

Distilling Abstract Machines

(Extended Abstract)

Beniamino Accattoli

Carnegie Mellon University & Università di Bologna

Pablo Barenbaum

University of Buenos Aires – CONICET

Damiano Mazza

CNRS, UMR 7030, LIPN, Université Paris 13, Sorbonne Paris Cité

In the theory of higher-order programming languages, abstract machines and explicit substitutions are two tools used to model the execution of programs on real machines while omitting many details of the actual implementation. Abstract machines can usually be seen as evaluation strategies in calculi of explicit substitutions (see at least [25, 11, 31, 16]), that can in turn be interpreted as small-step cut-elimination strategies in sequent calculi [9].

Another tool providing a fine analysis of higher-order evaluation is linear logic, especially via the new perspectives on cut-elimination provided by *proof nets*, its graphical syntax. Explicit substitutions (ES) have been connected to linear logic by Kesner and co-authors in a sequence of works [21, 27, 28], culminating in the *linear substitution calculus* (LSC), a new formalism with ES behaviorally isomorphic to proof nets (introduced in [5], developed in [6, 1, 4, 2, 3], and bearing similarities with calculi by De Bruijn [13], Nederpelt [37], and Milner [36]). Since linear logic can model all evaluation schemes (call-by-name/value/need) [33], the LSC can express them modularly, by minor variations on rewriting rules and evaluation contexts. In this paper we revisit the relationship between environment-based abstract machines and ES. Traditionally, calculi with ES simulate machines. The LSC, instead, distills them.

A Bird’s Eye View. In a simulation, every machine transition is simulated by some steps in the calculus with ES. In a distillation—that will be a formal concept, not just an analogy—only some of the machine transitions are simulated, while the others are mapped to the structural equivalence of the calculus, a specific trait of the LSC. Now, structural equivalence commutes with evaluation, *i.e.* it can be postponed. Thus, the transitions sent on the structural congruence fade away, without compromising the result of evaluation. Additionally, we show that machine executions and their distilled representation in the LSC have the same asymptotic length, *i.e.* the distillation process preserves the complexity of evaluation. The main point is that the LSC is an arguably simpler than abstract machines, and also—as we will show—it can uniformly represent many different machines in the literature.

Traditional vs Contextual ES. Traditional calculi with ES (see [26] for a survey) implement β -reduction $(\lambda x.t)u \rightarrow_{\beta} t\{x \leftarrow u\}$ introducing an annotation (the explicit substitution $[x \leftarrow u]$),

$$(\lambda x.t)u \rightarrow_{\beta} t[x \leftarrow u]$$

and percolating it through the term structure,

$$\begin{aligned} (tw)[x \leftarrow u] &\rightarrow_{@} t[x \leftarrow u]w[x \leftarrow u] \\ (\lambda x.t)[y \leftarrow u] &\rightarrow_{\lambda} \lambda x.t[y \leftarrow u] \end{aligned} \tag{1}$$

until they reach variable occurrences on which they finally substitute or get garbage collected,

$$\begin{array}{lcl} x[x \leftarrow u] & \rightarrow_{\text{var}} & u \\ y[x \leftarrow u] & \rightarrow_{\neq} & y \end{array}$$

The LSC, instead, is based on a *contextual* view of evaluation and substitution, also known as *at a distance*. The idea is that one can get rid of the rules percolating through the term structure—*i.e.* $@$ and λ —by introducing contexts C (i.e. terms with a hole $\langle \cdot \rangle$) and generalizing the base cases, obtaining just two rules, *linear substitution* (ls) and *garbage collection* (gc):

$$\begin{array}{lcl} C\langle x \rangle[x \leftarrow u] & \rightarrow_{\text{ls}} & C\langle u \rangle[x \leftarrow u] \\ t[x \leftarrow u] & \rightarrow_{\text{gc}} & t \quad \text{if } x \notin \text{fv}(t) \end{array}$$

Dually, the rule creating substitutions (B) is generalized to act up to a context of substitutions $[\dots \leftarrow \dots] := [x_1 \leftarrow w_1] \dots [x_k \leftarrow w_k]$ obtaining rule dB (B at a distance):

$$(\lambda x.t)[\dots \leftarrow \dots]u \rightarrow_{\text{dB}} t[x \leftarrow u][\dots \leftarrow \dots]$$

