Optimizing a Non-Deterministic Abstract Machine with Environments

Małgorzata Biernacka
Institute of Computer Science, University of Wrocław, Poland

Dariusz Biernacki
Institute of Computer Science, University of Wrocław, Poland

Sergueï Lenglet
Université de Lorraine, Nancy, France
Université Sorbonne Paris Nord, France

Alan Schmitt
INRIA, France

Abstract

Non-deterministic abstract machine (NDAM) is a recent implementation model for programming languages where one must choose among several redexes at each reduction step, like process calculi. These machines can be derived from a *zipper semantics*, a mix between structural operational semantics and context-based reduction semantics. Such a machine has been generated also for the λ-calculus without a fixed reduction strategy, i.e., with the full non-deterministic β-reduction.

In that machine, substitution is an external operation that replaces all the occurrences of a variable at once. Implementing substitution with environments is more low-level and more efficient as variables are replaced only when needed. In this paper, we define a NDAM with environments for the λ-calculus without a fixed reduction strategy. We also introduce other optimizations, including a form of refocusing, and we show that we can restrict our optimized NDAM to recover some of the usual λ-calculus machines, e.g., the Krivine Abstract Machine. Most of the improvements we propose in this work could be applied to other NDAMs as well.

2012 ACM Subject Classification Theory of computation → Abstract machines

Keywords and phrases Abstract machine, Explicit substitutions, Refocusing

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.11

Related Version *Extended Version*: `https://inria.hal.science/hal-04568253` [15]

Supplementary Material *Software (Implementation of the Optimized Abstract Machine)*: `https://gitlab.inria.fr/skeletons/lambda-am.git` [13]
archived at `swh:1:dir:ba776d5226cde3b68cc6346fcd3a3495ad4d04e8`

Funding This work is partially funded by the National Science Centre of Poland under grant no. 2019/33/B/ST6/00289, and PHC Polonium.

Acknowledgements We thank the anonymous reviewers for their comments.

1 Introduction

Abstract machines are first-order tail-recursive transition systems for term reduction. Known examples of machines for languages based on the λ-calculus include SECD [39], CEK [32], and the KAM [38]. They serve as an operational semantics [39, 32, 31] or an implementation model [36, 41] of programming languages, but are also relevant in proof theory [38], higher-order model checking [46], or as cost models [10], to name but a few applications.

Abstract machines for the λ-calculus usually implement a *deterministic* strategy which selects a specific redex at each step [27, 31]. In contrast, a Non-Deterministic Abstract Machine (NDAM) [14] may choose any of the available redexes: it explores the term, making

an arbitrary choice when several are possible, e.g., going left or right of an application. If the machine focuses on a subterm which cannot be further reduced, it annotates it and backtracks to make a different choice. Annotations prevent the machine from visiting an already explored subterm.

NDAMs have been originally developed as an implementation model for non-deterministic languages, in particular, for process calculi. However, considering an NDAM for the λ-calculus without a fixed reduction strategy is worthwhile too, at least from the theoretical point of view – one can expect that such an NDAM should subsume various deterministic reduction strategies or abstract machines, and, therefore, play a role of a unifying framework in which one can study and compare them. Moreover, such an abstract machine should be readily adjustable to account for the λ-calculus extended with concurrency primitives.

Defining an NDAM can be hard, so the framework in [14] comes with a sound and complete automatic derivation procedure from another semantics format called *zipper semantics*, which is a middle ground between a Structural Operational Semantics (SOS) [44] and a context-based reduction semantics [32, 26]. Like an SOS, a zipper semantics explores a term with structural rules, but it remembers its position in the term using an evaluation context, i.e., a term with a hole [32]. Zipper semantics and their derived NDAMs have been presented for process calculi [47, 40], but also for the λ-calculus without a fixed reduction strategy.

As a result of the generic derivation procedure, the NDAM for the λ-calculus cannot be perfectly tailored for it, and, moreover, it inherits some of the artifacts from the underlying zipper semantics. In particular, it relies on substitution, an external operation which replaces all the occurrences of a variable at once, traversing the entire term. A more elementary and efficient implementation of β-reduction that is traditionally used in abstract machines uses environments, i.e., delayed substitutions, where a variable is looked up only when needed, and closures are built [39]. The goal of this work is to obtain an environment-based NDAM for the λ-calculus without a fixed reduction strategy.

To this end we first define a zipper semantics with environments for the λ-calculus without a fixed reduction strategy. It is inspired by $\lambda\sigma_{\Uparrow}$, a classic calculus with explicit substitutions [25]. It turns out that we only need to implement a small subset of the rewrite rules of Curien et al.'s calculus in our zipper semantics. Then, the NDAM automatically derived from the zipper semantics itself uses environments.

Unfortunately, the derived NDAM still suffers from performance issues. For example, it is not in refocused form [27, 28]: after a β-reduction, the NDAM reconstructs the whole term before looking for the next redex. The reason is that an NDAM should be able to find any redex, even the ones next to the root. As a compromise between efficiency and completeness, we let the machine reconstruct an arbitrary part of the term from the contraction site. It does not reconstruct the whole term after each β-reduction, but it may still reconstruct enough of it so that it can find β-redexes closer to the root. We also propose other optimizations, in particular related to the way the NDAM manages annotations.

The end product is an optimized NDAM for the λ-calculus with environments. By restricting its behavior, we are then able to express some of the existing λ-calculus machines or strategies, as we expected. For example, if we forbid the exploration step which goes under a λ-abstraction, we restrict ourselves to weak strategies. In addition, if we always go left of an application, we simulate the KAM [38]. We can also encode more complex strategies like full normal order by enforcing priorities on the machine steps of the optimized NDAM. We thus demonstrate that NDAMs can be brought closer to usual, more efficient abstract machines designed for a specific reduction strategy.

Contributions. Our contributions are as follows:

- We define a zipper semantics with environments for the λ-calculus, using a subset of a known calculus with explicit substitutions.
- From the zipper semantics we derive a correct-by-construction non-deterministic abstract machine for the λ-calculus with environments and without a fixed reduction strategy.
- We optimize, in a provably-correct way, the derived NDAM with optimization techniques that we devise to improve the performance of NDAMs in general, which are either generic, like a form of refocusing, or language-specific, like a better management of annotations.
- We provide an OCaml implementation of the optimized machine to experiment with.
- We show that the optimized NDAM we obtain is able to express many of the existing machines designed for a specific reduction strategy.

Structure of the paper. Section 2 recalls the definition of the zipper semantics with substitution, which we characterize with a zipper semantics with environments. In Section 3, we first present the NDAM derived from the environment-based zipper semantics, and then define a more optimized machine. In Section 4, we show how to obtain previously known machines from the optimized NDAM. Section 5 discusses related work and Section 6 concludes the paper. The accompanying report [15] contains the proofs missing from the paper.

2 Zipper Semantics

We present the zipper semantics of the λ-calculus with environments and prove its equivalence with the substitution-based variant.

2.1 Syntax and Substitution-based Zipper Semantics

We present the syntax of the λ-calculus where variables are represented with de Bruijn indices. We let n, m, p range over indices, i.e., natural numbers, and define the syntax of λ-calculus terms, ranged over with t and s, as follows.

$$t, s ::= \lambda.t \mid n \mid t\ s$$

A de Bruijn index counts how many λ-abstractions separate the variable from its binder. For example, using more familiar named variables, $\lambda.0$ stands for $\lambda x.x$, and $\lambda.\lambda.1$ for $\lambda x.\lambda y.x$. We use a de Bruijn representation for the environment-based calculus (Section 2.2), to prevent unwanted name captures when moving an environment inside a λ-abstraction. We write $t\{s/n\}$ for the capture-avoiding substitution of n with s in t, defined as follows, using the auxiliary renaming operation on indices $\langle t\rangle_p^n$ [29].

$$(t_1\ t_2)\{s/n\} \triangleq t_1\{s/n\}\ t_2\{s/n\} \qquad \langle t_1\ t_2\rangle_p^n \triangleq \langle t_1\rangle_p^n\ \langle t_2\rangle_p^n$$

$$(\lambda.t)\{s/n\} \triangleq \lambda.t\{s/n+1\} \qquad \langle \lambda.t\rangle_p^n \triangleq \lambda.\langle t\rangle_{p+1}^n$$

$$m\{s/n\} \triangleq \begin{cases} m-1 & \text{if } m > n \\ \langle s\rangle_0^n & \text{if } m = n \\ m & \text{if } m < n \end{cases} \qquad \langle m\rangle_p^n \triangleq \begin{cases} m+n & \text{if } m \geq p \\ m & \text{if } m < p \end{cases}$$

The goal of zipper semantics is to go through a term looking for a redex, using an evaluation context to remember the current focus in the term. We represent evaluation contexts $\mathbb{E}$, $\mathbb{F}$ as stacks of elementary contexts $\mathfrak{F}$, called *frames*.

$$\mathbb{E}, \mathbb{F} ::= \mathfrak{F} :: \mathbb{E} \mid \blacksquare \qquad \mathfrak{F} ::= \square\ t \mid t\ \square \mid \lambda$$

$$\textsf{appL}\ \frac{t \xrightarrow{\square\, s :: \mathbb{E}}_{\mathsf{ev}} t'}{t\ s \xrightarrow{\mathbb{E}}_{\mathsf{ev}} t'} \qquad \textsf{appR}\ \frac{s \xrightarrow{t\, \square :: \mathbb{E}}_{\mathsf{ev}} s'}{t\ s \xrightarrow{\mathbb{E}}_{\mathsf{ev}} s'} \qquad \textsf{lam}\ \frac{t \xrightarrow{\lambda :: \mathbb{E}}_{\mathsf{ev}} t'}{\lambda.t \xrightarrow{\mathbb{E}}_{\mathsf{ev}} t'} \qquad \textsf{lam}\beta\ \frac{}{\lambda.t \xrightarrow{\square\, s :: \mathbb{E}}_{\mathsf{ev}} \mathbb{E}\langle t\{s/0\}\rangle}$$

Figure 1 Zipper semantics of the substitution-based lambda-calculus.

In defining the zipper semantics, it is convenient to interpret contexts *inside-out*: the topmost frame represents the innermost construct. We define a plugging operation for frames $\mathfrak{F}\langle t\rangle$ and contexts $\mathbb{E}\langle t\rangle$ as follows.

$$(\square\ s)\langle t\rangle \triangleq t\ s \qquad (s\ \square)\langle t\rangle \triangleq s\ t \qquad \lambda\langle t\rangle \triangleq \lambda.t$$
$$\blacksquare\langle t\rangle \triangleq t \qquad (\mathfrak{F} :: \mathbb{E})\langle t\rangle \triangleq \mathbb{E}\langle \mathfrak{F}\langle t\rangle\rangle$$

The definition of the zipper semantics is in Figure 1. Informally, $t \xrightarrow{\mathbb{E}}_{\mathsf{ev}} t'$ means that $\mathbb{E}\langle t\rangle$ reduces to t'. The rules appL, appR, and lam are going through the term looking for a redex. For example in appL, $t\ s$ reduces to t' in $\mathbb{E}$ if t reduces to t' in $\square\ s :: \mathbb{E}$. We see that the focus changes from $t\ s$ to t, resulting in pushing $\square\ s$ on top of $\mathbb{E}$. When we focus on a λ-abstraction inside a context $\square\ s :: \mathbb{E}$, we recognize a redex, and we can β-reduce with the axiom lamβ.

