of $\lambda^*_{\overline{\rho}}$ (and the control character λ) that they represent (collected in stack S in Figure 19). The machine is implemented by function `normalise23_push` in the code.

We have derived KN and are now at the end of our journey.

10. Related and future work

Single-stage and eval-readback (Section 3) approaches require different CPS transformations. For the former, a single-layer CPS without control delimiters is enough [17] because reduction is performed in a single stage. All the artefacts shown in this paper are single-stage. For eval-readback, either a two-layer CPS or a single-layer CPS with control delimiters is required [6]. Both NBE and eval-readback are popular within the programming languages community. However, single-stage structural and natural semantics are conceptually simpler, and their implementations more amenable to program transformation because no specific CPS techniques nor meta-theory for delimiting control is required.

In a recent personal communication with Olivier Danvy, he has informed us of a related unpublished work [13] that presents a derivation involving the full-reducing machine of Curien [10] itself based on the KAM machine [7]. Our work and [13] have been independently developed and are, to our knowledge, the only works demonstrating the derivation of full-reducing machines. The differences between our work and Danvy's are substantial.

- The full-reducing machines are different. Moreover, we *arrive* at KN whereas [13] *departs* from Curien's machine.
- We follow a single-stage approach to derive KN, and use single-layer CPS and plain CPS-related techniques. In contrast, [13] follows the eval-readback approach present in Curien's machine and presents two derivation paths, one using two-layer CPS and another using single-layer CPS with control delimiters.
- The precise control of levels in $\lambda\!\!\lambda$ scopes (rules $\text{L}\small{\text{AM}}_{\overline{\rho}}$ and ($\xi_{\overline{\rho}}$) in Figure 9) results in index alignment and balanced derivations which makes reasoning by structural induction easier and substantiates the optimisation of the original KN machine (Section 6.1). In [13] environments carry a lexical adjustment value that is incremented when popping a binding off the environment which complicates reasoning by structural induction on environments.
- In Section 9, we introduce explicit control to combine the hybrid and subsidiary reduction-free normalisers into one, and derive an explicit-control eval/apply abstract machine. When

removing explicit control the resulting machine is context-dependent, i.e., it does not have the *shallow-inspection* property. This prevents the refunctionalisation of the machine. However, the problem is not in our derivation but in the fact that context-dependency is present in KN, and has to be introduced at some point in order to derive the machine. In any case, we have shown in Figures 11 and 16 that environment-based machines with the shallow-inspection property can be derived. In [13], machines do not have explicit control because two-layer CPS or single-layer CPS with control delimiters are used.

In [19] a full-reducing strategy is specified in eval-readback style that is used in a proof assistant. The eval stage $V()$ is implemented by an optimised, pre-compiled abstract machine. This machine has been contrived, not derived. The readback stage $N()$ is symbolic. The strategy resulting from the composition of $V()$ and $N()$ is the same as the strategy resulting from the composition of symbolic `eval` and symbolic `byValue` in [22, p.390], save for the right-to-left sequencing order in which operands are reduced before operators. (The strategy implements strict semantics for redices, but performs β-reduction, not the β_V-reduction of the lambda-value calculus of [23], and consequently, it is not a full-reducing strategy of that calculus.) We are currently studying the derivation of a *whole* machine from the single-stage natural semantics obtained (via lightweight fusion by fixed-point promotion) from eval-readback `eval` and `byValue`. A question to answer is whether optimisations can be incorporated by program transformation. We are also studying the inter-derivation of a machine from the single-stage structural and natural semantics of the full-reducing strategy of the lambda-value calculus that we have presented in [16].

The closure calculi $\lambda_{\widetilde{\rho}}$ and $\lambda^*_{\overline{\rho}}$ we have introduced are rather natural extensions of λ_ρ, as illustrated by the following diagram:

$$\begin{tikzcd}
& \lambda_\rho \arrow[dl, "\begin{array}{l}+\overline{n}\\+\lfloor\Lambda\rfloor\end{array}"'] \arrow[dr, "+\mathsf{c}\cdot\mathsf{c}"] & \\
\lambda^*_{\overline{\rho}} \arrow[dr, "\begin{array}{l}\lfloor\Lambda\rfloor\approx\lfloor n\rfloor\\ \quad\mid \lambda\!\!\lambda.\mathsf{c}\\ \quad\mid \mathsf{c}\cdot\mathsf{c}\end{array}"'] & & \lambda_{\widehat{\rho}} \arrow[dl, "\begin{array}{l}+\overline{n}\\+\lfloor n\rfloor\\+\lambda\!\!\lambda.\mathsf{c}\end{array}"] \\
& \lambda_{\widetilde{\rho}} &
\end{tikzcd}$$

We have not defined the reduction theory of $\lambda_{\widetilde{\rho}}$, only presented the reduction strategy $\widetilde{no}$, which has taken us to the KN machine. Such theory is of interest since it has to consider compatibility with environments (reducing bindings inside environments) which poses a challenge.