Logical Perspective on the LSC. From a sequent calculus point of view, rules $@$ and λ , corresponding to *commutative* cut-elimination cases, are removed and integrated—via the use of contexts—directly in the definition of the *principal* cases B, var and $\neq$, obtaining the contextual rules dB, ls, and gc. This is the analogous for terms of the removal of commutative cases provided by proof nets. From a linear logic point of view, $\rightarrow_{\text{dB}}$ can be identified with the multiplicative cut-elimination case $\rightarrow_{\text{m}}$, while $\rightarrow_{\text{ls}}$ and $\rightarrow_{\text{gc}}$ correspond to exponential cut-elimination. Actually, garbage collection has a special status, as it can always be postponed. We will then identify exponential cut-elimination $\rightarrow_{\text{e}}$ with linear substitution $\rightarrow_{\text{ls}}$ alone.

The LSC has a simple meta-theory, and is halfway between traditional calculi with ES—with whom it shares the small-step dynamics—and λ -calculus—of which it retains most of the simplicity.

Distilling Abstract Machines. Abstract machines implement the traditional approach to ES, by

1. *Weak Evaluation*: forbidding reduction under abstraction (no rule $\rightarrow_{\lambda}$ in (1)),
2. *Evaluation Strategy*: looking for redexes according to some notion of weak evaluation context E ,
3. *Context Representation*: using environments e (aka lists of substitutions) and stacks π (lists of terms) to keep track of the current evaluation context.

The LSC distills—*i.e.* factorizes—abstract machines. The idea is that one can represent the strategy of an abstract machine by directly plugging the evaluation context in the contextual substitution/exponential rule, obtaining:

$$E\langle x \rangle[x \leftarrow u] \xrightarrow{\text{e}} E\langle u \rangle[x \leftarrow u]$$

and factoring out the parts of the machine that just look for the next redex to reduce. By defining $\rightarrow$ as the closure of $\xrightarrow{\text{e}}$ and $\rightarrow_{\text{m}}$ by evaluation contexts E , one gets a clean representation of the machine strategy.

The mismatch between the two approaches is in rule $\rightarrow_{@}$, that contextually—by nature—cannot be captured. In order to get out of this *cul-de-sac*, the very idea of *simulation* of an abstract machine must be refined to that of *distillation*.

The crucial observation is that the equivalence $\equiv$ induced by $\rightarrow_{@} \cup \rightarrow_{gc}$ has the same special status of $\rightarrow_{gc}$, *i.e.* it can be postponed without affecting reduction lengths. More abstractly, $\equiv$ is a *strong bisimulation* with respect to $\multimap$, *i.e.* it verifies (note *one step to one step*, and *viceversa*)

$$\begin{array}{ccc} t \multimap r & & t \multimap r \\ \equiv & \Rightarrow \exists q \text{ s.t. } & \equiv \\ u & & u \multimap q \end{array}$$

and

$$\begin{array}{ccc} t & & t \multimap r \\ \equiv & \Rightarrow \exists r \text{ s.t. } & \equiv \\ u \multimap q & & u \multimap q \end{array}$$

Now, $\equiv$ can be considered as a *structural equivalence* on the language. Indeed, the strong bisimulation property states that the transformation expressed by $\equiv$ is irrelevant with respect to $\multimap$, in particular $\equiv$ -equivalent terms have $\multimap$ -evaluations of the same length ending in $\equiv$ -equivalent terms (and this holds even locally).

Abstract machines then are *distilled*: the logically relevant part of the substitution process is retained by $\multimap$ while both the search of the redex $\rightarrow_{@}$ and garbage collection $\rightarrow_{\neq}$ are isolated into the equivalence $\equiv$. Essentially, $\multimap$ captures principal cases of cut-elimination while $\equiv$ encapsulate the commutative ones (plus garbage collection, corresponding to principal cut-elimination involving weakenings).

Case Studies. We will analyze along these lines many abstract machines. Some are standard (KAM [29], CEK [23], ZINC [32]), some are new (MAM, WAM), and of others we provide simpler versions (SECD [30], Lazy KAM [16, 20], Sestoft's [39]). The previous explanation is a sketch of the distillation of the KAM, but the approach applies *mutatis mutandis* to all the other machines, encompassing most realizations of call-by-name, call-by-value, and call-by need evaluation. The main contribution of the paper is indeed a modular *contextual* theory of abstract machines. We start by distilling some standard cases, and then rationally reconstruct and simplify non-trivial machines as the SECD, the lazy KAM, and Sestoft's abstract machine for call-by-need (deemed SAM), by enlightening their mechanisms as different encoding of evaluation contexts, modularly represented in the LSC.