A reduction step is defined as a transition $t \xrightarrow{\blacksquare}_{\mathsf{ev}} t'$. The semantics of Figure 1 is a slight optimization of the original zipper semantics for the λ-calculus [14], where the axiom lamβ is split in two rules. It is however straightforward to prove that the two zipper semantics coincide, and therefore our semantics also characterizes the usual context-based reduction semantics of the λ-calculus [14].

2.2 Environment-based Zipper Semantics

Our goal is to replace substitution with environments, while still being able to reduce an arbitrary β-redex, including under a λ-abstraction. To this end, we take inspiration from $\lambda\sigma_{\Uparrow}$ [25], a calculus with explicit substitutions and strong reduction. The syntax of *environments* ϵ, ϕ and of *closures* c, d are as follows.

$$\epsilon, \phi ::= \mathtt{id} \mid \uparrow \mid l \cdot \epsilon \mid \epsilon \circ \phi \mid \Uparrow\epsilon \qquad c, d ::= l[\epsilon] \mid \lambda.c \mid c\ d \qquad l, k ::= t \mid c$$

Closures extend terms by pairing them with an environment $t[\epsilon]$. Environments are like explicit substitutions in $\lambda\sigma_{\Uparrow}$. An environment $l \cdot \epsilon$ maps 0 to l and the variables $n \geq 1$ are mapped by ϵ. The identity environment $\mathtt{id}$ maps variables to themselves. The composition $\epsilon \circ \phi$ is such that ϵ is applied first, and then ϕ.

Whenever an environment ϵ is applied to the body of a λ-abstraction $\lambda.l$, it is *lifted* $\Uparrow\epsilon$ to avoid index clashes. Informally, ϵ should not be applied to 0 in l, as 0 is bound by the λ, and the occurrences of 0 in the image of ϵ should not be captured by that λ. To avoid this, we *shift* ($\uparrow$) the result of the application of ϵ. To summarize, $\Uparrow\epsilon$ acts like $0 \cdot \epsilon \circ \uparrow$: the variable 0 is mapped to 0, and any $n \geq 1$ is mapped by ϵ and the result is then shifted. This reminds that the lift construct is not mandatory in calculi with explicit substitutions, as its behavior can be expressed with the other constructs [1, 25]: we use it because it is more convenient in proofs, in particular to prove termination results.

The zipper semantics for the environment-based λ-calculus is defined in Figure 2. It takes as source a term or a closure l, but always results in a closure c. As in the substitution-based calculus, we use a context to remember the current position when exploring l. With

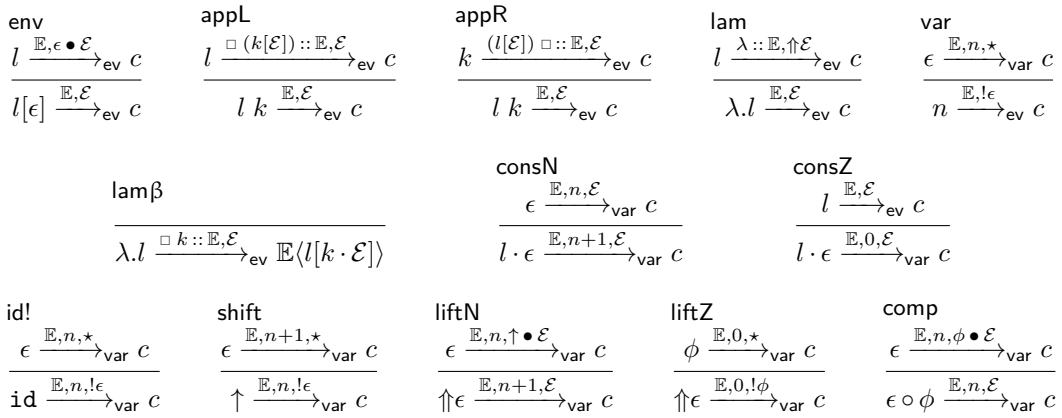

Figure 2 Zipper semantics for the environment-based λ-calculus.

environments, context frames are built from terms or closures: $\mathfrak{F} ::= \square\, l \mid l\, \square \mid \lambda$. An evaluation context $\mathbb{E}$, $\mathbb{F}$ is still a stack of frames interpreted inside-out, and we plug terms or closures inside a frame or a context. The plugging operation is defined as in Section 2.1.

The zipper semantics uses an extra argument to record what the current environment is while exploring l. To this end, we define an option type $\mathcal{E}$, $\mathcal{F}$, which we call *local environment*, signalling whether the environment is defined or not.

$$\mathcal{E}, \mathcal{F} ::= \star \mid !\epsilon$$

When $\mathcal{E} = \star$, the local environment is not (yet) defined. Although $\star$ could be seen as $\mathtt{id}$, we distinguish the two to make explicit the absence or presence of an environment. We extend the following constructs to local environments and overload some of the symbols. The rationale behind these definitions is explained when we detail the rules of the zipper semantics.

$$l[\star] \triangleq l \qquad l \cdot \star \triangleq l \cdot \mathtt{id} \qquad \epsilon \bullet \star \triangleq\, !\epsilon \qquad \Uparrow\star \triangleq \star$$
$$l[!\epsilon] \triangleq l[\epsilon] \qquad l \cdot\, !\epsilon \triangleq l \cdot \epsilon \qquad \epsilon_1 \bullet\, !\epsilon_2 \triangleq\, !\epsilon_1 \circ \epsilon_2 \qquad \Uparrow !\epsilon \triangleq\, !\Uparrow\epsilon$$

Zipper semantics transitions are of two kinds: $l \xrightarrow{\mathbb{E},\mathcal{E}}_{\mathsf{ev}} c$ means that l with local environment $\mathcal{E}$ inside $\mathbb{E}$ is reducing to c, i.e., $\mathbb{E}\langle l[\mathcal{E}]\rangle$ is β-reducing to c. The transition $\epsilon \xrightarrow{\mathbb{E},n,\mathcal{E}}_{\mathsf{var}} c$ means that solving n in the environment $\epsilon \bullet \mathcal{E}$ results in a term or closure which itself reduces within $\mathbb{E}$ to c: $\mathbb{E}\langle n[\epsilon \bullet \mathcal{E}]\rangle$ is β-reducing to c. The rules defining the semantics implement some of the rewriting rules of $\lambda\sigma_{\Uparrow}$.

The transition $l \xrightarrow{\mathbb{E},\mathcal{E}}_{\mathsf{ev}} c$ is going through l until it finds a β-redex, keeping track of the current environment $\mathcal{E}$ while doing so. We start with an empty context and an undefined environment $l \xrightarrow{\blacksquare,\star}_{\mathsf{ev}} c$. When we focus on a construct $l[\epsilon]$ (rule env), we set the local environment to $!\epsilon$ if it was undefined; otherwise, we compose it with ϵ. The definition of $\bullet$ reflects these two cases.

In the application case (rules appL and appR), we pack the local environment with the term or closure pushed on the context. In the body of a λ-abstraction, the local environment becomes $\Uparrow\mathcal{E}$ (rule lam): if it is undefined ($\mathcal{E} = \star$), it remains so in the body of the abstraction, which explains why $\Uparrow\star$ is defined as $\star$. Otherwise, $\mathcal{E} = !\epsilon$, and ϵ is lifted. For a variable n

$$
\begin{array}{lll}
[\![l[\epsilon_1]]\!](\epsilon_2) \triangleq [\![[\![l]\!](\epsilon_1)]\!](\epsilon_2) & [\![0]\!](l\cdot\epsilon) \triangleq [\![l]\!] & [\![n+1]\!](l\cdot\epsilon) \triangleq [\![n]\!](\epsilon) \\
[\![l_1\ l_2]\!](\epsilon) \triangleq [\![l_1]\!](\epsilon)\ [\![l_2]\!](\epsilon) & [\![0]\!](\Uparrow\epsilon) \triangleq 0 & [\![n+1]\!](\Uparrow\epsilon) \triangleq [\![n]\!](\epsilon\circ\uparrow) \\
[\![\lambda.l]\!](\epsilon) \triangleq \lambda.[\![l]\!](\Uparrow\epsilon) & [\![n]\!](\mathtt{id}) \triangleq n & \\
[\![n]\!](\epsilon_1\circ\epsilon_2) \triangleq [\![[\![n]\!](\epsilon_1)]\!](\epsilon_2) & [\![n]\!](\uparrow) \triangleq n+1 &
\end{array}
$$

Figure 3 Translation from closures to terms.

(rule var), $\mathcal{E}$ should be defined as $!\epsilon$ for some ϵ, so that we can resolve n in ϵ using the var transition, explained below. Finally, if we focus on a λ-abstraction $\lambda.l$ in the appropriate context $\square\ k :: \mathbb{E}$, we can β-reduce, which consists in pushing k on top of $\mathcal{E}$; if $\mathcal{E} = \star$, then this operation creates the environment $k\cdot\mathtt{id}$ (rule lamβ).

The transition $\epsilon \xrightarrow{\mathbb{E},n,\mathcal{E}}_{\mathsf{var}} c$ resolves n in ϵ, using $\mathcal{E}$ to handle compositions. Initially in rule var, $\mathcal{E}$ is set to $\star$. If $\epsilon = \epsilon_1\circ\epsilon_2$, then we focus on ϵ_1 and remember ϵ_2 in $\mathcal{E}$ (rule comp). More generally, the transition $\epsilon \xrightarrow{\mathbb{E},n,\star}_{\mathsf{var}} c$ is solving n in ϵ, while $\epsilon \xrightarrow{\mathbb{E},n,!\phi}_{\mathsf{var}} c$ is solving n in $\epsilon\circ\phi$. For example, the rule shift says that resolving n in $\uparrow\circ\phi$ consists in resolving $n+1$ in ϕ. Similarly, solving n in $\mathtt{id}\circ\phi$ requires solving n in ϕ (rule id!).

As explained before, $\Uparrow\epsilon$ maps 0 to 0 (rule liftZ) and any $n \geq 1$ is solved in ϵ and then shifted (rule liftN). The rules consN and consZ deal with environments $l\cdot\epsilon$ as expected: if $n \geq 1$, then we look for $n-1$ in ϵ. Otherwise, l is the expected result, and we switch to a ev transition to continue the search for a redex.

The var transition is not defined when $\mathcal{E} = \star$ and the environment is $\mathtt{id}$, $\uparrow$, or $\Uparrow\epsilon$ with $n = 0$, i.e., in the cases producing a variable (respectively n, $n+1$, and 0). The zipper semantics is solving a variable to find a β-redex, and turning a variable into another variable does not help in that regard.

