Inter-deriving Semantic Artifacts for Object-Oriented Programming (Extended Abstract)

Olivier Danvy and Jacob Johannsen

Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
{danvy,cnn}@daimi.au.dk
http://www.daimi.au.dk/~{danvy,cnn}

Abstract. We present a new abstract machine for Abadi and Cardelli's untyped calculus of objects. What is special about this semantic artifact (i.e., man-made construct) is that is mechanically corresponds to both the reduction semantics (i.e., small-step operational semantics) and the natural semantics (i.e., big-step operational semantics) specified in Abadi and Cardelli's monograph. This abstract machine therefore embodies the soundness of Abadi and Cardelli's reduction semantics and natural semantics relative to each other.

To move closer to actual implementations, which use environments rather than actual substitutions, we then represent object methods as closures and in the same inter-derivational spirit, we present three new semantic artifacts: a reduction semantics for a version of Abadi and Cardelli's untyped calculus of objects with explicit substitutions, an environment-based abstract machine, and a natural semantics (i.e., an interpreter) with environments. These three new semantic artifacts mechanically correspond to each other, and furthermore, they are coherent with the previous ones since as we show, the two abstract machines are bisimilar. Overall, though, the significance of these artifacts lies in them not having been designed from scratch and then proved correct: instead, they were mechanically inter-derived.

1 Introduction

Our goal here is to apply Danvy et al.'s 'syntactic correspondence' and 'functional correspondence' [3, 8, 12, 21, 37, 38, 39], which were developed for the λ-calculus with effects, to Abadi and Cardelli's untyped calculus of objects [1, Chapter 6].

1.1 Background and First Contribution

The syntactic correspondence between reduction semantics and abstract machines: This correspondence mechanically links a reduction semantics (i.e., a small-step operational semantics with an explicit representation of reduction contexts [27, 28]) to an abstract machine. In such a reduction semantics, evaluation is implemented by iterated reduction, and the corresponding reduction sequence can be depicted as follows:

$$\begin{tikzcd}
\circ \arrow[rrr, dotted, "\text{reduction step}"] \arrow[dr, "\mathsf{decompose}"] & & & \circ \arrow[rrr, dotted, "\text{reduction step}"] \arrow[dr, "\mathsf{decompose}"] & & & \circ \arrow[rr, dotted, no head] \arrow[dr, "\mathsf{decompose}"] & & {} \\
& \circ \arrow[r, "\mathsf{contract}"'] & \circ \arrow[ur, "\mathsf{plug}"] & & \circ \arrow[r, "\mathsf{contract}"'] & \circ \arrow[ur, "\mathsf{plug}"] & & \circ \arrow[r, "\mathsf{contract}"'] & {}
\end{tikzcd}$$

At each step, a non-value term is decomposed into a reduction context and a potential redex. If the potential redex is an actual one (i.e., if it is not stuck), it is contracted. The contractum is then plugged into the context, yielding the next term in the reduction sequence.

At each step, the function plug therefore constructs an intermediate term. In the course of evaluation, this term is then immediately decomposed by the subsequent call to decompose. The composition of plug and decompose can thus be replaced by a more efficient function, refocus, that directly goes from redex site to redex site in the reduction sequence:

$$\begin{tikzcd}
\circ \arrow[dr, "\mathsf{decompose}"] & & & \circ \arrow[dr, "\mathsf{decompose}"] & & & \circ \arrow[dr, "\mathsf{decompose}"] & & \\
{} \arrow[r, dashed] & \circ \arrow[r, "\mathsf{contract}"'] & \circ \arrow[ur, "\mathsf{plug}"] \arrow[rr, dashed, "\mathsf{refocus}"'] & & \circ \arrow[r, "\mathsf{contract}"'] & \circ \arrow[ur, "\mathsf{plug}"] \arrow[rr, dashed, "\mathsf{refocus}"'] & & \circ \arrow[r, "\mathsf{contract}"'] & {}
\end{tikzcd}$$

As shown by Danvy and Nielsen [25], refocus can take the form of a state-transition function. Therefore, together with contract, the result is an abstract machine. And what is remarkable here is that the abstract machines obtained by refocusing are not unnatural ones.

