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 Noguería
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, refunctorialisation, inlining, light-weight 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-translation 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

---

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.

---

* Research partially funded by the Spanish ‘Ministerio de Economía y Competitividad’ through projects DESAFIOS10 TIN2009-14599-C03-00 and STRONSOFT 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.

**Research partially funded by the Spanish ‘Ministerio de Economía y Competitividad’ through projects DESAFIOS10 TIN2009-14599-C03-00 and STRONSOFT 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.**
reduction semantics, but we only used the stack grammar to recover search functions normaliser (and so the reduction semantics) from name strategy (Section 3). Consequently, we can use single-layer but hybrid normal order strategy that relies on a subsidiary call-by-name strategy, i.e., it

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. Derivation path of KN.

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 \( \lambda_{\phi} \) calculus which naturally extends the \( \lambda_{\\phi} \) 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 \( \lambda_{\phi} \) to the pure lambda calculus \( \lambda \) by providing a substitution function \( \sigma \) that simulates capture-avoiding substitution in the pure lambda calculus.

- We define the small-step structural operational semantics of normal order in \( \lambda_{\phi} \) 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 \( \lambda_{\phi} \).

- 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 \( \lambda_{\phi} \) without ephemeral terms that we call \( \lambda_{\phi}^\prime \), 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 \( \lambda \) for short, whose terms are defined by the grammar \( \Lambda ::= n \mid (\Lambda) \mid \lambda \Lambda \). A natural number \( n \) represents a variable bound to the \( n \)th lambda starting from 0, or a free variable when \( n \) is greater than or equal to the nesting level. For example, the abstraction \( \lambda n.0 \) is the identity function whereas \( \lambda n.1 \) is a constant function delivering free 0 when the function is applied to an operand. Uppercase, often primed, letters \( M, M', N, \)

\(^{2}\)The \( \lambda_{\phi} \) calculus is itself an extension of Curien’s lambda calculus of closures \( \lambda_{\phi} \) [9], which is an extension of the pure lambda calculus that adds closures for handling explicit substitutions [1, 20].

\(^{3}\)http://babel.is.fi.upm.es/~agarcia/papers/KrivineFull
Call-by-name in the pure untyped lambda calculus differs from call-by-normal order is an identity on variables. In contrast, call-by-name provides structural compatibility with abstractions, that is, 'go un-redex first', understanding 'leftmost' as in [11] or 'leftmost-outmost' 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 $(\lambda B)N$ contracted next instead of the redexes in $B$. For example, given the term $(\lambda 0 1)(\lambda .1)$ where $\Omega$ 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 $\Omega$.

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 (\beta) is well-known. The second (\mu) says the redex must be searched for in the operand if the operator is not in whnf. These two rules make up subsidiary call-by-name. The third rule (\mu) says the redex must be searched for in the operand if it is a whnf but not an abstraction (if an abstraction then (\beta) is applicable). Finally, (\nu) 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 (\mu) and (\nu) are non-overlapping because the third premise in (\mu) 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 $\{[\Lambda]^*\}$. Last, rule (\xi) provides structural compatibility with abstractions, that is, 'go under lambda'.

In the big-step, normal order $\psi_{bn}$ relies on subsidiary and uniform call-by-name $\psi_{bn}$ to reduce operators to whnf (first premises of rules $\text{Red}_n$ and $\text{Neu}_n$), and then fully reduces the resulting redex (rule $\text{Red}_n$) or the resulting neutral (rule $\text{Neu}_n$). Finally, $\text{LAM}_{no}$ says that normal order goes under lambda and $\text{VAR}_{no}$ 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.

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 $\psi_{bn} \circ \psi_{bn}$ where $\psi_{bn}$ is eval and $\psi_{bn}$ is readback:

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 $\psi_{no} = \psi_{bn} \circ \psi_{bn}$, 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 (\beta) within the context hole.

\begin{align*}
\text{Closure context:} & \quad C_{no} := [ | C_{bn} | \Lambda | \lambda C_{no} | | C_{ne} | ] \\
\text{Contraction:} & \quad C_{no}[(\lambda B)N] \rightarrow_{\text{no}} C_{bn}[\Lambda(N/B)]
\end{align*}

Given a term $M$, it is either in nf or is uniquely decomposed into a context, derived from non-terminal $C_{no}$, and a redex within the hole. The unique decomposition of $C_{no}$ 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: $C_{no} => C_{no} M_{no} [\Lambda(N/B)]$. Hybridisation is signalled by the presence of call-by-name subcontexts $C_{bn}$.