2.3 Properties of the Environment-based Zipper Semantics

We state the main properties of the environment-based zipper semantics. We first show it corresponds to the substitution-based semantics. To this end, in Figure 3 we define a translation $[\![l]\!](\epsilon)$ from closures to terms: $[\![l]\!](\epsilon) = t$ means that the term or closure l under environment ϵ translates into the term t. The translation is defined first by case analysis on the term or closure l, to push the environment ϵ to the leaves of l. At the level of variables, it then computes how an index n is translated by case analysis on the environment.

We write $[\![l]\!]$ as a shorthand for $[\![l]\!](\mathtt{id})$. The equations defining the translation contain the rewrite rules we need from the ones defining $\lambda\sigma_{\Uparrow}$. We can derive some of the remaining rules of $\lambda\sigma_{\Uparrow}$ as properties of the translation: for example, we can show that $[\![l]\!](\Uparrow\mathtt{id}) = [\![l]\!](\mathtt{id})$ [15, Lemma 7] or that $[\![l]\!](\epsilon\circ\phi) = [\![[\![l]\!](\epsilon)]\!](\phi)$ [15, Lemma 12]. The proofs are usually by induction on the size of pairs (l,ϵ), strictly decreasing between the left-hand side and right-hand side of each equation defining the translation [15, Appendix A]. The size is the same as in the proof of termination of rewriting in $\lambda\sigma_{\Uparrow}$ [25, Proposition 4.2], and relies on the fact that we use lift; the proof would be more complex without lift [1].

The main result relates β-redexes on both sides.

▶ **Lemma 1.** *For all l, k, ϵ, $[\![l[k\cdot\epsilon]]\!] = [\![l]\!](\Uparrow\epsilon)\{[\![k]\!]/0\}$.*

Curien et al. prove a similar result when relating $\lambda\sigma_{\Uparrow}$ to the plain λ-calculus [25, Proposition 4.11]: rephrased in our setting, they prove that given two terms t and s, $[\![t[s \cdot \mathtt{id}]]\!] = t\{s/0\}$. We generalize their result to any closures l, k and any environment ϵ. With this result, we can relate the two zipper semantics.

▶ **Theorem 2.** *If* $l \xrightarrow{\blacksquare,\star}_{\mathsf{ev}} c$, *then* $[\![l]\!] \xrightarrow{\blacksquare}_{\mathsf{ev}} [\![c]\!]$.
For all $t \xrightarrow{\blacksquare}_{\mathsf{ev}} s$, *there exists* c *such that* $t \xrightarrow{\blacksquare,\star}_{\mathsf{ev}} c$ *and* $[\![c]\!] = s$.

The environment-based semantics is equivalent to the substitution-based one, which itself characterizes a context-based semantics of the λ-calculus [14].

The environment-based zipper semantics also satisfies the properties required so it can be derived into an NDAM [14]. In particular, it must be *machine constructive*: in each rule, it is possible to construct the terms in the premise from those of the conclusion. The semantics must also be *terminating*, ensuring that the search for a redex in the abstract machine does not loop infinitely. To prove it, it is enough to exhibit a strictly decreasing size of transitions such that the size of the premise of each inductive rule is smaller than that of its conclusion. The definition of such a size is already necessary to prove Theorem 2 [15, Appendix A]. We point out that the size is strictly decreasing because we distinguish $\star$ from $\mathtt{id}$, which allows us to give different sizes for both. Replacing $\star$ with $\mathtt{id}$ would make the zipper rule $\mathsf{id!}$ in Figure 2 not terminating.

The original theory [14] requires the zipper semantics to be *reversible*, which is the reverse of being machine constructive: for each rule, the terms in the conclusion can be deduced from those in the premise. This property makes it easier to design the backtracking mechanism of the derived machine. The zipper semantics of Figure 2 is not reversible: in rules consN and consZ, respectively l and ϵ in the conclusion do not occur in the premise. We also need to inverse the operator $\bullet$ in rules env, liftN, and comp. We show in Section 3.1 that we only need a small change in the derivation procedure to allow backtracking for these rules. However, we argue that the derivation procedure could easily be adapted to account for non-reversible zipper semantics in many cases.

3 Non-Deterministic Abstract Machines

We discuss the shortcomings of the NDAM derived from the environment-based semantics, and present an optimized abstract machine dealing with these issues.

3.1 Derived Abstract Machine

A zipper semantics is not yet an implementation, as it does not explain what to do when several rules can be applied during the search for a redex, like in the application or λ-abstraction cases. The automatically derived abstract machine (DAM in short) augments the zipper semantics with a backtracking mechanism and annotations to handle these cases.

Roughly, each zipper rule becomes a *forward* step, which focuses on a subterm. When several rules can be applied, the machine chooses arbitrarily one of them, e.g., going left or right of an application. Because of this non-determinism, it can make a wrong choice and focus on a subterm which cannot reduce. In such a case, the machine *annotates* the subterm as a normal form, and then switches to a *backward* mode, in which it undoes enough steps to return to the last configuration it made a choice, to make a different one if possible. The forward steps may focus only on subterms without annotations, ensuring that the machine cannot make the same (wrong) choice twice. The DAM relies on a stack π to remember in which order the forward steps have been applied. The backward mode then uses this stack to undo the steps in the reverse order.

Formally, we let Σ, Δ range over annotations, and we extend the syntax of terms, closures, and environments with annotations.

$$\Sigma, \Delta ::= \emptyset \mid \mathsf{ev} \mid \mathsf{var} \qquad t, s ::= n^{\Sigma} \mid \lambda^{\Sigma}.t \mid t \mathbin{@^{\Sigma}} s$$

$$\epsilon, \phi ::= \mathtt{id}^{\Sigma} \mid \uparrow^{\Sigma} \mid l \overset{\Sigma}{\cdot} \epsilon \mid \epsilon \overset{\Sigma}{\circ} \phi \mid \Uparrow^{\Sigma} \epsilon \qquad c, d ::= l[\epsilon]^{\Sigma} \mid \lambda^{\Sigma}.c \mid c \mathbin{@^{\Sigma}} d$$

We use l and k to range over (annotated) terms and closures. The machine annotates a term or closure with ev and an environment with var. We write $\mathsf{an}(l)$ or $\mathsf{an}(\epsilon)$ for the topmost annotation of l or ϵ: for instance $\mathsf{an}(l \mathbin{@^{\Sigma}} k) = \Sigma$. We write $l^{+\mathsf{ev}}$ and $\epsilon^{+\mathsf{var}}$ for the operation of annotating the topmost construct of l or ϵ: for example, $(l \mathbin{@^{\Sigma}} k)^{+\mathsf{ev}} = l \mathbin{@^{\mathsf{ev}}} k$ independently of Σ. Contexts in machine configurations are built of annotated terms, closures, and environments.

The syntax of the stack π is of the shape $\pi ::= \rho(\tilde{e}) :: \pi \mid \blacktriangle$ where $\blacktriangle$ represents the empty stack, ρ ranges over the names of the zipper rules in Figure 2, and $\tilde{e}$ represents information stored in the stack, which might be needed to make backtracking possible. In most cases, $\tilde{e}$ is empty.

The DAM is defined in Figure 4; we discuss some of the steps here. Machine configurations $\langle l \mid \mathbb{E}, \mathcal{E}, \pi \rangle_{\mathsf{ev}}$ and $\langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, \pi \rangle_{\mathsf{var}}$ reflect the transitions of the zipper semantics. Each zipper rule is turned into a forward step, where the conclusion and premise of the rule are respectively the source and target of the step. The stack π also records the rule being applied. For example, the rules appL, lam and consN translate as steps D1, D3, and D14, respectively. Each forward step has a side condition which checks that the new subterm under focus is not annotated, thus ensuring it has not already been visited.

If no forward step applies to the current term, then a switching step annotates it and changes to the backward mode bev or bvar. The configurations $\langle \pi \mid l, \mathbb{E}, \mathcal{E} \rangle_{\mathsf{bev}}$ and $\langle \pi \mid \epsilon, \mathbb{E}, n, \mathcal{E} \rangle_{\mathsf{var}}$ then discriminate on π to undo the last applied rule. Each forward step has a mirroring backward step to undo it. The switching steps are steps D7 and D21 for the ev and var modes respectively, and the backward steps corresponding to appL, lam and consN are steps D8, D11, and D22, respectively.

Finally, when the machine finds a redex, it applies the axiom $\mathsf{lam}\beta$ (step D6). The machine constructs the result of the β-reduction plugged inside $\mathbb{E}$. After reduction, some annotations may no longer be valid, so to be safe, the DAM erases all of them: the operation $|l|$ sets every annotation in l to $\emptyset$.

Reversibility. In the the original derivation procedure [14], the role of π is mainly to record which rules are being applied, to know in which order to undo them if needed. We go beyond that design in this paper, and use π to also store some information to be able to backtrack even in the case of non-reversible zipper rules. We argue that what we do is generic enough to be added to the original derivation procedure.

The rules consN, consZ, and liftZ are not reversible because some entities of the conclusion of the rule do not occur in its premise. It is easy to automatically collect such missing entities, and record them in the stack π. For example, the steps D14 and D15 – corresponding to consN and consZ – remember in π the part of the environment which is discarded, either its head or its tail.

A rule may be not reversible also because it relies on an external operator which is not injective, like $\bullet$. In that case, we need to provide extra operations and remember some information in the stack to be able to invert it. By its definition, $\epsilon \bullet \mathcal{E} = !\epsilon_1 \circ \epsilon_2$ because either $\epsilon = \epsilon_1$ and $\mathcal{E} = !\epsilon_2$, or $\epsilon = \epsilon_1 \circ \epsilon_2$ and $\mathcal{E} = \star$. To distinguish between the two possibilities, we remember in the stack π a boolean b defined by an operation $\mathcal{E} \overset{?}{=} \star$ which evaluates to $\top$

$$\langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l_1 \mid \Box\ (l_2[\mathcal{E}]) :: \mathbb{E}, \mathcal{E}, \mathsf{appl} :: \pi\rangle_{\mathsf{ev}} \quad \text{if } \mathsf{an}(l_1) = \emptyset \tag{D1}$$

$$\langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l_2 \mid (l_1[\mathcal{E}])\ \Box :: \mathbb{E}, \mathcal{E}, \mathsf{appr} :: \pi\rangle_{\mathsf{ev}} \quad \text{if } \mathsf{an}(l_2) = \emptyset \tag{D2}$$

$$\langle \lambda^{\emptyset}.l \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l \mid \lambda :: \mathbb{E}, \Uparrow\mathcal{E}, \mathsf{lam} :: \pi\rangle_{\mathsf{ev}} \quad \text{if } \mathsf{an}(l) = \emptyset \tag{D3}$$

$$\langle l[\epsilon]^{\emptyset} \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle l \mid \mathbb{E}, \epsilon \bullet \mathcal{E}, \mathsf{env}(\mathcal{E} \stackrel{?}{=} \star) :: \pi\rangle_{\mathsf{ev}} \quad \text{if } \mathsf{an}(l) = \emptyset \tag{D4}$$