In fact, this syntactic correspondence between reduction semantics and abstract machines has made it possible to obtain a variety of abstract machines for the λ-calculus, be it pure or with effects. Some of these machines were independently known and some others are new [10, 11]. Symmetrically, it also has made it possible to exhibit the calculi and the reduction strategies (in the form of reduction semantics) corresponding to pre-existing abstract machines.

The functional correspondence between natural semantics and abstract machines: This correspondence mechanically links a natural semantics (i.e., a big-step operational semantics, as implemented by an interpreter [32, 40]) to an abstract machine. It is based on the framework initiated by Reynolds in his seminal article "Definitional Interpreters for Higher-Order Programming Languages" [41]. In a nutshell, successively transforming an interpreter using closure conversion, transformation into continuation-passing style (CPS), and defunctionalization yields an abstract machine [4]. And what is remarkable here is that the abstract machines obtained by CPS transformation and defunctionalization are not unnnatural ones.

In fact, this functional correspondence between natural semantics and abstract machines has made it possible to obtain a variety of abstract machines for the λ-calculus, be it pure or with effects. Some of these machines were independently known and some others are new [5, 6, 9]. Symmetrically, it also has made it possible to exhibit the interpreter (in the form of a natural semantics) corresponding to pre-existing abstract machines.

Our starting point here: Together, the syntactic and the functional correspondences make it possible to connect three semantic artifacts (i.e., man-made constructs) soundly: reduction semantics, abstract machines, and natural semantics. Better: the correspondence make it possible to *inter-derive* these semantics (or more precisely, their representation as functional programs), mechanically. This inter-derivation contrasts with defining several semantics, which requires work, and proving their soundness relative to each other, which requires more work. As Rod Burstall soberly put it once, "theory should be call by need." Our goal here is to apply these two correspondences to Abadi and Cardelli's untyped calculus of objects.

Abadi and Cardelli's untyped calculus of objects: Abadi and Cardelli's monograph "A Theory of Objects" is a landmark. Nowadays it provides standard course material about object-oriented languages and programming. Of interest to us here is its Chapter 6 where an untyped calculus of objects, the ς-calculus, is developed in the same spirit as its predecessor, the λ-calculus [7, 14], which was initially developed as an untyped calculus of functions. The ς-calculus is specified with a reduction semantics, for a given reduction order, and with a natural semantics, for a given evaluation order. A soundness theorem (Proposition 6.2-3, page 64) links the two semantics. Operational reduction is also shown to be complete with respect to many-step reduction with a completeness theorem (Theorem 6.2-4, page 65). Soundness matters because it shows that the interpreter implementing the natural semantics is faithful to the reduction semantics and vice versa. Completeness matters because it shows that the reductions may be meaningfully re-ordered, thus enabling practical optimizations such as constant propagation and more generally partial evaluation [15, 31].

First contribution: Using the syntactic correspondence, we exhibit an abstract machine that embodies the reduction semantics of the ς-calculus and its reduction strategy. Using the functional correspondence, we exhibit an abstract machine that embodies the natural semantics of the ς-calculus and its evaluation strategy. The two abstract machines are identical. This abstract machine, which is new, therefore mediates between the reduction semantics and the natural semantics, and practically confirms the soundness theorem:

$$\begin{tikzcd}[column sep=huge] \begin{array}{c}\text{reduction}\\ \text{semantics}\\ \text{for the}\\ \varsigma\text{-calculus}\end{array} \arrow[r, "\text{syntactic}", "\text{correspondence}"'] & \begin{array}{c}\text{abstract}\\ \text{machine}\\ \text{for the}\\ \varsigma\text{-calculus}\end{array} & \begin{array}{c}\text{natural}\\ \text{semantics}\\ \text{for the}\\ \varsigma\text{-calculus}\end{array} \arrow[l, "\text{functional}"', "\text{correspondence}"] \end{tikzcd}$$