Call-by-Need. Along the way, we show that the contextual (or *at a distance*) approach of the LSC naturally leads to simple machines with just one global environment, as the newly introduced MAM (M for Milner). Such a feature is then showed to be a key ingredient of call-by-need machines, by using it to introduce a new and simple call-by-need machine, the WAM (W for Wadsworth), and then showing how to obtain (simplifications of) the Lazy KAM and the SAM by simple tweaks.

Distillation Preserves Complexity. It is natural to wonder what is lost in the distillation process. What is the asymptotic impact of distilling machine executions into $\multimap$? Does it affect in any way the complexity of evaluation? We will show that *nothing is lost*, as machine executions are only linearly longer than $\multimap$. More precisely, they are *bilinear*, *i.e.* they are linear in 1) the length of $\multimap$, and in 2) the size $|t|$ of the starting term t . In other words, the search of redexes and garbage collection can be safely ignored in quantitative (time) analyses, *i.e.* the LSC and $\multimap$ provide a complexity-preserving abstraction of abstract machines. While in call-by-name and call-by-value such an analysis follows from an easy local property of machine executions, the call-by-need case is subtler, as such a local property does not hold and bilinearity can be established only via a global analysis.

Linear Logic and Weak Linear Head Reduction. Beyond the contextual view, our work also unveils a deep connection between abstract machines and linear logic. The strategies modularly encoding the various machines (generically noted $\multimap$ and parametric in a fixed notion of evaluation contexts) are in fact call-by-name/value/need versions of *weak linear head reduction* (WLHR), a fundamental notion in the theory of linear logic [2, 35, 17, 22, 15]. This insight — due to Danos and Regnier for the KAM [18]—is not ours, but we develop it in a simpler and tighter way, modularly lifting it to many other abstract machines.

Call-by-Name. The call-by-name case (catching the KAM and the new MAM) is in fact special, as our distillation theorem has three immediate corollaries, following from results about WLHR in the literature:

1. *Invariance*: it implies that the length of a KAM/MAM execution is an invariant time cost model (*i.e.* polynomially related to, say, Turing machines, in both directions), given that in [4] the same is shown for WLHR.
2. *Evaluation as Communication*: we implicitly establish a link between the KAM/MAM and the π -calculus, given that the evaluation of a term via WLHR is isomorphic to evaluation via Milner’s encoding in the π -calculus [2].
3. *Plotkin’s Approach*: our study complements the recent [3], where it is shown that WLHR is a standard strategy of the LSC. The two works together provide the lifting to explicit substitutions of Plotkin’s approach of relating a machine (the SECD machine in that case, the KAM/MAM in ours) and a calculus (the call-by-value λ -calculus and the LSC, respectively) via a standardization theorem and a standard strategy [38].

Related Work. Beyond the already cited works, Danvy and coauthors have studied abstract machines in a number of works and ways (see at least [7, 8, 12, 11, 20]). What here we call *commutative transitions* essentially corresponds to what Danvy and Nielsen call *decompose* phase in [19]. The call-by-need calculus we use—that is a contextual re-formulation of Maraist, Odersky, and Wadler’s calculus [34]—is a novelty of this paper. It is simpler than both Ariola and Felleisen’s [10] and Maraist, Odersky, and Wadler’s calculi because it does not need any re-association axioms. Morally, it is a version with let-bindings (avatars of ES) of Chang and Felleisen’s calculus [14]. A similar calculus is used by Danvy and Zerny in [20], where, similarly to us, they explore various call-by-need machines in relation to various calculi. The differences are that 1) they focus on the contrast between store-based and storeless machines, 2) they do not stress the value of contexts as we do here, 3) they do not connect their study to linear logic. Another call-by-need machine, with whom we do not deal with, appears in [24].

References

- [1] Beniamino Accattoli (2012): *An Abstract Factorization Theorem for Explicit Substitutions*. In: *RTA*, pp. 6–21. Available at <http://dx.doi.org/10.4230/LIPIcs.RTA.2012.6>.
- [2] Beniamino Accattoli (2013): *Evaluating functions as processes*. In: *TERMGRAPH*, pp. 41–55. Available at <http://dx.doi.org/10.4204/EPTCS.110.6>.
- [3] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner & Carlos Lombardi (2014): *A nonstandard standardization theorem*. In: *POPL*, pp. 659–670. Available at <http://doi.acm.org/10.1145/2535838.2535886>.