$$\langle n^{\emptyset} \mid \mathbb{E}, !\epsilon, \pi\rangle_{\mathsf{ev}} \to \langle \epsilon \mid \mathbb{E}, n, \star, \mathsf{var} :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon) = \emptyset \tag{D5}$$

$$\langle \lambda^{\emptyset}.l \mid \Box\ k :: \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle |\mathbb{E}\langle l[k \mathbin{\cdot^{\emptyset}} \mathcal{E}]^{\emptyset}\rangle| \mid \blacksquare, \star, \blacktriangle\rangle_{\mathsf{ev}} \tag{D6}$$

$$\langle l \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \to \langle \pi \mid l^{+\mathsf{ev}}, \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \quad \text{otherwise} \tag{D7}$$

$$\langle \mathsf{appl} :: \pi \mid l_1, \Box\ (l_2[\mathcal{E}]) :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \tag{D8}$$

$$\langle \mathsf{appr} :: \pi \mid l_2, (l_1[\mathcal{E}])\ \Box :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l_1 \mathbin{@^{\emptyset}} l_2 \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \tag{D9}$$

$$\langle \mathsf{env}(b) :: \pi \mid l, \mathbb{E}, \mathcal{E}'\rangle_{\mathsf{bev}} \to \langle l[\epsilon]^{\emptyset} \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \quad \text{if } \bullet^{-1}(b, \mathcal{E}') = (\epsilon, \mathcal{E}) \tag{D10}$$

$$\langle \mathsf{lam} :: \pi \mid l, \lambda :: \mathbb{E}, \Uparrow\mathcal{E}\rangle_{\mathsf{bev}} \to \langle \lambda^{\emptyset}.l \mid \mathbb{E}, \mathcal{E}, \pi\rangle_{\mathsf{ev}} \tag{D11}$$

$$\langle \mathsf{var} :: \pi \mid \epsilon, \mathbb{E}, n, \star\rangle_{\mathsf{bvar}} \to \langle n^{\emptyset} \mid \mathbb{E}, !\epsilon, \pi\rangle_{\mathsf{ev}} \tag{D12}$$

$$\langle \blacktriangle \mid l, \blacksquare, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l[\mathcal{E}]\rangle_{\mathsf{nf}} \tag{D13}$$

$$\langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, \mathsf{consN}(l) :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon) = \emptyset \tag{D14}$$

$$\langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, 0, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle l \mid \mathbb{E}, \mathcal{E}, \mathsf{cons0}(\epsilon) :: \pi\rangle_{\mathsf{ev}} \quad \text{if } \mathsf{an}(l) = \emptyset \tag{D15}$$

$$\langle \mathtt{id}^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \star, \mathsf{id}! :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon) = \emptyset \tag{D16}$$

$$\langle \uparrow^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n+1, \star, \mathsf{shift} :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon) = \emptyset \tag{D17}$$

$$\langle \Uparrow^{\emptyset}\epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \uparrow^{\emptyset} \bullet \mathcal{E}, \mathsf{liftN}(\mathcal{E} \stackrel{?}{=} \star) :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon) = \emptyset \tag{D18}$$

$$\langle \Uparrow^{\emptyset}\epsilon \mid \mathbb{E}, 0, !\phi, \pi\rangle_{\mathsf{var}} \to \langle \phi \mid \mathbb{E}, 0, \star, \mathsf{lift0}(\epsilon) :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon_2) = \emptyset \tag{D19}$$

$$\langle \epsilon_1 \mathbin{\circ^{\emptyset}} \epsilon_2 \mid \mathbb{E}, n, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \epsilon_1 \mid \mathbb{E}, n, \epsilon_2 \bullet \mathcal{E}, \mathsf{comp}(\mathcal{E} \stackrel{?}{=} \star) :: \pi\rangle_{\mathsf{var}} \quad \text{if } \mathsf{an}(\epsilon_1) = \emptyset \tag{D20}$$

$$\langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, \pi\rangle_{\mathsf{var}} \to \langle \pi \mid \epsilon^{+\mathsf{var}}, \mathbb{E}, n, \mathcal{E}\rangle_{\mathsf{bvar}} \quad \text{otherwise} \tag{D21}$$

$$\langle \mathsf{consN}(l) :: \pi \mid \epsilon, \mathbb{E}, n, \mathcal{E}\rangle_{\mathsf{bvar}} \to \langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \tag{D22}$$

$$\langle \mathsf{cons0}(\epsilon) :: \pi \mid l, \mathbb{E}, \mathcal{E}\rangle_{\mathsf{bev}} \to \langle l \mathbin{\cdot^{\emptyset}} \epsilon \mid \mathbb{E}, 0, \mathcal{E}, \pi\rangle_{\mathsf{var}} \tag{D23}$$

$$\langle \mathsf{id}! :: \pi \mid \epsilon, \mathbb{E}, n, \star\rangle_{\mathsf{bvar}} \to \langle \mathtt{id}^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \tag{D24}$$

$$\langle \mathsf{shift} :: \pi \mid \epsilon, \mathbb{E}, n+1, \star, \rangle_{\mathsf{bvar}} \to \langle \uparrow^{\emptyset} \mid \mathbb{E}, n, !\epsilon, \pi\rangle_{\mathsf{var}} \tag{D25}$$

$$\langle \mathsf{liftN}(b) :: \pi \mid \epsilon, \mathbb{E}, n, \mathcal{E}'\rangle_{\mathsf{bvar}} \to \langle \Uparrow^{\emptyset}\epsilon \mid \mathbb{E}, n+1, \mathcal{E}, \pi\rangle_{\mathsf{var}} \quad \text{if } \bullet^{-1}(b, \mathcal{E}') = (\uparrow^{\emptyset}, \mathcal{E}) \tag{D26}$$

$$\langle \mathsf{lift0}(\epsilon_1) :: \pi \mid \epsilon_2, \mathbb{E}, 0, \star\rangle_{\mathsf{bvar}} \to \langle \Uparrow^{\emptyset}\epsilon_1 \mid \mathbb{E}, 0, !\epsilon_2, \pi\rangle_{\mathsf{var}} \tag{D27}$$

$$\langle \mathsf{comp}(b) :: \pi \mid \epsilon_1, \mathbb{E}, n, \mathcal{E}'\rangle_{\mathsf{bvar}} \to \langle \epsilon_1 \mathbin{\circ^{\emptyset}} \epsilon_2 \mid \mathbb{E}, n, \mathcal{E}, \pi\rangle_{\mathsf{var}} \quad \text{if } \bullet^{-1}(b, \mathcal{E}') = (\epsilon_2, \mathcal{E}) \tag{D28}$$

Figure 4 Derived Abstract Machine.

$$\langle l_1 \mathbin{@^\Sigma} l_2 \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l_1 \mid \square\ (l_2[\mathcal{E}]) :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \mathcal{E}, \top \not\Vdash l_1 \tag{O1}$$
$$\langle l_1 \mathbin{@^\Sigma} l_2 \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l_2 \mid (l_1[\mathcal{E}])\ \square :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \mathcal{E}, \bot \not\Vdash l_2 \tag{O2}$$
$$\langle \lambda^\Sigma.l \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l \mid \lambda :: \mathbb{E}, \Uparrow\mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \Uparrow\mathcal{E}, \bot \not\Vdash l \tag{O3}$$
$$\langle l[\epsilon]^\Sigma \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle l \mid \mathbb{E}, \epsilon \bullet \mathcal{E}\rangle_{\mathsf{ev}} \tag{O4}$$
$$\langle n \mid \mathbb{E}, !\epsilon\rangle_{\mathsf{ev}} \to \langle \epsilon \mid \mathbb{E}, n, \star, (n, \epsilon)\rangle_{\mathsf{var}} \tag{O5}$$
$$\langle \lambda^\Sigma.l \mid \square\ k :: \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \to \langle \mathbb{E} \mid l[k \cdot \mathcal{E}]^\emptyset\rangle_{\mathsf{rec}} \tag{O6}$$
$$\langle n \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \to \langle \mathbb{E} \mid n[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{bev}} \tag{O7}$$
$$\langle c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \to \langle \mathbb{E} \mid c^{+\mathsf{ev}}\rangle_{\mathsf{bev}} \quad \text{otherwise} \tag{O8}$$

$$\langle \square\ l :: \mathbb{E} \mid c\rangle_{\mathsf{bev}} \to \langle c \mathbin{@^\emptyset} l \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O9}$$
$$\langle l\ \square :: \mathbb{E} \mid c\rangle_{\mathsf{bev}} \to \langle l \mathbin{@^\emptyset} c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O10}$$
$$\langle \lambda :: \mathbb{E} \mid c\rangle_{\mathsf{bev}} \to \langle \lambda^\emptyset.c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O11}$$
$$\langle \blacksquare \mid c\rangle_{\mathsf{bev}} \to \langle c\rangle_{\mathsf{nf}} \tag{O12}$$

$$\langle l \cdot \epsilon \mid \mathbb{E}, n+1, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \mathcal{E}, P\rangle_{\mathsf{var}} \tag{O13}$$
$$\langle l \cdot \epsilon \mid \mathbb{E}, 0, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle l \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}} \quad \text{if } \mathcal{E}, \mathsf{infun}(\mathbb{E}) \not\Vdash l \tag{O14}$$
$$\langle \mathtt{id} \mid \mathbb{E}, n, !\epsilon, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \star, P\rangle_{\mathsf{var}} \tag{O15}$$
$$\langle \uparrow \mid \mathbb{E}, n, !\epsilon, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n+1, \star, P\rangle_{\mathsf{var}} \tag{O16}$$
$$\langle \Uparrow\epsilon \mid \mathbb{E}, n+1, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle \epsilon \mid \mathbb{E}, n, \uparrow \bullet \mathcal{E}, P\rangle_{\mathsf{var}} \tag{O17}$$
$$\langle \Uparrow\epsilon \mid \mathbb{E}, 0, !\phi, P\rangle_{\mathsf{var}} \to \langle \phi \mid \mathbb{E}, 0, \star, P\rangle_{\mathsf{var}} \tag{O18}$$
$$\langle \epsilon_1 \circ \epsilon_2 \mid \mathbb{E}, n, \mathcal{E}, P\rangle_{\mathsf{var}} \to \langle \epsilon_1 \mid \mathbb{E}, n, \epsilon_2 \bullet \mathcal{E}, P\rangle_{\mathsf{var}} \tag{O19}$$
$$\langle \phi \mid \mathbb{E}, m, \mathcal{E}, (n, \epsilon)\rangle_{\mathsf{var}} \to \langle \mathbb{E} \mid n[\epsilon]^{\mathsf{ev}}\rangle_{\mathsf{bev}} \quad \text{otherwise} \tag{O20}$$

$$\langle \square\ l :: \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle \mathbb{E} \mid c \mathbin{@^\emptyset} l\rangle_{\mathsf{rec}} \tag{O21}$$
$$\langle l\ \square :: \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle \mathbb{E} \mid l \mathbin{@^\emptyset} c\rangle_{\mathsf{rec}} \tag{O22}$$
$$\langle \lambda :: \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle \mathbb{E} \mid \lambda^\emptyset.c\rangle_{\mathsf{rec}} \tag{O23}$$
$$\langle \mathbb{E} \mid c\rangle_{\mathsf{rec}} \to \langle c \mid \mathbb{E}, \star\rangle_{\mathsf{ev}} \tag{O24}$$

Figure 5 Optimized Abstract Machine.

if $\mathcal{E} = \star$, and to $\bot$ otherwise (cf steps D4, D18, and D20). We then define the invert of $\bullet$, written $\bullet^{-1}$, such that $\bullet^{-1}(\top, !\epsilon_1 \circ \epsilon_2) = (\epsilon_1 \circ \epsilon_2, \star)$ and $\bullet^{-1}(\bot, !\epsilon_1 \circ \epsilon_2) = (\epsilon_1, !\epsilon_2)$, and use it in the backtracking steps D10, D26, and D28. Defining $\bullet^{-1}$ or $\overset{?}{=}$ cannot be done for any non injective operator, but if they exist, such operations could be provided to the automatic derivation procedure with the zipper semantics.