1.2 Further Background and Contributions

Substitutions vs. environments: Practical implementations of the λ-calculus do not use actual substitutions. Instead, they use 'environments,' which are mappings representing delayed substitutions, and represent functions with 'closures,' which are pairs of terms and environments [34]. In such practical implementations, an identifier is not a thing to be substituted by a term, but a thing to be looked up in the current environment. At the turn of the 1990's [17], Curien proposed a 'calculus of closures,' the λρ-calculus, to account for this implementation strategy of the λ-calculus, and explicit substitutions were born [2, 18, 43]. Both the syntactic and the functional correspondences have been applied to calculi of explicit substitutions, environment-based abstract machines, and natural semantics using environments [4, 10].

Abadi and Cardelli's untyped calculus of objects with methods as closures: We present a version of the ς-calculus with explicit substitutions, the $\varsigma\rho$-calculus. Instead of performing substitution when invoking a method, we represent methods as closures. We state three semantic artifacts for the $\varsigma\rho$-calculus: a natural semantics, an abstract machine, and a reduction semantics.

Contributions: Using the syntactic correspondence, we exhibit an environment-based abstract machine that embodies the reduction semantics of the $\varsigma\rho$-calculus and its reduction strategy. Using the functional correspondence, we exhibit an environment-based abstract machine that embodies the natural semantics of the $\varsigma\rho$-calculus and its evaluation strategy. Again, the two abstract machines are identical, which establishes the soundness of the reduction semantics and of the natural semantics for the $\varsigma\rho$-calculus relative to each other. We then show that this environment-based abstract machine and the abstract machine with actual substitutions from Section 1.1 are bisimilar, which establishes the coherence of the $\varsigma\rho$-calculus with respect to the ς-calculus:

$$\begin{tikzcd}
\text{reduction semantics for the ς-calculus} \arrow[r, "\text{syntactic}", "\text{correspondence}"'] & \text{abstract machine for the ς-calculus} \arrow[d, dashed, leftrightarrow, "\text{bisimilarity}"] & \text{natural semantics for the ς-calculus} \arrow[l, "\text{functional}"', "\text{correspondence}"] \\
\text{reduction semantics for the $\varsigma\rho$-calculus} \arrow[r, "\text{syntactic}", "\text{correspondence}"'] & \text{abstract machine for the $\varsigma\rho$-calculus} & \text{natural semantics for the $\varsigma\rho$-calculus} \arrow[l, "\text{functional}"', "\text{correspondence}"]
\end{tikzcd}$$

As for having a completeness theorem for the $\varsigma\rho$-calculus, Melliès's proof applies mutatis mutandis [1, Theorem 6.2-4, page 65].

1.3 Overview

In Section 2, we remind the reader of the ς-calculus (Section 2.1) and we present its reduction semantics; through the syntactic correspondence, we obtain the corresponding abstract machine (Section 2.2). Through the functional correspondence, we then present the natural semantics corresponding to this abstract machine (Section 2.3). This natural semantics coincides with Abadi and Cardelli's. In Section 3, we introduce the $\varsigma\rho$-calculus, which is a version of the ς-calculus with explicit substitutions where methods are represented with closures, and we specify it with a natural semantics that uses environments (Section 3.1); through the functional correspondence, we obtain the corresponding abstract machine (Section 3.2). Through the syntactic correspondence, we then present the reduction semantics corresponding to this abstract machine (Section 3.3). In Section 4.1, we present a mapping from $\varsigma\rho$-closures to ς-terms that performs the actual substitutions that were delayed by the given environments in the given terms. In Section 4.2, using this mapping, we show that the two abstract machines are bisimilar, which establishes a coherence between the three semantic artifacts for the ς-calculus and the three semantic artifacts for the $\varsigma\rho$-calculus. We then review related work in Section 5 and conclude in Section 6.