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 reduction semantics of call-by-name in λP (adapted from [5])

\[
\begin{align*}
\text{Red. context: } C_\text{cont} &::= \frac{C_\text{cont} \cdot (M \cdot N)[\rho]}{C_\text{cont} \cdot (M \cdot N)[\rho]} \quad (\text{APP}) \\
\text{Contraction: } C_\text{cont} &::= \frac{C_\text{cont} \cdot (M \cdot N)[\rho]}{C_\text{cont} \cdot (M \cdot N)[\rho]} \quad (\text{APP}) \\
\text{Reduction: } \sigma(C,N) &::= \frac{\lambda \cdot \sigma(C,N)}{\lambda \cdot \sigma(C,N)} \quad (\lambda) \\
\sigma(n[p], k) &::= \begin{cases} n & \text{if } n \leq k \\ \sigma(n - 1, k + 1) & \text{if } n > k \end{cases} \\
\sigma((\lambda B)[p], k) &::= \lambda \cdot \sigma(B[p], k + 1) \\
\sigma((M \cdot N)[p], k) &::= \sigma(M[p], k) \cdot \sigma(N[p], k) \\
\sigma(M[N, k]) &::= \sigma (M, k) \cdot \sigma (N, k)
\end{align*}
\]

Figure 4. Substitution function σ connects λP and λ.

The structural (left) and natural (right) operational semantics of normal order [17].

\[
\begin{align*}
\delta \cdot \rho \cdot \text{N} &\rightarrow_\text{in} \delta \cdot \text{M} \cdot \text{N} \quad (\text{APP}) \\
\delta \cdot \rho \cdot \text{M} \cdot \text{N} &\rightarrow_\text{in} \delta \cdot \text{M} \cdot \text{N} \quad (\text{APP}) \\
\end{align*}
\]

Figure 3. Structural and reduction semantics of call-by-name in λP (adapted from [5]).

The connection between λP 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 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 λP 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.

 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 λP 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.

5 "Ephemeral" in the sense that closure applications are shortcut when deriving big-step artefacts [5].
ρ := e | C : ρ
Λ : S | [Λ,l] : S

(1) T → (T[ρ], ϵ, 0)
(2) ((n + 1)[C : ρ], S, l) → (n[ρ], S, l)
(3) (0[C : ρ], S, l) → (C, S, l)
(4) ((M N)[ρ], S, l) → (M[ρ], N[ρ], S, l)
(5) ((λ.B)[ρ], S, l) → (B[ρ], S, l + 1)
(6) ((n + 1) + 1 : ρ, λ : S, l + 1) → ((n + 1)[C : ρ], S, l)
(7) (n, S, l) → (l - n, S, l)
(8) ([M,l], N[ρ], S, l') → (N[ρ], [M,l], S, l)
(9) ([B,l], λ : S, l') → ([λ.B,l], S, l')
(10) ([N,l], [M,l'], S, l'') → ([M,N,l'], S, l'')
(11) ([T,l], ϵ, l') → T

Figure 5. Cregut’s full-reducing KN with its calculus of closures λρ 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 (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 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 [Λ,l] whose meaning is explained below. The syntax suggests an implicit calculus which we name λρ. 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 level 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 l + 1 encodes the nesting of the abstraction’s formal parameter. In the 7th rule, the appropriate de Bruijn index is computed by subtracting 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 λ.((λM)0)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 λM (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).

6. Introducing the calculus of closures λρ

We introduce the λρ calculus as the natural extension of λρ that subsumes λρ.