3.2 Optimized Abstract Machine

Being the result of a generic procedure, the DAM is not optimized for the particular case of the λ-calculus with environments. We explain how to optimize it, resulting in the Optimized Abstract Machine, defined in Figure 5.

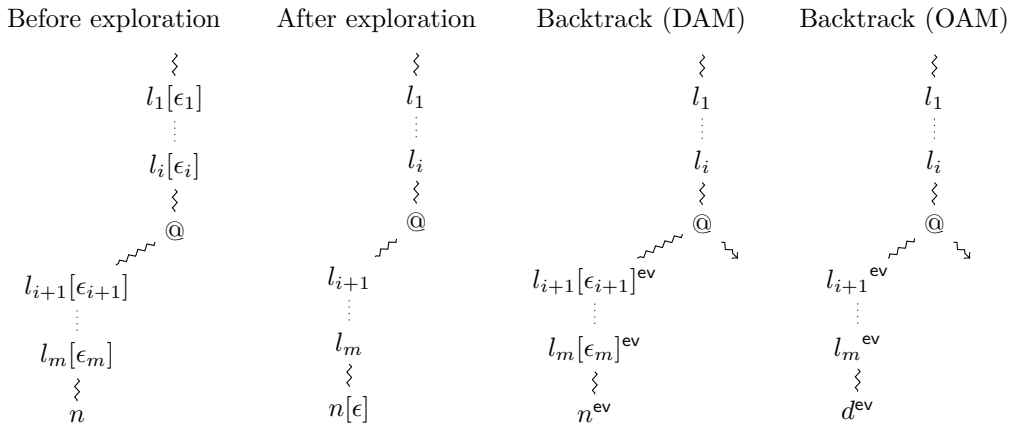

Figure 6 Differences between the DAM and OAM when backtracking.

Bypassing the bvar mode. The DAM does not exploit the fact that solving n in ϵ is syntax-directed: at most one zipper rule applies to a given n and ϵ. Suppose the machine configuration $\mathcal{D} = \langle n^{\Sigma} \mid \mathbb{E}, !\epsilon, \pi \rangle_{\mathsf{ev}}$ initiates a sequence of steps $\mathcal{D} \to^{m} \mathcal{D}'_{\mathsf{var}}$ such that no forward step applies to $\mathcal{D}'_{\mathsf{var}}$. The DAM then backtracks: starting with $\mathcal{D}'_{\mathsf{var}}$ and for each step, it applies the switching step to add the annotation, and then a backward step. In the end, we need $2m$ steps to add the annotations and get back to $\mathcal{D}$. Instead, the switching step from $\mathcal{D}^{m}_{\mathsf{var}}$ could restore $\mathcal{D}$ directly, meaning that we no longer need backtracking steps for the var mode, nor annotations for environments.

To implement this change, whenever we start solving n inside ϵ in the OAM, we remember the initial values of the variable and the environment as a pair (n, ϵ) in the var mode (step O5). We let P range over such pairs. The pair is passed along in steps O13 to O19, and is discarded if the variable can in fact be solved in step O14. If variable resolution fails, the "otherwise" step restores the initial values as a closure $n[\epsilon]^{\mathsf{ev}}$ annotated as a normal form (step O20).

Refocusing. The DAM is not in refocused form: after β-reduction, it plugs the resulting closure into $\mathbb{E}$ before looking for the next redex. This is necessary for the machine to be complete w.r.t. the zipper semantics, so that it is able to reach any available redex, including those in $\mathbb{E}$. A middle ground between performance and completeness is to reconstruct an arbitrary part of the context $\mathbb{E}$, while still having the possibility to recreate it completely.

In the OAM, β-reduction produces a closure in the context $\mathbb{E}$ (step O6). The OAM then enters the rec mode dedicated to partially reconstruct the full closure using $\mathbb{E}$. The steps O21 to O23 push the current top frame of the context on the closure. At any point, step O24 may conclude the reconstruction and restart the machine in the ev mode to find the next redex. The reconstruction being non-deterministic itself, the machine may reconstruct fully using the whole context, meaning that any redex in $\mathbb{E}$ is still reachable, and the machine is still complete w.r.t. the zipper semantics.

Removing the stack π. The stack π is used for backtracking, mainly to remember in which order to undo forward steps. To simplify the machine, we would like to use the context $\mathbb{E}$ for that role, as it uniquely changes when the rules appL, appR, and lam are applied. However, it stays the same when env or var are applied (cf. Figure 2). The idea is then to undo the former steps, but not the latter ones. Such a change entails significant differences in the behavior of the OAM compared to the DAM, that we illustrate on an example in Figure 6.

The first picture represents a possible exploration path in a closure, which goes through several constructs $l_1[\epsilon_1] \dots l_m[\epsilon_m]$. Along this path, the two machines collect $\epsilon_1 \dots \epsilon_m$ and combine them into an environment ϵ (second picture).

If solving n in ϵ produces a normal form d, the DAM restores the closure in its original shape while adding annotations, up to some application where it stops backtracking and starts exploring another path. Concretely, the DAM undoes the variable resolution steps, annotates n as a normal form, and puts back the environments $(\epsilon_i \dots \epsilon_m)$ at their original position. The OAM backtracks up to the same application, annotating all the constructs on the way, but without undoing the variable resolution steps – leaving d in place – or restoring $(\epsilon_i \dots \epsilon_m)$.

In practice, the OAM does not feature the stack π, and there are only three backtracking steps (O9 to O11) which mirror the steps O1 to O3. The steps O4 and O5 cannot be reversed. A construct $l[\epsilon]^\Sigma$ in the OAM is therefore either yet to be traversed, or is of the form $n[\epsilon]^{\mathsf{ev}}$ after a failed variable resolution. If we go through a plain term (without any $l[\epsilon]^\Sigma$), we might reach a variable n with the local environment $\star$. To be uniform, we create an annotated closure $n[\mathtt{id}]^{\mathsf{ev}}$ (step O7), turning the plain term itself into a closure. Unlike in the DAM, plain terms cannot be annotated in the OAM: only closures are annotated.

Keeping annotations. Instead of erasing all the annotations after β-reduction as in the DAM, we would like to keep as much of them as possible. Suppose the β-redex is $(\lambda^\emptyset.l) \mathbin{@^\emptyset} k$ in some context $\mathbb{E}$ and current environment $\mathcal{E}$. The normal forms in $\mathbb{E}$ are still normal form after the β-reduction, so if any of them are annotated as such, we should keep these annotations. In contrast, the annotations in the body of the λ-abstraction l may no longer be valid: a term $0^{\mathsf{ev}} \mathbin{@^{\mathsf{ev}}} t$ is no longer a normal form if the β-reduction replaces 0 with a λ-abstraction. As an approximation, we ignore all the annotations inside l.

We therefore need a way to know if an annotation is in the body of a β-redex. We notice that if solving a variable in a local environment $\mathcal{E}$ produces a normal form, then the machine adds an annotation and resets $\mathcal{E}$ in the configuration to $\star$ (steps O20 or O7). We then propagate the annotation upward from the leaves with the combinations of step O8 and steps O9 to O11. In that process, the local environment remains $\star$. We have $\mathcal{E} \neq \star$ only when we explore a closure for which we do not know yet if it is a normal form. This observation is enough to know if an annotation is meaningful after a β-reduction. Let c such that $\mathsf{an}(c) = \mathsf{ev}$ in some environment $\mathcal{E}$; the annotation is valid only if $\mathcal{E} = \star$. Otherwise, the environment has changed after c has been annotated and c is in the body of a β-redex.

On top of that, we need to consider a particular case. Suppose we look for a redex in $(\lambda^\emptyset.0 \mathbin{@^\emptyset} l) \mathbin{@^\emptyset} \lambda^\emptyset.k$, starting with the λ-abstraction on the right, annotating it as $(\lambda^\emptyset.0 \mathbin{@^\emptyset} l) \mathbin{@^\emptyset} \lambda^{\mathsf{ev}}.k$. After β-reduction and solving 0, we get $(\lambda^{\mathsf{ev}}.k) \mathbin{@^\emptyset} l[\epsilon]^\emptyset$ in the environment $\star$ (with $\epsilon = \lambda^{\mathsf{ev}}.k \cdot \mathtt{id}$); according to the previous criterion, the annotation of the λ-abstraction is valid, preventing the machine to β-reduce further. For a λ-abstraction, to be a normal form means it is not in the function position of an application. If a λ-abstraction ends up in such a position after a β-reduction, we need to ignore its annotation.

Consider a local environment $\mathcal{E}$ and a term or closure l. We use booleans $\top, \bot$, ranged over by b, to indicate whether l is in the function position of an application. Given a context $\mathbb{E}$, the function $\mathsf{infun}(\mathbb{E})$ returns $\top$ if the hole of the context is in function position ($\mathbb{E} = \Box\, l :: \mathbb{E}'$), and $\bot$ otherwise. We write $\mathcal{E}, b \vdash l$ if the annotation of l is still valid, i.e., with $\mathcal{E} = \star$, and l is a closure c such that $\mathsf{an}(c) = \mathsf{ev}$, and in the case where l is a λ-abstraction, with $b = \bot$. Otherwise, we write $\mathcal{E}, b \nvdash l$. We use this predicate as a side condition of the steps O1 to O3 and step O14, meaning that we can take these steps if the new subterm under focus is not annotated, or its annotation is meaningless. There is no side condition for step O4, because for this step, the local environment $\epsilon \bullet \mathcal{E}$ cannot be $\star$ by definition of $\bullet$.