Prerequisites: We assume the reader to be mildly familiar with Sections 6.1 and 6.2 of Abadi and Cardelli's monograph [1] and with the concepts of reduction semantics (BNF of terms and of reduction contexts, notion of redex, one-step reduction, evaluation as iterated reduction), of abstract machines (initial, intermediate, and final states, and state-transition functions), of natural semantics (interpreters as evaluation functions), and of bisimulation. As for the syntactic and functional correspondences, the unfamiliar reader can just flip through Danvy's invited paper at WRS'04 [20] or through Danvy and Millikin's recent note about small-step and big-step abstract machines [23] for what is not self-explanatory.

2 Abadi and Cardelli's Untyped Calculus of Objects: The ς-Calculus

We consider in turn a reduction semantics for the ς-calculus (Section 2.1), the corresponding abstract machine (Section 2.2), and the corresponding natural semantics (Section 2.3).

2.1 A Reduction Semantics

BNF of terms and of values: An object is a collection of named attributes. Names are labels and all labels are distinct within each object. All attributes are methods with a bound variable representing self (and to be bound to the host object at invocation time) and a body whose execution yields a result.

$$\begin{array}{ll}(\mathsf{Term}) & t ::= x \mid [l = \varsigma(x)t,\ \ldots,\ l = \varsigma(x)t] \mid t.l \mid t.l \Leftarrow \varsigma(x)t \\ (\mathsf{Value}) & v ::= [l = \varsigma(x)t,\ \ldots,\ l = \varsigma(x)t]\end{array}$$

This grammar for terms defines the same language as in Abadi and Cardelli's book but it uses a more uniform naming convention.

NB: Occasionally, we index a value by its number of methods, as in $v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}]$.

Notion of redex: Methods can be invoked or updated [1, Definition 6.2-1 (1)]. Here is the grammar of potential redexes:

$$pr ::= v.l \mid v.l \Leftarrow \varsigma(x)t$$

The contraction rules read as follows:

$$\begin{array}{rl} v^n.l_j \rightarrowtail & t_j\{v^n/x_j\} \\ & \text{if } 1 \le j \le n, \text{ where } v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}] \\ v^n.l_j \Leftarrow \varsigma(x)t \rightarrowtail & [l_j = \varsigma(x)t,\ l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}\backslash\{j\}}] \\ & \text{if } 1 \le j \le n, \text{ where } v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}] \end{array}$$

A potential redex is an actual one when its side conditions are satisfied, and contraction can take place. Otherwise, the potential redex is stuck.

BNF of reduction contexts: The following grammar for reduction contexts does not occur in Abadi and Cardelli's book but it plausibly reflects the 'evaluation strategy of the sort commonly used in programming languages' [1, Section 6.2.4, page 63]:

$$(\mathsf{Context}) \qquad C ::= [\,] \mid C[[\,].l] \mid C[[\,].l \Leftarrow \varsigma(x)t]$$

Lemma 1 (Unique decomposition). *Any term which is not a value can be uniquely decomposed into a reduction context and a potential redex.*

One is then in position to define a decomposition function mapping a term to either a value or to a reduction context and a potential redex, a contraction function mapping an actual redex to its contractum, and a plug function mapping a reduction context and a term to a term. Thus equipped, one can define a one-step reduction function (noted $\rightarrow$ below) and then an evaluation function as the iteration of the one-step reduction function (noted $\rightarrow^*$ below). We have implemented and copiously tested this reduction semantics (as well as all the other semantic artifacts of this article) in Standard ML.

2.2 The Corresponding Abstract Machine

Applying the syntactic correspondence (i.e., calculating the refocus function) yields the following eval/apply abstract machine [36]:

$$\begin{array}{rcl}
\langle v,\, C\rangle & \Rightarrow_S & \langle C,\, v\rangle \\
\langle t.l,\, C\rangle & \Rightarrow_S & \langle t,\, C[[\;].l]\rangle \\
\langle t.l \Leftarrow \varsigma(x)t',\, C\rangle & \Rightarrow_S & \langle t,\, C[[\;].l \Leftarrow \varsigma(x)t']\rangle \\
 & & \\
\langle [\;],\, v\;\rangle & \Rightarrow_S & v \\
\langle C[[\;].l_j],\, v^n\rangle & \Rightarrow_S & \langle t_j\{v^n/x_j\},\, C\rangle \\
 & & \text{if } 1 \leq j \leq n, \text{ where } v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}] \\
\langle C[[\;].l_j \Leftarrow \varsigma(x)t],\, v^n\rangle & \Rightarrow_S & \langle C,\, [l_j = \varsigma(x)t, l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}\setminus\{j\}}]\rangle \\
 & & \text{if } 1 \leq j \leq n, \text{ where } v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}]
\end{array}$$