λρ C := Λ[ρ] | Π | [n] | ΛC | ΠC
ρ := e | C : ρ

The calculus only adds two ephemeral constructors which are required for full-reduction, namely, absolute indices [n] and closure abstractions ΛC. Absolute indices are de Bruijn indices that are not relative to an environment. Absolute indices are different from closures n[e]. They assign a fixed level to free variables (open terms). Absolute indices are required for 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 contructions (hereafter ‘ephemeral closures’), gathered in E := [n] | ΛE | E E. As was the case
the structural operational semantics of normal order will guarantee
is a proper closure. This simulation property is proven
provided:

\[(\langle A, B \rangle, I + 1 : \rho \rangle, I)\]

parameters retrieved from the environment (5th clause), lifts applica-
calculates the absolute index of formal
incides with the length of the environment \(|p|\) minus the current
proper bindings
in the environment, i.e., bindings other than
the level parameter /\.

It is in the 4th rule, when going under clos-
I + 1, but does not increment
pushed on the environment, namely
now carries a lambda level parameter
I on the left of Figure 9 are notions of reduction for the new con-
—\(\beta_j\) operates on pairs \((C, N)\) rather than just closures. The rules
on the left of Figure 9 are notions of reduction for the new con-
structs and come naturally from \(\sigma_c\). \(\forall X\) is the rule for bound
variables, \(\forall X\) for lifting to closure application, \(\forall X\) for formal
closure parameters, \(\forall X\) for free variables, and \(\lambda X\) for lifting to closure
abstraction where the formal parameter (the incremented lambda
level) is pushed on the environment. The first rule on the right (\(\beta_j\))
contracts \(\beta_j\)-redices \((\langle X, B \rangle[n \rho] \cdot \rho)\) \(-\) \(N\), where the formal parameter
\(n\) that was pushed on the top of the environment by an immediately
preceding ephemeral expansion \(\lambda X\) is discarded and replaced
by the operand \(N\). The other compatibility rules (\(\mu_1\), \(\mu_2\), \(\nu_2\),
and \(\xi_2\)) are obtained by adapting to closure-level pairs the cor-
responding rules in Figure 2. A pair’s lambda level is incremented in
\((\xi_2)\), for it ‘goes under closure abstraction’, leaving B’s environ-
ment untouched.

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

Figure 7. Substitution function \(\sigma_c\).


\[
\begin{align*}
\sigma_c(C, N) & \rightarrow E \\
\sigma_c([n_1], I) & \rightarrow [n_1] \\
\sigma_c(n[p], I) & = \begin{cases} \sigma_c(n^{th}(p), I) & \text{if } n < |p| \\
0 & \text{if } n \geq |p| \end{cases} \\
\sigma_c(\langle A, B \rangle, I) & = \langle \sigma_c(A[I + 1 : \rho], I) \rangle \\
\sigma_c(\lambda X, I) & = \lambda X.\sigma_c(B, I + 1) \\
\sigma_c(R, I) & = \begin{cases} I - n & \text{if } n \geq |p| \end{cases} \\
\sigma_c([M \cdot N], I) & = \sigma_c(M, I) \cdot \sigma_c(N, I) \\
\sigma_c([M \cdot N, I] & = \sigma_c(M, I) \cdot \sigma_c(N, I)
\end{align*}
\]

Figure 8. Height of a closure.

with \(\lambda \) (Section 4), the ephemeral closures of \(\lambda \) are required to
define reduction contexts for closures (Section 6.1).

The connection between \(\lambda \) and \(\lambda \) is established by substitution
function \(\sigma_c\) (Figure 7, top) which is the analogous of function \(\sigma\)
in \(\lambda \) and simulates capture-avoiding substitution in \(\lambda \). Function \(\sigma_c\)
now carries a lambda level parameter \(I\) and enforces index align-
ment like \(\lambda N\) (Section 5). Observe that in the 3rd clause, \(\sigma_c\)
increments the level encoding the scope of the formal parameter that is
pushed on the environment, namely \(I + 1\), but does not increment
the level parameter \(I\). It is in the 4th rule, when going under clos-
ure abstraction, that the lambda level \(l\) is incremented but the en-
vironment 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 \(n < |p|\)), free
variables are given their absolute indices (2nd clause, case \(n \geq |p|\))
which are calculated by subtracting to the current index \(n\) the num-
ber of proper bindings in the environment, i.e., bindings other than
levels encoding the nesting of formal parameters. This number co-
cides with the length of the environment \(|p|\) minus the current
lambda level \(I\). Finally, \(\sigma_c\) calculates the absolute index of formal
parameters retrieved from the environment (5th clause), lifts applica-
tion closures to closure applications (6th clause), and distributes
over closure applications (7th clause).