Example. We show a machine run starting from $t = (\lambda^{\emptyset}.0\,@^{\emptyset}\,\lambda^{\emptyset}.1)\,@^{\emptyset}(\lambda^{\emptyset}.0)\,@^{\emptyset}\,1$, i.e., $(\lambda x.x\ \lambda y.x)\ (\lambda x.x)\ z$. To simplify, we omit empty annotations and the symbol @ if it is not annotated, writing for instance t as $(\lambda.0\ \lambda.1)\ (\lambda.0)\ 1$

The machine may go right of the topmost application, focusing on an unsolvable variable, since $\mathcal{E}$ is $\star$ for now. The machine annotates it and backtracks.

$$\langle t \mid \blacksquare, \star\rangle_{\mathsf{ev}} \overset{\mathrm{O2}}{\to} \langle 1 \mid (\lambda.0\ \lambda.1)\ (\lambda.0)\ \square :: \blacksquare, \star\rangle_{\mathsf{ev}} \overset{\mathrm{O7}}{\to} \langle (\lambda.0\ \lambda.1)\ (\lambda.0)\ \square :: \blacksquare \mid 1[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{bev}}$$
$$\overset{\mathrm{O10}}{\to} \langle (\lambda.0\ \lambda.1)\ (\lambda.0)\ 1[\mathtt{id}]^{\mathsf{ev}} \mid \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_1$$

The machine has no choice but to go left of the application. Then it might go left again, and inside the body of the λ-abstraction.

$$\mathcal{O}_1 \overset{\mathrm{O1O1O3}}{\to\to\to} \langle 0\ \lambda.1 \mid \lambda :: \square\ \lambda.0 :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_2$$

The term $0\ \lambda.1$ is a normal form, so the machine annotates it and backtracks.

$$\mathcal{O}_2 \to^* \overset{\mathrm{O8}}{\to} \langle \lambda :: \square\ \lambda.0 :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare \mid 0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{bev}}$$
$$\overset{\mathrm{O11}}{\to} \langle \lambda.0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}} \mid \square\ \lambda.0 :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_3$$

At this point, the machine β-reduces and switches to the rec mode, in which it may decide to reconstruct the frame in the context. It then restarts in ev mode to find the next redex.

$$\mathcal{O}_3 \overset{\mathrm{O6}}{\to} \langle \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare \mid (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}]\rangle_{\mathsf{rec}}$$
$$\overset{\mathrm{O22}}{\to} \langle \blacksquare \mid (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}]\ 1[\mathtt{id}]^{\mathsf{ev}}\rangle_{\mathsf{rec}}$$
$$\overset{\mathrm{O24}}{\to} \langle (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}]\ 1[\mathtt{id}]^{\mathsf{ev}} \mid \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_4$$

The machine cannot go right of the application, because we have $\star, \bot \vdash 1[\mathtt{id}]^{\mathsf{ev}}$: the annotation is still valid. The machine therefore has to go left.

$$\mathcal{O}_4 \overset{\mathrm{O1}}{\to} \langle (0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}})[\lambda.0 \cdot \mathtt{id}] \mid \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}}$$
$$\overset{\mathrm{O4}}{\to} \langle 0[\mathtt{id}]^{\mathsf{ev}}\ @^{\mathsf{ev}}\ \lambda^{\mathsf{ev}}.1[\mathtt{id}]^{\mathsf{ev}} \mid \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\lambda.0 \cdot \mathtt{id}\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_5$$

Because the local environment is no longer $\star$, the annotations of the subterm under focus are not valid and can be ignored. For instance, we can go right of the application and under the λ-abstraction to reach the variable.

$$\mathcal{O}_5 \overset{\mathrm{O2O3O4}}{\to\to\to} \langle 1 \mid \lambda :: 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \square :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\mathtt{id} \circ \Uparrow(\lambda.0 \cdot \mathtt{id})\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_6$$

Solving the variable in the var mode produces $\lambda.0$, a normal form, annotated as such by the machine, which then goes back to the first application in the context.

$$\mathcal{O}_6 \to^* \langle \lambda.0 \mid \lambda :: 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \square :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\uparrow\rangle_{\mathsf{ev}}$$
$$\to^* \langle \lambda.0[\Uparrow\uparrow]^{\mathsf{ev}} \mid \lambda :: 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \square :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}}$$
$$\to^* \langle 0[\mathtt{id}]^{\mathsf{ev}}[\lambda.0 \cdot \mathtt{id}]\ \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} \mid \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_7$$

The machine has no choice but to go left of the application, until it reaches the variable, which can be solved.

$$\mathcal{O}_7 \overset{\mathrm{O1O4O4}}{\to\to\to} \langle 0 \mid \square\ \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, !\mathtt{id} \circ (\lambda.0 \cdot \mathtt{id})\rangle_{\mathsf{ev}}$$
$$\overset{\mathrm{O5O19O15O14}}{\to\to\to\to} \langle \lambda.0 \mid \square\ \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} :: \square\ 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_8$$

The machine may then β-reduce and not reconstruct anything from the context.

$$\mathcal{O}_8 \overset{\text{O6O24}}{\to\to} \langle 0[\lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} \cdot \mathtt{id}] \mid \Box\, 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \triangleq \mathcal{O}_9$$

The variable can be solved immediately, producing a λ-abstraction. A β-reduction is then possible, without any reconstruction because the context is empty.

$$\mathcal{O}_9 \to^* \langle \lambda^{\mathsf{ev}}.\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}} \mid \Box\, 1[\mathtt{id}]^{\mathsf{ev}} :: \blacksquare, \star\rangle_{\mathsf{ev}} \to^* \langle (\lambda^{\mathsf{ev}}.0[\Uparrow\uparrow]^{\mathsf{ev}})[1[\mathtt{id}]^{\mathsf{ev}} \cdot \mathtt{id}] \mid \blacksquare, \star\rangle_{\mathsf{ev}}$$

The machine has to go through the resulting closure to annotate it as a normal form, pushing the environment $1[\mathtt{id}]^{\mathsf{ev}} \cdot \mathtt{id}$ in the body of the λ-abstraction while doing so. The run ends with $\langle \lambda^{\mathsf{ev}}.0[\Uparrow(1[\mathtt{id}]^{\mathsf{ev}} \cdot \mathtt{id}) \circ \Uparrow\uparrow]^{\mathsf{ev}} \rangle_{\mathsf{nf}}$.

Implementation. We have implemented the OAM as a skeletal semantics [20] from which we extract an OCaml program that evaluates all possible reductions from a starting term. The implementation is very close to the formal definition of the OAM and is available online [13].

3.3 Correspondence between the DAM and the OAM

To relate the OAM and the DAM, we define an equivalence $\approx$ between OAM and DAM machine states, and prove that from a sequence of OAM steps, we can build a sequence of equivalent DAM steps and conversely. We let $\mathcal{O}$ and $\mathcal{D}$ range over the machines states of the optimized and the derived machines, respectively. We informally explain the ingredients needed to define $\approx$; the formal definitions and proofs are in the report [15, Appendix B].

Consider two similar runs of the OAM and DAM from the same starting point: after doing the same β-reductions, they are at the same position in the closure. As explained in Figure 6 some environments may be pushed to the leaves and some variables may be solved in the OAM, but not in the DAM. Besides the closures on the OAM side may contain more annotations than on the DAM one, but some of these annotations could be meaningless.

We relate two states $\mathcal{O}$ and $\mathcal{D}$ by pushing all the environments to the leaves, and then solving all the variables, which is exactly what the translation $[\![\cdot]\!]$ of Figure 3 is doing. We define two similar translations on annotated closures or terms $[\![\cdot]\!]_{\mathsf{o}}$ (for the OAM) and $[\![\cdot]\!]_{\mathsf{d}}$ (for the DAM) which produce annotated terms. The translation $[\![\cdot]\!]_{\mathsf{o}}$ also removes meaningless annotations, by using the same criterion as in Section 3.2. We then compare the results r_{o} and r_{d} of these translations using a predicate $\triangleright$, which checks that the two annotated terms are the same, except that r_{o} may contain more annotations than r_{d}.

We extend $[\![\cdot]\!]_{\mathsf{o}}$, $[\![\cdot]\!]_{\mathsf{d}}$ and $\triangleright$ to contexts frame by frame, and we write $\mathcal{O} \approx \mathcal{D}$ if either:

- $\mathcal{O} = \langle l \mid \mathbb{E}, \mathcal{E}\rangle_{\mathsf{ev}}$, $\mathcal{D} = \langle k \mid \mathbb{F}, \mathcal{F}, \pi\rangle_{\mathsf{ev}}$, $[\![l]\!]_{\mathsf{o}}(\mathcal{E}) \triangleright [\![k]\!]_{\mathsf{d}}(\mathcal{F})$, and $[\![\mathbb{E}]\!]_{\mathsf{o}} \triangleright [\![\mathbb{F}]\!]_{\mathsf{d}}$;
- $\mathcal{O} = \langle \mathbb{E} \mid c\rangle_{\mathsf{bev}}$, $\mathcal{D} = \langle \pi \mid k, \mathbb{F}, \mathcal{F}\rangle_{\mathsf{bev}}$, $[\![c]\!]_{\mathsf{o}}(\star) \triangleright [\![k]\!]_{\mathsf{d}}(\mathcal{F})$, and $[\![\mathbb{E}]\!]_{\mathsf{o}} \triangleright [\![\mathbb{F}]\!]_{\mathsf{d}}$;
- $\mathcal{O} = \langle c\rangle_{\mathsf{nf}}$, $\mathcal{D} = \langle d\rangle_{\mathsf{nf}}$, and $[\![c]\!]_{\mathsf{o}}(\star) \triangleright [\![d]\!]_{\mathsf{d}}(\star)$.

The equivalence is defined only on the modes which are common to both machines. Equivalent machine states can be considered as synchronization points between the OAM and DAM machine runs. In particular, the contexts $\mathbb{E}$ and $\mathbb{F}$ of two equivalent states have the same shape, so the machines are at the same position in the term or closure being explored.

We extend the equivalence to sequences of machines states, writing $(\mathcal{O}_0, \mathcal{O}_1 \dots \mathcal{O}_i) \approx (\mathcal{D}_0, \mathcal{D}_1 \dots \mathcal{D}_i)$ if $\mathcal{O}_0 = \langle l, \blacksquare, \star\rangle_{\mathsf{ev}}$, $\mathcal{D}_0 = \langle l, \blacksquare, \star, \blacktriangle\rangle_{\mathsf{ev}}$ for the same l, $\mathcal{O}_j \to^* \mathcal{O}_{j+1}$ and $\mathcal{D}_j \to^* \mathcal{D}_{j+1}$ for all $0 \leq j < i$, and $\mathcal{O}_j \approx \mathcal{D}_j$ for all $0 \leq j \leq i$. The correspondence theorems then state that given two equivalent sequences, if one of the machines is doing a step, then we can find equivalent states to continue the sequences.