This machine evaluates a closed term t by starting in the configuration $\langle t,\, [\;]\rangle$ and by iterating $\Rightarrow_S$ (noted $\Rightarrow_S^*$ below). It halts with a value v if it reaches a configuration $\langle [\;],\, v\rangle$ It becomes stuck if it reaches either of the configurations $\langle C[[\;].l],\, v\rangle$ or $\langle C[[\;].l \Leftarrow \varsigma(x)t],\, v\rangle$ and v does not contain a method with the label l.

The following proposition is a corollary of the soundness of refocusing:

Proposition 1 (Full correctness). *For any closed term t, $t \rightarrow^* v$ if and only if $\langle t,\, [\;]\rangle \Rightarrow_S^* v$.*

2.3 The Corresponding Natural Semantics

In Section 2.2, the function implementing the abstract machine is in defunctionalized form [24]. Refunctionalizing it [22] yields an evaluation function in continuation-passing style (CPS). Writing this evaluation function in direct style [19] yields an evaluation function that implements the following natural semantics:

$$(\mathrm{INV}_\varsigma) \quad \frac{\vdash t \leadsto v^n \qquad \vdash t_j\{v^n/x_j\} \leadsto v}{\vdash t.l_j \leadsto v} \quad \begin{array}{l}\text{if } 1 \leq j \leq n, \text{ where} \\ v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}]\end{array}$$

$$(\mathrm{UPD}_\varsigma) \quad \frac{\vdash t \leadsto v^n}{\begin{array}{r}\vdash t.l_j \Leftarrow \varsigma(x)t' \leadsto [l_j = \varsigma(x)t', \\ l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}\setminus\{j\}}]\end{array}} \quad \begin{array}{l}\text{if } 1 \leq j \leq n, \text{ where} \\ v^n = [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}]\end{array}$$

This natural semantics coincides with Abadi and Cardelli's [1, Section 6.2.4, page 64].

The following proposition is a corollary of the soundness of the CPS transformation and of defunctionalization:

Proposition 2 (Full correctness). *For any closed term t, $\langle t,\, [\;]\rangle \Rightarrow_S^* v$ if and only if $\vdash t \leadsto v$.*

2.4 Summary and Conclusion

Using the syntactic correspondence and the functional correspondence, we have mechanically derived an abstract machine that mediates between Abadi and Cardelli's reduction semantics and natural semantics for the ς-calculus and the 'evaluation strategy of the sort commonly used in programming languages.' The two derivations confirm (1) the soundness of the two semantics relative to each other and (2) the BNF of the reduction contexts we put forward in Section 2.1. They also pave the way to using closures, which we do next.

3 Object Methods as Closures: the ςρ-Calculus

We consider in turn a natural semantics for the ς-calculus with environments (Section 3.1), the corresponding environment-based abstract machine (Section 3.2), and the corresponding reduction semantics (Section 3.3). The resulting calculus is one of explicit substitutions, the ςρ-calculus.

3.1 A Natural Semantics

Let us adapt the natural semantics of Section 2.3 to operate with environments. Three changes take place:

1. The category of values changes to objects where each method holds its own environment (noted 'e'):
$$(\mathsf{Value}) \qquad v ::= [l = (\varsigma(x)t)[e],\ \ldots,\ l = (\varsigma(x)t)[e]]$$
2. The environment is defined as an association list:
$$(\mathsf{Environment}) \qquad e ::= \bullet \mid (x,\, v) \cdot e$$
and an auxiliary function *lookup* is used to look up an identifier in the current environment.
3. The evaluation judgment now reads as follows:
$$e \vdash t \rightsquigarrow v$$

Again, we occasionally index a value with the number of its methods.