- [4] Beniamino Accattoli & Ugo Dal Lago (2012): *On the Invariance of the Unitary Cost Model for Head Reduction*. In: *RTA*, pp. 22–37. Available at <http://dx.doi.org/10.4230/LIPIcs.RTA.2012.22>.
- [5] Beniamino Accattoli & Delia Kesner (2010): *The Structural λ -Calculus*. In: *CSL*, pp. 381–395. Available at http://dx.doi.org/10.1007/978-3-642-15205-4_30.
- [6] Beniamino Accattoli & Luca Paolini (2012): *Call-by-Value Solvability, revisited*. In: *FLOPS*, pp. 4–16. Available at http://dx.doi.org/10.1007/978-3-642-29822-6_4.
- [7] Mads Sig Ager, Dariusz Biernacki, Olivier Danvy & Jan Midtgaard (2003): *A functional correspondence between evaluators and abstract machines*. In: *PPDP*, pp. 8–19. Available at <http://doi.acm.org/10.1145/888251.888254>.
- [8] Mads Sig Ager, Olivier Danvy & Jan Midtgaard (2004): *A functional correspondence between call-by-need evaluators and lazy abstract machines*. *Inf. Process. Lett.* 90(5), pp. 223–232. Available at <http://dx.doi.org/10.1016/j.ipl.2004.02.012>.
- [9] Zena M. Ariola, Aaron Bohannon & Amr Sabry (2009): *Sequent calculi and abstract machines*. *ACM Trans. Program. Lang. Syst.* 31(4). Available at <http://doi.acm.org/10.1145/1516507.1516508>.
- [10] Zena M. Ariola & Matthias Felleisen (1997): *The Call-By-Need lambda Calculus*. *J. Funct. Program.* 7(3), pp. 265–301. Available at <http://journals.cambridge.org/action/displayAbstract?aid=44089>.
- [11] Malgorzata Biernacka & Olivier Danvy (2007): *A concrete framework for environment machines*. *ACM Trans. Comput. Log.* 9(1). Available at <http://doi.acm.org/10.1145/1297658.1297664>.
- [12] Malgorzata Biernacka & Olivier Danvy (2007): *A syntactic correspondence between context-sensitive calculi and abstract machines*. *Theor. Comput. Sci.* 375(1-3), pp. 76–108. Available at <http://dx.doi.org/10.1016/j.tcs.2006.12.028>.
- [13] Nicolaas G. de Bruijn (1987): *Generalizing Automath by Means of a Lambda-Typed Lambda Calculus*. In: *Mathematical Logic and Theoretical Computer Science, Lecture Notes in Pure and Applied Mathematics* 106, Marcel Dekker, pp. 71–92.
- [14] Stephen Chang & Matthias Felleisen (2012): *The Call-by-Need Lambda Calculus, Revisited*. In: *ESOP*, pp. 128–147. Available at http://dx.doi.org/10.1007/978-3-642-28869-2_7.
- [15] Pierre Clairambault (2011): *Estimation of the Length of Interactions in Arena Game Semantics*. In: *FOS-SACS*, pp. 335–349. Available at http://dx.doi.org/10.1007/978-3-642-19805-2_23.
- [16] Pierre Crégut (2007): *Strongly reducing variants of the Krivine abstract machine*. *Higher-Order and Symbolic Computation* 20(3), pp. 209–230. Available at <http://dx.doi.org/10.1007/s10990-007-9015-z>.
- [17] Vincent Danos, Hugo Herbelin & Laurent Regnier (1996): *Game Semantics & Abstract Machines*. In: *LICS*, pp. 394–405. Available at <http://doi.ieeecomputersociety.org/10.1109/LICS.1996.561456>.
- [18] Vincent Danos & Laurent Regnier (2004): *Head Linear Reduction*. Technical Report.
- [19] Olivier Danvy & Lasse R. Nielsen (2004): *Refocusing in Reduction Semantics*. Technical Report RS-04-26, BRICS.
- [20] Olivier Danvy & Ian Zerny (2013): *A synthetic operational account of call-by-need evaluation*. In: *PPDP*, pp. 97–108. Available at <http://doi.acm.org/10.1145/2505879.2505898>.
- [21] Roberto Di Cosmo, Delia Kesner & Emmanuel Polonovski (2003): *Proof Nets And Explicit Substitutions*. *Math. Str. in Comput. Sci.* 13(3), pp. 409–450. Available at <http://dx.doi.org/10.1017/S0960129502003791>.
- [22] Thomas Ehrhard & Laurent Regnier (2006): *Böhm Trees, Krivine’s Machine and the Taylor Expansion of Lambda-Terms*. In: *CiE*, pp. 186–197. Available at http://dx.doi.org/10.1007/11780342_20.
- [23] Matthias Felleisen & Daniel P. Friedman (1986): *Control operators, the SECD-machine, and the lambda-calculus*. In: *3rd Working Conference on the Formal Description of Programming Concepts*.
- [24] Ronald Garcia, Andrew Lumsdaine & Amr Sabry (2009): *Lazy evaluation and delimited control*. In: *POPL*, pp. 153–164. Available at <http://doi.acm.org/10.1145/1480881.1480903>.