Function \(\sigma_c\) simulates capture-avoiding substitution in \(\lambda \), that
is, \(\sigma_c(B[N[p] : \rho'], I) \rightarrow \sigma_c((\sigma_c(N[p], I)) \cdot \sigma_c(B)[\rho'], I)\), provided
that \(N[p]\) is a proper closure. This simulation property is proven
by induction on the height of \(B[N[p] : \rho']\) 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 \(N[p]\) is always a proper closure.

6.1 Structural operational semantics of normal order in \(\lambda\)

Figure 9 shows the structural operational semantics of normal
order in \(\lambda\) which we have obtained from the structural version in
\(\lambda\) (Figure 2) by adding ephemerals and \(\lambda\)'s parameters-as-levels
(Section 5). The lambda level \(I\) has to be carried along and thus
\(\rightarrow_{\beta_i}\) operates on pairs \((C, N)\) rather than just closures. The rules
on the left of Figure 9 are notions of reduction for the new con-

\[
\begin{align*}
\langle M, 0 \rangle_{\beta_j} & \rightarrow [M, 0]_{\beta_j} \\
\langle M_1, 0 \rangle_{\beta_j} & \rightarrow [M_1, 0]_{\beta_j} \\
\langle M_2, 0 \rangle_{\beta_j} & \rightarrow [M_2, 0]_{\beta_j}
\end{align*}
\]

The input term \(M\) is injected into the closure \(M[\cdot]\) abbreviated \(M\).
The closures \(M_i\) map via \(\sigma_c\) to ground terms \(M_i\) which are the result
of step-by-step normal order in \(\lambda\). The reduction relation \(\rightarrow_{\beta_i}\)
is that induced by all the rules in Figure 10 except \(\beta_j\). The reduc-
tion relation \(\rightarrow_{\beta_j}\) is that induced by all the rules on the right
column. Due to the compatibility rules, which are exactly those
which are included in ephemeral closures \(E\) but are not included in \(\lambda\)
for closure abstraction bodies in \(\lambda\) are proper closures with delayed
stubs in their environments. These environments may be en-
Hence, the combination of \(\lambda N\) and \(\beta_j\), and their closures can be
only be removed when demanded by \(\forall X\).

As discussed in Section 6, the substitution function \(\sigma_c\) (Fig-
ure 7) connects \(\lambda \) to \(\lambda \). Moreover, the connection can be es-
tablished at the step-by-step level between \(\rightarrow_{\beta_j}\) in \(\lambda\) and \(\rightarrow_{\beta_j}\) in
\(\lambda\), as illustrated by the following diagram.

The syntax for closure whnf (hereafter whnf) and closure ncs (hereafter ncs)
(Section 9.4) is shown at the bottom of Figure 9. The ncs are in-
ecluded in ephemeral closures \(E\) but are not included in \(\lambda\) be-
cause abstraction bodies in \(\lambda\) are proper closures with delayed
stubs in their environments. These environments may be en-
Hence, the combination of \(\lambda N\) and \(\beta_j\), and their closures can be
only be removed when demanded by \(\forall X\).

As discussed in Section 6, the substitution function \(\sigma_c\) (Fig-
ure 7) connects \(\lambda \) to \(\lambda \). Moreover, the connection can be es-
tablished at the step-by-step level between \(\rightarrow_{\beta_j}\) in \(\lambda\) and \(\rightarrow_{\beta_j}\) in
\(\lambda\), as illustrated by the following diagram.

\[
\begin{align*}
\langle M, 0 \rangle_{\beta_j} & \rightarrow [M, 0]_{\beta_j} \\
\langle M_1, 0 \rangle_{\beta_j} & \rightarrow [M_1, 0]_{\beta_j} \\
\langle M_2, 0 \rangle_{\beta_j} & \rightarrow [M_2, 0]_{\beta_j}
\end{align*}
\]

The input term \(M\) is injected into the closure \(M[\cdot]\) abbreviated \(M\).
The closures \(M_i\) map via \(\sigma_c\) to ground terms \(M_i\) which are the result
of step-by-step normal order in \(\lambda\). The reduction relation \(\rightarrow_{\beta_i}\)
is that induced by all the rules in Figure 10 except \(\beta_j\). The reduc-
tion relation \(\rightarrow_{\beta_j}\) is that induced by all the rules on the right
column. Due to the compatibility rules, which are exactly those

The notions of reduction \( \text{VAR}_p, \text{PAR}_p, \) and \( \text{FRE}_p \) merely implement substitution on demand and do not interfere with \( \text{(p2p)} \).