The two rules from Section 2.3 are then straightforwardly adapted:

$$(\mathrm{INV}_{\varsigma\rho})\ \frac{e \vdash t \rightsquigarrow v^n \quad (x_j,\, v^n)\cdot e_j \vdash t_j \rightsquigarrow v}{e \vdash t.l_j \rightsquigarrow v} \quad \text{if } 1 \leq j \leq n, \text{ where } v^n = [l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}]$$

$$(\mathrm{UPD}_{\varsigma\rho})\ \frac{e \vdash t \rightsquigarrow v^n}{e \vdash t.l_j \Leftarrow \varsigma(x)t' \rightsquigarrow v} \quad \text{if } 1 \leq j \leq n, \text{ where } v^n = [l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}] \text{ and } v = [l_j = (\varsigma(x)t')[e],\ l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}\setminus\{j\}}]$$

We also need the following rule to convert the methods of an object literal to method closures:

$$(\mathrm{CLO}_{\varsigma\rho}) \quad \frac{}{e \vdash [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}] \rightsquigarrow [l_i = (\varsigma(x_i)t_i)[e]^{i\in\{1..n\}}]}$$

In addition, we need the following new rule to look up variables in the current environment:

$$(\text{VAR-L}_{\varsigma\rho}) \quad \frac{}{e \vdash x \rightsquigarrow v} \qquad \text{if } \mathit{lookup}\,(x,\, e) = v$$

Alternatively, and as done, e.g., in the Categorical Abstract Machine [16], one could use two rules to incrementally peel off the environment. For closed terms, x always occurs in e. For open terms, evaluation would become stuck here.

3.2 The Corresponding Abstract Machine

To apply the functional correspondence, we successively CPS-transform and defunctionalize the evaluation function implementing the natural semantics of Section 3.1. The grammar of evaluation contexts now reads as follows:

$$(\mathsf{Context}) \qquad C ::= [\;] \mid C[[\;].l] \mid C[[\;].l \Leftarrow (\varsigma(x)t)[e]]$$

All in all, the functional correspondence yields the following eval/apply abstract machine:

$$\begin{array}{rcl}
\langle x,\, e,\, C\rangle & \Rightarrow_E & \langle C,\, v\rangle \\
 & & \text{if } \mathit{lookup}\,(x,\, e) = v \\
\langle [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}],\, e,\, C\rangle & \Rightarrow_E & \langle C,\, [l_i = (\varsigma(x_i)t_i)[e]^{i\in\{1..n\}}]\rangle \\
\langle t.l,\, e,\, C\rangle & \Rightarrow_E & \langle t,\, e,\, C[[\;].l]\rangle \\
\langle t.l \Leftarrow \varsigma(x)t',\, e,\, C\rangle & \Rightarrow_E & \langle t,\, e,\, C[[\;].l \Leftarrow (\varsigma(x)t')[e]]\rangle \\
 & & \\
\langle [\;],\, v\;\rangle & \Rightarrow_E & v \\
\langle C[[\;].l_j],\, v^n\rangle & \Rightarrow_E & \langle t_j,\, (x_j,\, v^n)\cdot e_j,\, C\rangle \\
 & & \text{if } 1 \leq j \leq n, \text{ where} \\
 & & v^n = [l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}] \\
\langle C[[\;].l_j \Leftarrow (\varsigma(x)t)[e]],\, v^n\rangle & \Rightarrow_E & \langle C,\, [l_j = (\varsigma(x)t)[e], l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}\setminus\{j\}}]\rangle \\
 & & \text{if } 1 \leq j \leq n, \text{ where} \\
 & & v^n = [l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}]
\end{array}$$

This machine evaluates a closed term t by starting in the configuration $\langle t,\, \bullet,\, [\;]\rangle$ and by iterating $\Rightarrow_E$ (noted $\Rightarrow_E^*$ below). It halts with a value v if it reaches a configuration $\langle [\;],\, v\rangle$. It becomes stuck if it reaches either of the configurations $\langle C[[\;].l],\, v\rangle$ or $\langle C[[\;].l \Leftarrow (\varsigma(x)t)[e]],\, v\rangle$ and v does not contain a method with the label l.