▶ **Theorem 3** (OAM implies DAM). *If* $(\mathcal{O}_0, \mathcal{O}_1 \dots \mathcal{O}_i) \approx (\mathcal{D}_0, \mathcal{D}_1 \dots \mathcal{D}_i)$ *and* $\mathcal{O}_i \to \mathcal{O}'$*, there exist* $\mathcal{O}_{i+1}$ *and* $\mathcal{D}_{i+1}$ *such that* $\mathcal{O}' \to^* \mathcal{O}_{i+1}$*,* $\mathcal{D}_i \to^* \mathcal{D}_{i+1}$*, and* $\mathcal{O}_{i+1} \approx \mathcal{D}_{i+1}$*.*

▶ **Theorem 4** (DAM implies OAM). *If* $(\mathcal{O}_0, \mathcal{O}_1 \dots \mathcal{O}_i) \approx (\mathcal{D}_0, \mathcal{D}_1 \dots \mathcal{D}_i)$ *and* $\mathcal{D}_i \to \mathcal{D}'$*, there exist* $\mathcal{O}_{i+1}$ *and* $\mathcal{D}_{i+1}$ *such that* $\mathcal{D}' \to^* \mathcal{D}_{i+1}$*,* $\mathcal{O}_i \to^* \mathcal{O}_{i+1}$*, and* $\mathcal{O}_{i+1} \approx \mathcal{D}_{i+1}$*.*

We sketch some of the cases of Theorem 3, the proofs for Theorem 4 are similar. Step O20 of the OAM (Figure 5) is matched in the DAM by a sequence of bvar steps which produces an equivalent configuration. Applying the axiom (step O6) leads to the rec mode, in which a number m of frames of $\mathbb{E}$ are reconstructed. If $\mathbb{E}$ contains n frames in total, the DAM matches by applying its β-reduction step (step D6 in Figure 4), which rebuilds the whole context, and then it gets to the same position as in the OAM with $n - m$ forward steps.

A backward step of the OAM, e.g. step O9, is not necessarily matched by the corresponding step in the DAM – step D8 in Figure 4 – because the DAM may have to undo steps corresponding to the applications of the rules env or var. We prove that we can undo them first and then apply step D8 in the DAM. Finally, suppose a valid annotation in the OAM has been erased at the same position in the DAM. The subterm under focus is still a normal form, so the sequence which added the annotation on the DAM side before its erasure can still be applied to that subterm to restore the annotation.

4 Recovering Known Strategies and Abstract Machines

The OAM can be used to obtain specialized abstract machines that implement various reduction strategies. A reduction strategy can be seen as a way to choose a particular redex. In the OAM, the transitions O1, O2, O3 are navigating through the term or closure by going left ($\swarrow$) or right ($\searrow$) of the application, or down under the lambda ($\downarrow$), respectively. By controlling which of these navigation rules to apply and in which order, we can define various deterministic or non-deterministic strategies. To define reduction strategies succinctly, we use the phased format of strategies [18]. For example, the call-by-name strategy can be defined as $cbn = \swarrow cbn; \beta$, a strategy that navigates to the left of an application as much as possible (iterating itself), and then tries to perform β-reduction. Formally, a strategy can be defined as a set of term decompositions. The sequence of strategies $r; s$ is a strategy containing any decomposition in r, or a r-normal form that is in s. Furthermore, $r \cup s$ is a strategy that contains any decomposition that is in r or in s. If a term admits multiple decompositions, according to r or to s, the strategy $r \cup s$ is non-deterministic.

Krivine Abstract Machine. The Krivine Abstract Machine [38] implements exactly call-by-name evaluation as defined above. In the OAM, it corresponds to restricting navigation to only $\swarrow$, and removing $\searrow$ and $\downarrow$. Thus, we go left as much as possible, and if we reach a λ-abstraction, we try to apply step O6 to β-reduce. If it fails because the context is empty, we are done, and we apply steps O8 and O12: the (annotated) lambda abstraction will be returned as the normal form. If, on the other hand, we reach a variable, the machine tries to solve it by rule O5. If it is a free variable, the machine cannot solve it and backtracks, thus reconstructing the call-by-name normal form – an application of a (annotated) variable to a number of closures (possibly none). An inspection of reachable machine configurations reveals that excluding the two navigation rules further entails that rules O10, O11, and also O16-O18, O22, O23 are not reachable: environments are constructed using only two operators, and thus the machine is significantly simplified. Moreover, the call-by-name strategy being deterministic, the reconstruction transitions can be short-circuited and eliminated altogether.

Right-to-left open call by value. We consider a variant of the call-by-value strategy that operates on open terms, and evaluates arguments in an application from right to left: it is defined as $rcbw = \searrow rcbw; \swarrow rcbw; \beta$. This strategy can be realized in the OAM by keeping $\swarrow$ and $\searrow$ and forbidding $\downarrow$, but with the priority given to $\searrow$, applying it whenever possible before we try $\swarrow$. Whenever we can apply the β-rule, the argument on top of the stack must be a value, because of the precedence of $\searrow$ over $\swarrow$: the machine only performs a restricted β-contraction, as in call-by-value. The resulting machine is closest to the variant presented in [19], or it can be seen as a right-to-left variant of the CEK machine [32].

In contrast, we cannot obtain the left-to-right variant of call-by-value in a similar way. To do it, we would need to prioritize $\swarrow$ over $\searrow$, which in turn requires further inspection of the top of the stack to make sure that β-contraction is only applied when the argument is a value. Other similar call-by-value strategies [43, 7] cannot be expressed. It is due to the fact that OAM only looks at the top-level constructor and no further.

Normal order. The normal-order strategy is a fully reducing strategy that can be seen as an iteration of call-by-name until no redex is left anywhere in the term. Its phased form is $no = (\beta; \swarrow no; \searrow no) \cup \downarrow no$. It is worth noting that in this case the sum of strategies $\cup$ does not produce a non-deterministic strategy, because $(\beta; \swarrow no; \searrow no)$ and $\downarrow no$ each contains decompositions of structurally different terms. Normal order is an example of a hybrid strategy that does not behave uniformly, but the navigation steps depend on the context: the $\downarrow$ rule can only be applied when the lambda abstraction is not in the operator position. A canonical abstract machine that implements normal order is Crégut's KN machine [23]. In the OAM, we first try to apply β, then $\swarrow$, then $\searrow$. Otherwise, we can try $\downarrow$ (only when we know that β is not applicable). This adjustment of OAM results in a variant of KN.

Weak reduction. A simple restriction that results in a non-deterministic strategy can be obtained by removing the rule $\downarrow$ from the OAM. This strategy can be described in the phased form as $weak = \swarrow weak \cup \searrow weak \cup \beta$.

Head reduction. Consider the following two strategies in phased form: $head = (\beta; \swarrow head) \cup \downarrow head$ and $ihead = (\swarrow ihead; \beta) \cup \downarrow ihead$. They both perform head reduction and compute head normal forms (i.e., terms of the form $\lambda x_1 \ldots \lambda x_n . y\ t_1 \ldots t_n$), but the first strategy is "outermost", and the second is "innermost". The two strategies are deterministic, because for each term only one of the $\cup$-substrategies can be used. If we restrict OAM in that we remove the rule $\searrow$ and order the remaining evaluation rules: first $\swarrow$, $\downarrow$, and then β, we obtain the implementation of the *ihead* strategy. If, on the other hand, we put these rules in the order: β first, then $\swarrow$, then $\downarrow$, we obtain the machine for the *head* strategy. In both cases, we can further trim the machine by removing all the transitions made unreachable by removing $\searrow$.

Unexpressible strategies. We have remarked above that the left-to-right call by value is not expressible in an instance of OAM obtained by restriction and prioritization only. The same is true for any strategy that hinges on the same issue, like the *fireball calculus* [43, 7], the non-deterministic variant of CbV where we can reduce freely to the left or to the right of the application as long as it is possible, and only then we try to β-reduce. This strategy in phased form is defined as $cbw = (\swarrow cbw \cup \searrow cbw); \beta$, but in fact this definition enforces that β-contraction will be applied only when the argument is a weak normal form. So it is not possible to directly obtain this kind of behavior in the OAM without further inspection of the stack in O6, as in the left-to-right call by value.

5 Related Work

Non-deterministic abstract machines for the λ-calculus. In [14], Biernacka et al. present a general framework for deriving non-deterministic abstract machines from zipper semantics. One of the example derivations in their work leads to a non-deterministic abstract machine for the λ-calculus, where no reduction strategy is imposed and each redex in a given term can be selected for contraction thanks to a loop-free backtracking mechanism encoded in the transition system of the machine. The abstract machines of the present work follow the same principles, but they are optimized crucially in two ways: they replace substitution with environments and they are in the refocused form. Both these advancements greatly improve the performance measures of the machines since they influence, respectively, how substitution is implemented [6] and how term decomposition is resumed after a redex has been contracted [27, 28].

A lightweight form of non-determinism is present in [3], a recent work by Accattoli and Barenbaum who introduce an abstract machine for normal-order reduction in the λ-calculus that has a diamond property. Their machine is non-deterministic in that when dealing with a partially normalized term of the form $x\,t_1\,\ldots\,t_n$, it can choose arbitrarily which argument t_i to reduce next. (The machine can also jump from one job to another when a normal form has been reached, without a step-by-step backtracking.) The machine is diamond non-deterministic, i.e., it satisfies the strongest, one-step form of confluence, which implies that the choice of a reduction site does not affect the result, nor the length of the machine run. The machines considered in the present work, as more general, obviously are not diamond non-deterministic in this sense.

Explicit substitutions and environment-based abstract machines. In their pioneering work on explicit substitutions, Abadi et al. [1] present an abstract machine for weak-head normalization in the $\lambda\sigma$-calculus that generalizes the KAM (originally featuring de Bruijn indices and closures) to open terms. Full normalization is then achieved as an iteration of the weakly normalizing machine. Substitutions from the calculus play the role of environments in the machine, which makes the relationship between the calculus and the machine particularly explicit. The $\lambda\sigma$-calculus is also used by Crégut [23] for designing and establishing the correctness of his strongly normalizing abstract machines KN and KNL. Call-by-value and call-by-name reduction strategies and their implementation on an environment-based abstract machine are discussed in the context of a variant of $\lambda\sigma$ with names and garbage collection by Rose [45].

In [24], Curien introduces a minimal weak non-deterministic calculus of closures $\lambda\rho$, requiring a big-step reduction rule for term application, that allows him to express the reduction strategies for the KAM and the Categorical Abstract Machine (CAM) [22], taking advantage of the presence of closures in the calculus. The author also sketches a generalization of the weak calculus to a strong version that accounts for full β-reduction, and is formalized in [25] as $\lambda\sigma_{\Uparrow}$.

The full power of the $\lambda\sigma_{\Uparrow}$-calculus is then used in Hardin et al.'s framework for reasoning about the correspondence between several weak reduction strategies in $\lambda\sigma_w$, a subcalculus of $\lambda\sigma_{\Uparrow}$, and known abstract machines such as the KAM, SECD, CAM, and Cardelli's FAM [21]. Curien's weak calculus of closures is, in turn, studied by Biernacka and Danvy [19] who introduce its modification $\lambda\hat{\rho}$ that, unlike $\lambda\rho$, makes it possible to define one-step reduction strategies underlying abstract machines such as the KAM, CEK, and Leroy's ZINC [41]. It is shown that $\lambda\hat{\rho}$ and the abstract machines are in an intimate relation established by an enriched refocusing. The calculus $\lambda\hat{\rho}$ is then used by García-Pérez et al. [35, 34] to reconstruct

and prove Cregut's KN correct in a very precise sense. In our case, the underlying small-step semantics is zipper zemantics, rather than the more traditional structural operational semantics.