The step-by-step connection rests on the property that \( \sigma \) simulates capture-avoiding substitution in \( \lambda \) (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 \( \text{(p2p)} \) come from \( \sigma \), and since \( \sigma \) and \( \text{(p2p)} \) do not go under environments, then \( \sigma \) commutes with \( \text{whnf} \), i.e., given \( (P, 0) \to \text{whnf} (Q, 0) \) and \( (R, 0) \to \text{whnf} (S, 0) \), it is the case that \( \sigma(P, 0) = \sigma(R, 0) \) iff \( \sigma(Q, 0) = \sigma(S, 0) \).

7. From structural operational semantics to reduction-free normaliser

7.1 From structural to reduction semantics

The search functions \( \text{search\_whnf} \) and \( \text{search\_nf} \) implement the compatibility rules of the structural operational semantics of normal order in \( \lambda_p \) (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 \( \text{search} \) invokes \( \text{search\_nf} \). (From now on we omit for brevity the entry functions of all our semantics.) Function \( \text{search\_nf} \) searches for a \( \text{nf} \) or for the next redex to be contracted. It relies on \( \text{search\_whnf} \) to check if operators in applications are in whnf, or for the next redex in the call-by-name subredunction 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 whnfc-ness would only reflect it implicitly. The deriving tree above a second premiss of \( \text{(p2p)} \) will only contain call-by-name rules because \( \text{(p2p)} \) only reflect it implicitly. The derivation trees above a second premiss of \( \text{(p2p)} \) will only contain call-by-name rules because \( \text{(p2p)} \) only reflect it implicitly.

The step-by-step connection rests on the property that \( \sigma \) simulates capture-avoiding substitution in \( \lambda \) (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 \( \text{(p2p)} \) come from \( \sigma \), and since \( \sigma \) and \( \text{(p2p)} \) do not go under environments, then \( \sigma \) commutes with \( \text{whnf} \), i.e., given \( (P, 0) \to \text{whnf} (Q, 0) \) and \( (R, 0) \to \text{whnf} (S, 0) \), it is the case that \( \sigma(P, 0) = \sigma(R, 0) \) iff \( \sigma(Q, 0) = \sigma(S, 0) \).

7.2 Syntactic correspondence

The correspondence between the reduction semantics of Figure 10 and the environment-based eval/apply abstract machine of Figure 9. The parameters-as-levels and closure-converted structural operational semantics of normal order in \( \lambda_p \).
$S ::= C_0 \mid C_1(S) : S \mid C_2 : S \mid C_3(S) \\
\mid C_4(S) : S \mid C_5(S)$

$$T \rightarrow \left( T[c] \cdot C_0 \cdot T \right)_{\text{a}}$$

$$\begin{array}{ll}
\text{(if } n < \beta) & (n\{\cdot\} : S, I)_{\text{a}} \rightarrow \left( n^\beta \{\cdot\} : S, I \right)_{\text{a}} \\
\text{(if } n \geq \beta) & (n\{\cdot\} : S, I)_{\text{a}} \rightarrow \left( (n - (\beta - I) - I) : S, I \right)_{\text{a}} \\
(n, S, l)_{\text{a}} & (n, S, l)_{\text{a}} \\
((\lambda\, B)\{\cdot\} : S, I)_{\text{a}} & ((\lambda\, B)\{\cdot\} : S, I)_{\text{a}} \\
((M\, N)\{\cdot\} : S, I)_{\text{a}} & ((M\, N)\{\cdot\} : S, I)_{\text{a}} \\
(M \cdot N, S, l)_{\text{a}} & (M \cdot N, S, l)_{\text{a}} \\
(E, C_0, I)_{\text{a}} & \Rightarrow E
\end{array}$$

Figure 11. Normal order environment-based eval/apply machine with continuation stack $S$.
Figure 12. Natural semantics of normal order in $\lambda_p$.