- [25] Thérèse Hardin & Luc Maranget (1998): *Functional Runtime Systems Within the Lambda-Sigma Calculus*. *J. Funct. Program.* 8(2), pp. 131–176. Available at <http://journals.cambridge.org/action/displayAbstract?aid=44157>.
- [26] Delia Kesner (2009): *A Theory of Explicit Substitutions with Safe and Full Composition*. *Logical Methods in Computer Science* 5(3). Available at <http://arxiv.org/abs/0905.2539>.
- [27] Delia Kesner & Stéphane Lengrand (2007): *Resource operators for lambda-calculus*. *Inf. Comput.* 205(4), pp. 419–473. Available at <http://dx.doi.org/10.1016/j.ic.2006.08.008>.
- [28] Delia Kesner & Fabien Renaud (2009): *The Prismoid of Resources*. In: *MFCS*, pp. 464–476. Available at http://dx.doi.org/10.1007/978-3-642-03816-7_40.
- [29] Jean-Louis Krivine (2007): *A call-by-name lambda-calculus machine*. *Higher-Order and Symbolic Computation* 20(3), pp. 199–207. Available at <http://dx.doi.org/10.1007/s10990-007-9018-9>.
- [30] Peter John Landin (1964): *The Mechanical Evaluation of Expressions*. *The Computer Journal* 6(4), pp. 308–320, doi:10.1093/comjnl/6.4.308. Available at <http://dx.doi.org/10.1093/comjnl/6.4.308>.
- [31] Frédéric Lang (2007): *Explaining the lazy Krivine machine using explicit substitution and addresses*. *Higher-Order and Symbolic Computation* 20(3), pp. 257–270. Available at <http://dx.doi.org/10.1007/s10990-007-9013-1>.
- [32] Xavier Leroy (1990): *The ZINC experiment: an economical implementation of the ML language*. Technical report 117, INRIA. Available at <http://gallium.inria.fr/~xleroy/publi/ZINC.pdf>.
- [33] John Maraist, Martin Odersky, David N. Turner & Philip Wadler (1999): *Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus*. *Theor. Comput. Sci.* 228(1-2), pp. 175–210. Available at [http://dx.doi.org/10.1016/S0304-3975\(98\)00358-2](http://dx.doi.org/10.1016/S0304-3975(98)00358-2).
- [34] John Maraist, Martin Odersky & Philip Wadler (1998): *The Call-by-Need Lambda Calculus*. *J. Funct. Program.* 8(3), pp. 275–317. Available at <http://journals.cambridge.org/action/displayAbstract?aid=44169>.
- [35] Gianfranco Mascari & Marco Pedicini (1994): *Head Linear Reduction and Pure Proof Net Extraction*. *Theor. Comput. Sci.* 135(1), pp. 111–137. Available at [http://dx.doi.org/10.1016/0304-3975\(94\)90263-1](http://dx.doi.org/10.1016/0304-3975(94)90263-1).
- [36] Robin Milner (2007): *Local Bigraphs and Confluence: Two Conjectures*. *Electr. Notes Theor. Comput. Sci.* 175(3), pp. 65–73. Available at <http://dx.doi.org/10.1016/j.entcs.2006.07.035>.
- [37] Robert. P. Nederpelt (1992): *The fine-structure of lambda calculus*. Technical Report CSN 92/07, Eindhoven Univ. of Technology.
- [38] Gordon D. Plotkin (1975): *Call-by-Name, Call-by-Value and the lambda-Calculus*. *Theor. Comput. Sci.* 1(2), pp. 125–159. Available at [http://dx.doi.org/10.1016/0304-3975\(75\)90017-1](http://dx.doi.org/10.1016/0304-3975(75)90017-1).
- [39] Peter Sestoft (1997): *Deriving a Lazy Abstract Machine*. *J. Funct. Program* 7(3), pp. 231–264.