Accattoli et al. [4] study the relationship between the linear substitution calculus (LSC) [11] and a number of abstract machines for weak normalization, including the KAM, CEK, a simplified version of SECD, the lazy KAM [23], and Sestoft's call-by-need machine [48] (ZINC is also sketched). It is argued that a weak version of LSC, a calculus with an explicit representation of reduction contexts and a notion of (explicit) substitution at a distance, can abstractly and effectively represent or distill, in the authors' words, abstract machines for weak normalization. Interestingly, the authors also introduce new machines inspired by LSC: the Milner Abstract Machine (MAM) that uses only one, global environment and therefore avoids the concept of closure at a cost of explicit α-renaming, and the Milner Abstract machine by-neeD (MAD) that takes advantage of the single global environment introduced in the MAM that facilitates sharing of computed values.

In [5], the above ideas are used to design the Strong Milner Abstract Machine, a machine for normal-order strong normalization that features a single environment and is shown to have a linear overhead both in the number of steps in a reduction sequence in LSC and in the size of the initial term – Strong MAM is distilled (decoded) into LSC and analysed for complexity. Such issues are central in Accattoli's Useful MAM [2], that is the first reasonable, i.e., polynomially related to the cost model of RAM or Turing machines, machine for strong normalization in the λ-calculus in the literature. Useful MAM, however, is not studied in the context of LSC or another calculus of explicit substitutions. In the same line of work, Accattoli and Barras provide a thorough complexity analysis and comparison between machines with local and global environments [6], whereas Accattoli et al. [7, 10], present reasonable abstract machines for open call-by-value that rely on a global environment.

A special form of environments, called crumbled environments, is introduced in [9], where Accattoli et al. study call-by-value evaluation of both closed and open λ-terms. Such environments stem from the representation of terms considered by the authors, i.e., a representation with explicit sharing or explicit substitutions (in a new formulation), and they encode evaluation contexts besides storing delayed substitutions. The crumbling technique is then used by Accattoli et al. to define a reasonable abstract machine, called SCAM, for strong call-by-value [8]. Their work takes advantage of yet another calculus of explicit substitutions – the value substitution calculus (VSC) and of the concept of implosive sharing known from the literature on call-by-need [49]. Other reasonable abstract machines for strong call-by-value and strong call-by-need have been derived through the functional correspondence [12] from higher-order evaluators with heap-based sharing and local environments by Biernacka et al. [16, 17].

In contrast to the above collection of results by Accattoli et al. and Biernacka et al., in the present work we do not consider any sharing-oriented optimizations that go beyond the introduction of local environments, and we focus solely on the interaction of such environments with non-determinism. Whether such optimizations can be incorporated in NDAMs is a worthwhile question that requires further research.

6 Conclusion

We present an environment-based abstract machine for the λ-calculus without a fixed reduction strategy. To this end, we define a zipper semantics – inspired by a calculus of explicit substitutions [25] – and derive an NDAM from it. We then optimize the NDAM so that it requires fewer steps to find a redex. In the resulting OAM, we can recognize existing deterministic machines by restricting and/or prioritizing steps.

Our work demonstrates that zipper semantics can be defined with environments instead of substitution. It also shows that the derived machine can be used as a stepping stone towards more optimized machines. Some optimizations are language specific, like knowing which annotations to keep after reduction, but others could be made generic in the NDAM derivation procedure. For example, it is always possible to bypass the backtracking steps of a deterministic mode of the machine, like we do with the bvar mode in the OAM. Refocusing is also possible as long as the stack π can be replaced by another element for backtracking, such as the context $\mathbb{E}$. In fact, the non-deterministic machines for the λ-calculus with substitution and for HOcore [14] could be automatically derived in the refocused form from their respective zipper semantics.

We believe that our approach of defining a zipper semantics, optimizing the derived machine, and possibly restricting it to make it deterministic is generic and can be used to define non-deterministic and deterministic abstract machines for other calculi. In particular, we would like to apply this technique to the λ-calculus with futures [33, 42], and see whether we can recover the different task creation strategies of this calculus, such as the eager strategy [37] or the lazy one [30].

References

1 Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

2 Beniamino Accattoli. The Useful MAM, a reasonable implementation of the strong λ-calculus. In Jouko A. Väänänen, Åsa Hirvonen, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings*, volume 9803 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2016. doi:10.1007/978-3-662-52921-8_1.

3 Beniamino Accattoli and Pablo Barenbaum. A diamond machine for strong evaluation. *CoRR*, abs/2309.12515, 2023. doi:10.48550/arXiv.2309.12515.

4 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

5 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*, volume 9458 of *Lecture Notes in Computer Science*, pages 231–250. Springer, 2015. doi:10.1007/978-3-319-26529-2_13.

6 Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In Wim Vanhoof and Brigitte Pientka, editors, *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, Namur, Belgium, October 09 - 11, 2017*, pages 4–16. ACM, 2017. doi:10.1145/3131851.3131855.

7 Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.23.

8 Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470630.

9 Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019. doi:10.1145/3354166.3354169.

10 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019. doi:10.1016/j.scico.2019.03.002.

11 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

12 Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard. A functional correspondence between evaluators and abstract machines. In *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden*, pages 8–19. ACM, 2003.

13 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Optimized abstract machine implementation. Software, swhId: swh:1:dir:ba776d5226cde3b68cc6346fcd3a3495ad4d04e8 (visited on 06/06/2024). URL: https://gitlab.inria.fr/skeletons/lambda-am.git.

14 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Non-deterministic abstract machines. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPIcs*, pages 7:1–7:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

15 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Optimizing a non-deterministic abstract machine with environments. Available at https://inria.hal.science/hal-04568253, 2024.

16 Małgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A derived reasonable abstract machine for strong call by value. In Niccolò Veltri, Nick Benton, and Silvia Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 6:1–6:14. ACM, 2021. doi:10.1145/3479394.3479401.

17 Małgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A simple and efficient implementation of strong call by need by an abstract machine. *Proc. ACM Program. Lang.*, 6(ICFP):109–136, 2022. doi:10.1145/3549822.

18 Małgorzata Biernacka, Witold Charatonik, and Tomasz Drab. The zoo of lambda-calculus reduction strategies, and Coq. In June Andronick and Leonardo de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITP.2022.7.

19 Małgorzata Biernacka and Olivier Danvy. A concrete framework for environment machines. *ACM Trans. Comput. Log.*, 9(1):6, 2007.

20 Martin Bodin, Philippa Gardner, Thomas Jensen, and Alan Schmitt. Skeletal semantics and their interpretations. *PACMPL*, 3(POPL):44:1–44:31, 2019.

21 Luca Cardelli. Compiling a functional language. In Robert S. Boyer, Edward S. Schneider, and Guy L. Steele Jr., editors, *Proceedings of the 1984 ACM Conference on LISP and Functional Programming, LFP 1984, Austin, Texas, USA, August 5-8, 1984*, pages 208–217. ACM, 1984. doi:10.1145/800055.802037.

22 Guy Cousineau, Pierre-Louis Curien, and Michel Mauny. The Categorical Abstract Machine. *Sci. Comput. Program.*, 8(2):173–202, 1987. doi:10.1016/0167-6423(87)90020-7.

23 Pierre Crégut. Strongly reducing variants of the Krivine abstract machine. *High. Order Symb. Comput.*, 20(3):209–230, 2007. doi:10.1007/s10990-007-9015-z.

24 Pierre-Louis Curien. An abstract framework for environment machines. *Theor. Comput. Sci.*, 82(2):389–402, 1991. doi:10.1016/0304-3975(91)90230-Y.

25 Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *J. ACM*, 43(2):362–397, 1996. doi:10.1145/226643.226675.

26 Olivier Danvy. From reduction-based to reduction-free normalization. In Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra, editors, *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, volume 5832 of *Lecture Notes in Computer Science*, pages 66–164. Springer, 2008.

27 Olivier Danvy and Lasse R. Nielsen. Syntactic theories in practice. *Electron. Notes Theor. Comput. Sci.*, 59(4):358–374, 2001.

28 Olivier Danvy and Lasse R. Nielsen. Refocusing in reduction semantics. Research Report BRICS RS-04-26, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, November 2004. A preliminary version appeared in the informal proceedings of the Second International Workshop on Rule-Based Programming (RULE 2001), Electronic Notes in Theoretical Computer Science, Vol. 59.4.

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

30 Marc Feeley. *An efficient and general implementation of futures on large scale shared-memory multiprocessors*. PhD thesis, Department of Computer Science, Brandeis University, 1993.

31 Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 2009.

32 Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986*, pages 193–222. North-Holland, 1987.

33 Cormac Flanagan and Matthias Felleisen. The semantics of future and an application. *J. Funct. Program.*, 9(1):1–31, 1999. doi:10.1017/S0956796899003329.

34 Álvaro García-Pérez and Pablo Nogueira. The full-reducing Krivine abstract machine KN simulates pure normal-order reduction in lockstep: A proof via corresponding calculus. *J. Funct. Program.*, 29:e7, 2019. doi:10.1017/S0956796819000017.

35 Álvaro García-Pérez, Pablo Nogueira, and Juan José Moreno-Navarro. Deriving the full-reducing Krivine machine from the small-step operational semantics of normal order. In Ricardo Peña and Tom Schrijvers, editors, *15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013*, pages 85–96. ACM, 2013. doi:10.1145/2505879.2505887.

36 Simon L. Peyton Jones. Implementing lazy functional languages on stock hardware: The Spineless Tagless G-machine. *J. Funct. Program.*, 2(2):127–202, 1992.

37 David A. Kranz, Robert H. Halstead Jr., and Eric Mohr. Mul-t: A high-performance parallel Lisp. In Richard L. Wexelblat, editor, *Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation (PLDI), Portland, Oregon, USA, June 21-23, 1989*, pages 81–90. ACM, 1989.

38 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

39 Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

40 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 145–155. IEEE Computer Society, 2008.

41 Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.

42 Joachim Niehren, Jan Schwinghammer, and Gert Smolka. A concurrent lambda calculus with futures. *Theor. Comput. Sci.*, 364(3):338–356, 2006. doi:10.1016/J.TCS.2006.08.016.

43 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO Theor. Informatics Appl.*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

44 Gordon D. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, September 1981.

45 Kristoffer H. Rose. Explicit substitutions – tutorial & survey. BRICS Lecture Series LS-96-3, DAIMI, Aarhus, Denmark, September 1996.

46 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014.

47 Davide Sangiorgi. Bisimulation in higher-order process calculi. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods and Calculi, Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94) San Miniato, Italy, 6-10 June, 1994*, volume A-56 of *IFIP Transactions*, pages 207–224. North-Holland, 1994.

48 Peter Sestoft. Deriving a lazy abstract machine. *J. Funct. Program.*, 7(3):231–264, 1997. doi:10.1017/s0956796897002712.

49 C.P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, 1971.