\[
\begin{align*}
\frac{n < |p|}{(n[p], l) \downarrow^N \frac{N}{p}} \quad \text{(VAR)} \\
\quad \frac{n \geq |p|}{(n[p], l) \downarrow^N \frac{n - (|p| - l)}{p} \quad \text{(PRE)} \\
\quad \frac{\langle \lambda. B \rangle[p], l \downarrow^N \frac{B[l+1 : p]}{l + 1 : p} \quad \text{(LAM)} \\
\quad \frac{\langle \lambda. B \rangle[p], l \downarrow^N \frac{B[l+1 : p]}{l + 1 : p} \quad \text{(ABS)} \\
(\lambda. B)[p], l \downarrow^N \frac{\lambda. B \downarrow^N \frac{n}{p}}{\lambda. B \downarrow^N \frac{n}{p}} \quad \text{(RED)} \\
\quad \frac{\langle M \rangle[p] \cdot \langle N \rangle[p], l \downarrow^N \frac{C}{l + 1 : p} \quad \text{(APP)} \\
\quad \frac{\langle M \rangle[p] \cdot \langle N \rangle[p], l \downarrow^N \frac{C}{l + 1 : p} \quad \text{(APP)} \\
(\lambda. B)[p], l \downarrow^N \frac{\lambda. B \downarrow^N \frac{n}{p}}{\lambda. B \downarrow^N \frac{n}{p}} \quad \text{(RED)} \\
\quad \frac{\langle M \rangle[p] \cdot \langle N \rangle[p], l \downarrow^N \frac{C}{l + 1 : p} \quad \text{(APP)} \\
\end{align*}
\]

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

\[
\begin{align*}
\frac{n < |p|}{(n[p], l) \downarrow^N \frac{N}{p}} \quad \text{(VAR)} \\
\quad \frac{n \geq |p|}{(n[p], l) \downarrow^N \frac{n - (|p| - l)}{p} \quad \text{(PRE)} \\
\quad \frac{\langle \lambda. B \rangle[p], l \downarrow^N \frac{B[l+1 : p]}{l + 1 : p} \quad \text{(LAM)} \\
\quad \frac{\langle \lambda. B \rangle[p], l \downarrow^N \frac{B[l+1 : p]}{l + 1 : p} \quad \text{(ABS)} \\
(\lambda. B)[p], l \downarrow^N \frac{\lambda. B \downarrow^N \frac{n}{p}}{\lambda. B \downarrow^N \frac{n}{p}} \quad \text{(RED)} \\
\quad \frac{\langle M \rangle[p] \cdot \langle N \rangle[p], l \downarrow^N \frac{C}{l + 1 : p} \quad \text{(APP)} \\
\quad \frac{\langle M \rangle[p] \cdot \langle N \rangle[p], l \downarrow^N \frac{C}{l + 1 : p} \quad \text{(APP)} \\
\end{align*}
\]

Figure 14. Shortcut natural semantics of normal order in $\lambda_p$. 

\[
\begin{align*}
\frac{(M, l) \downarrow^N \frac{M'}{M} \neq \lambda. B \downarrow^N \frac{n}{p}}{(M \cdot N, l) \downarrow^N \frac{M' \cdot N'}{M' \cdot N}} \quad \text{(NEU)} \\
\quad \frac{(M, l) \downarrow^N \frac{M'}{M} \neq \lambda. B \downarrow^N \frac{n}{p}}{(M \cdot N, l) \downarrow^N \frac{M' \cdot N'}{M' \cdot N}} \quad \text{(NEU)} \\
\end{align*}
\]
between explicit control and the continuation stack can be ob-
erved, and the second premise of \( \text{NEU}_5 \) and of \( \text{NEU}_6 \) checks if \( M' \) is a ground term.