Acknowledgments

We are grateful to Oliver Danvy for his inspiring dTFP course on inter-derivation techniques, for his hospitality, advice, and for his comments on the ideas of this paper. We are also grateful to the PPDP'13 anonymous reviewers for their positive yet balanced comments, and for their words of encouragement.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] K. Aehlig and F. Joachimski. Operational aspects of untyped normalisation by evaluation. *Mathematical Structures in Computer Science*, 14(4):587–611, 2004.

[3] M. S. Ager, D. Biernacki, O. Danvy, and J. Midtgaard. A functional correspondence between evaluators and abstract machines. In *Proceedings of International Conference on Principles and Practice of Declarative Programming*, pages 8–19, 2003.

[4] H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North Holland, 1984.

[5] M. Biernacka and O. Danvy. A concrete framework for environment machines. *ACM Trans. Comput. Log*, 9(1):6:1–6:29, Dec. 2007.

[6] M. Biernacka, D. Biernacki, and O. Danvy. An operational foundation for delimited continuations in the CPS hierarchy. *CoRR*, abs/cs/0508048, 2005.

[7] P. Crégut. An abstract machine for lambda-terms normalization. In *Proceedings of LISP and Functional Programming*, pages 333–340, 1990.

[8] P. Crégut. Strongly reducing variants of the Krivine abstract machine. *Higher-Order and Symbolic Computation*, 20(3):209–230, Sept. 2007.

[9] P.-L. Curien. An abstract framework for environment machines. *Theoretical Computer Science*, 82(2):389–402, May 1991.

[10] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Progress in Theoretical Computer Science. Birkhaüser, 1993.

[11] H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, 1958.

[12] O. Danvy. From reduction-based to reduction-free normalization. *Electr. Notes. Theor. Comput. Sci*, 124(2):79–100, 2005.

[13] O. Danvy, K. Milikin, and J. Munk. A correspondence between reduction-based and reduction-free normalization functions. Manuscript, 2007.

[14] O. Danvy, J. Johannsen, and I. Zerny. A walk in the semantic park. In *Proceedings of the 2011 Workshop on Partial Evaluation and Program Manipulation*, pages 1–12, 2011.

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

[16] A. García-Pérez and P. Nogueira. Towards Böhm trees for lambda-value: the operational and proof-theoretical machinery. *Mathematical Structures in Computer Science*, 2012. Submitted for publication.

[17] A. García-Pérez and P. Nogueira. A syntactic and functional correspondence between reduction semantics and reduction-free full normalisers. In *Proceedings of the 2013 Symposium on Partial Evaluation and Program Manipulation*, pages 107–116, 2013.

[18] A. García-Pérez and P. Nogueira. A syntactic and functional correspondence between full-reducing normalisers and full-reducing abstract machines. *Science of Computer Programming*, 2013. Submitted for publication.

[19] B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *Proceedings of International Conference on Functional Programming*, pages 235–246, 2002.

[20] D. Kesner. The theory of calculi with explicit substitutions revisited. In *Proceedings of the 21st International Workshop on Computer Science Logic*, volume 4646 of *Lecture Notes in Computer Science*, pages 238–252. Springer, 2007.

[21] A. Ohori and I. Sasano. Lightweight fusion by fixed point promotion. In *Proceedings of Symposium on Principles of Programming Languages*, pages 143–154, 2007.

[22] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, second edition, 1996.

[23] G. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

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

[25] J. G. Siek and R. Garcia. Interpretations of the gradually-typed lambda calculus. In *Proceedings of the Scheme and Functional Programming Workshop*, 2012.

[26] W. Swierstra. From mathematics to abstract machine: a formal derivation of an executable Krivine machine. In *Proceedings of the 4th Workshop on Mathematically Structured Functional Programming*, pages 163–177, 2012.