The following proposition is a corollary of the soundness of the CPS transformation and of defunctionalization:

Proposition 3 (Full correctness). *For any closed term t, $\bullet \vdash t \rightsquigarrow v$ if and only if $\langle t, \bullet, [\,]\rangle \Rightarrow_E^* v$.*

3.3 The Corresponding Reduction Semantics

BNF of terms, of values, and of closures: The BNF of terms does not change. The BNF of values is as in Section 3.1. In addition, as in Curien's $\lambda\rho$-calculus compared to the λ-calculus, a new syntactic category appears, that of closures:

$$\textsf{(Closure)} \qquad c ::= t[e] \mid [l = (\varsigma(x)t)[e],\, \ldots,\, l = (\varsigma(x)t)[e]] \mid c.l \mid c.l \Leftarrow (\varsigma(x)t)[e]$$

Notion of redex: The two original contraction rules are adapted to closures as follows:

$$\begin{aligned} v^n.l_j &\rightarrowtail t_j[(x_j,\, v^n)\cdot e_j] \\ &\qquad \text{if } 1 \le j \le n, \text{ where } v^n = [l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}] \\ v^n.l_j \Leftarrow (\varsigma(x)t)[e] &\rightarrowtail [l_j = (\varsigma(x)t)[e], l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}\setminus\{j\}}] \\ &\qquad \text{if } 1 \le j \le n, \text{ where } v^n = [l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}] \end{aligned}$$

As could be expected, there is also a contraction rule for looking variables up in the environment:

$$\begin{aligned} x[e] &\rightarrowtail v \\ &\qquad \text{if } \mathit{lookup}\,(x,\, e) = v \end{aligned}$$

In addition, we need three contraction rules to propagate the environment inside the terms:

$$\begin{aligned} [l_i = \varsigma(x_i)t_i{}^{i\in\{1..n\}}][e] &\rightarrowtail [l_i = (\varsigma(x_i)t_i)[e]^{i\in\{1..n\}}] \\ (t.l)[e] &\rightarrowtail t[e].l \\ (t.l \Leftarrow \varsigma(x)t')[e] &\rightarrowtail t[e].l \Leftarrow (\varsigma(x)t')[e] \end{aligned}$$

The grammar of potential redexes therefore reads as follows:

$$\begin{aligned} \mathit{pr} ::= {} & v.l \mid v.l \Leftarrow (\varsigma(x)t)[e] \mid \\ & x[e] \mid [l = \varsigma(x)t,\, \ldots,\, l = \varsigma(x)t][e] \mid (t.l)[e] \mid (t.l \Leftarrow \varsigma(x)t')[e] \end{aligned}$$

BNF of reduction contexts: The grammar for reduction contexts is the same as in Section 3.2.

Lemma 2 (Unique decomposition). *Any closure which is not a value can be uniquely decomposed into a reduction context and a potential redex.*

One is then in position to define a decomposition function mapping a closure to either a value or to a reduction context and a potential redex, a contraction function mapping an actual redex to its contractum, and a plug function mapping a reduction context and a closure to a closure. Thus equipped, one can define a one-step reduction function (noted $\rightarrow$ below) and then an evaluation function as the iteration of the one-step reduction function (noted $\rightarrow^*$ below).

Applying the syntactic correspondence yields the abstract machine from Section 3.2.

The following proposition is a corollary of the soundness of refocusing:

Proposition 4 (Full correctness). *For any closed term t, $\langle t, \bullet, [\,]\rangle \Rightarrow_E^* v$ if and only if $t[\bullet] \rightarrow^* v$.*

3.4 Summary and Conclusion

On the ground that practical implementations do not use actual substitutions, we have presented an analogue of the ς-calculus, the $\varsigma\rho$-calculus, that uses explicit substitutions. We have inter-derived three semantics artifacts for the $\varsigma\rho$-calculus: a natural semantics, an abstract machine, and a reduction semantics. These specifications are more suitable to support the formalization of a compiler since programs do not change (through substitution) in the course of execution. One is then free to change their representation, e.g., by compiling them.