The underlying calculus, which we call \( \lambda^*_p \), is an optimised variant of \( \lambda_p \) that omits the levels in ground terms \( [A] \):

\[
\begin{align*}
l_p & := \Lambda[A] \mid n \mid [A] \\
c_p & := \epsilon \mid C : \rho
\end{align*}
\]

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

9.1 A reduction-free normaliser with explicit control

The mutually recursive \( \psi_{c.ctl} \) and \( \psi_{ctl} \) of the shortcut natural semantics in Figure 14 differ in the treatment of abstractions. Rule \( \text{LAM}_{ct.ctl} \) takes place when the abstraction is applied to an operand whereas \( \text{LAM}_{ctl} \) takes place when the abstraction is unapplied. We transform the shortcut normalisers \( \text{normalise16}_{\text{w.hnf}} \) and \( \text{normalise16}_{\text{nf}} \) into a single \( \text{normalise}_{ctl} \) normaliser with explicit control that encodes the different treatment of abstractions.

We introduce the control characters \( \tau \) that respectively encode the different treatment of abstractions. We apply defunctionalisation and CPS transformation to the norm-
aliser with explicit control to obtain the eval/apply machine with explicit control.

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 \( \text{normalise}_{ctl\_cont} \) and \( \text{apply}_{ctl\_cont} \) 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 \( e \) 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 \( w \) when reducing operators in applications and the use of \( n \) when reducing operands in neutral closures. Observe that continuation \( C_1(C, e) \) 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 \( w \) when continuation \( C_1 \) is pushed on the stack. The remaining transitions just preserve the current control, except for the transition dealing with operands in neutral closures.

\[
\begin{align*}
S & := C_0 \mid C_1(C, e) : S \mid C_2 : S \mid C_3(C) : C
\end{align*}
\]

\[
\begin{align*}
c & := w \mid n
\end{align*}
\]

Figure 15. Natural semantics of normal order in \( \lambda_p \) with explicit control.

\[
\begin{align*}
\text{LAM}_{ct.ctl} & \rightarrow \text{LAM}_{ctl}
\end{align*}
\]

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

where the machine uses \( n \) and pushes \( C_3 \) on the stack, signalling the point at which call-by-name ends and normal order resumes. Consequently, control character \( w \) can be replaced by checking for the occurrence of \( C_1 \) on the top of the stack, and control character \( \rho \) 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]:

\[
\begin{align*}
T & \rightarrow (T', e, C_0, 0, n)
\end{align*}
\]

\[
\begin{align*}
\text{RED}_{ct} & \rightarrow \text{RED}_{ctl}
\end{align*}
\]

Pattern-matching on the stack breaks the shallow inspection required to defunctionalise 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 \( \text{normalise20}_{\text{cont}} \) and \( \text{apply20}_{\text{cont}} \) 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
terms going to apply:

\[
\begin{align*}
\text{if n} \leq |p| & \quad \text{then} & \quad (\lambda B)[p], C_1(N) : S, l & \rightarrow (B[l + 1 : p], C_1(N) : S, l) \\
\text{if n} \geq |p| & \quad \text{then} & \quad (\{M\}, C_1(N) : S, l) & \rightarrow (N, C_2([M]) : S, l) \\
\end{align*}
\]

The rest of the transitions remain the same and are omitted. The protruded machine is implemented in the code by functions normalise22_push and apply2l_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 (ζs) 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 for lookup instead of a recursive peeling-off of the environment (\text{\kappa} for lookup in the constructor of neutral closures) and a recursive peeling-off of lambda scopes. Second, the presence of defunctionalised continuations coming from the stack S defined in Figure 17.

We remove the visual differences in Figure 19. The function for lookup is replaced by a lookup definition (consequently, the transition for free variables \epsilon to be adapted). And defunctionalised continuations in S are replaced by the constructors of \lambda (and the control character \lambda) that they represent (collected in stack S in Figure 19). The machine is implemented by function normalise22_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 scopes (rules \lambda Mp and (\xi_p) in Figure 9) results in index alignment and balanced derivations. The differences between our work and Curien’s machine are substantial.

- The precise control of levels in \lambda scopes (rules \lambda Mp and (\xi_p) 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 $\beta$-reduction, not the $\beta_v$-reduction of the lambda-calculus. The strategy implements strict semantics for redices, but performs $\beta$-reduction, not the $\beta_v$-reduction of the lambda-calculus.

The closure calculi $\lambda_0$ and $\lambda_2$ we have introduced are rather natural extensions of $\lambda_2$, as illustrated by the following diagram:

---

We have not defined the reduction theory of $\lambda_2$, only presented the reduction strategy $\tilde{\pi}_0$, 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 dTPP 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