On the other hand, environments open the issue of space leaks since some of their bindings may become obsolete but can only be recycled when the environment itself it recycled. In functional programming, "flat" closures [13] (or again "display" closures [26]) are used instead: closures whose environment is restricted to the free variables of the term in the closure, which can be computed at compile time. The ς-calculus, however, is too dynamic in general for free variables to be computable at compile time: they need to be computed at run time. One could thus consider another possibility: to represent environments as a lightweight dictionary where each variable only occurs once.

4 Coherence between the ς-Calculus and the $\varsigma\rho$-Calculus

We establish the coherence between the ς-calculus and the $\varsigma\rho$-calculus by showing that their abstract machines are bisimilar (Section 4.2). To this end, we first introduce substitution functions mapping constructs from the $\varsigma\rho$-calculus to the ς-calculus (Section 4.1).

4.1 From Closures to Terms

We define by simultaneous induction three substitution functions that respectively map $\varsigma\rho$-values to ς-values, $\varsigma\rho$-terms to ς-terms, and environments of $\varsigma\rho$-values to temporary environments of ς-values and variables:

$$sub_{\mathbf{V}}([l_i = (\varsigma(x_i)t_i)[e_i]^{i\in\{1..n\}}]) = [l_i = \varsigma(x_i)sub_{\mathbf{T}}(t_i, (x_i, x_i) \cdot sub_{\mathbf{E}}(e_i))^{i\in\{1..n\}}]$$

$$\begin{aligned} sub_{\mathbf{T}}(x, e) &= lookup\,(x, e) \\ sub_{\mathbf{T}}(t.l, e) &= (sub_{\mathbf{T}}(t, e)).l \\ sub_{\mathbf{T}}(t.l \Leftarrow \varsigma(x)t', e) &= (sub_{\mathbf{T}}(t, e)).l \Leftarrow \varsigma(x)sub_{\mathbf{T}}(t', (x, x) \cdot e) \end{aligned}$$

$$\begin{aligned} sub_{\mathbf{E}}(\bullet) &= \bullet \\ sub_{\mathbf{E}}((x, v) \cdot e) &= (x, sub_{\mathbf{V}}(v)) \cdot sub_{\mathbf{E}}(e) \end{aligned}$$

Lemma 3. *For any closed term t and any environment e, $sub_{\mathbf{T}}(t, e) = t$.*

Proof. By simultaneous induction on the definition of $sub_{\mathbf{V}}$, $sub_{\mathbf{T}}$, and $sub_{\mathbf{E}}$ [30].

Let us also define a substitution function $sub_{\mathbf{C}}$ that maps $\varsigma\rho$-contexts to ς-contexts:

$$\begin{aligned} sub_{\mathbf{C}}([\,]) &= [\,] \\ sub_{\mathbf{C}}(C[[\,].l]) &= (sub_{\mathbf{C}}(C))[[\,].l] \\ sub_{\mathbf{C}}(C[[\,].l \Leftarrow (\varsigma(x)t)[e]]) &= (sub_{\mathbf{C}}(C))[[\,].l \Leftarrow \varsigma(x)sub_{\mathbf{T}}(t, (x, x) \cdot sub_{\mathbf{E}}(e))] \end{aligned}$$

4.2 A Bisimulation between the Two Abstract Machines

Definition 1. *Let $ST_{\varsigma\rho}$ denote the set of states of the abstract machine for the $\varsigma\rho$-calculus, and ST_{ς} denote the set of states of the abstract machine for the ς-calculus. The substitution relation $\simeq_S$: $ST_{\varsigma\rho} \times ST_{\varsigma}$ is defined as follows:*

$$\begin{aligned} \langle t, e, C\rangle &\simeq_S \langle sub_{\mathbf{T}}(t, e), sub_{\mathbf{C}}(C)\rangle \\ \langle C, v\rangle &\simeq_S \langle sub_{\mathbf{C}}(C), sub_{\mathbf{V}}(v)\rangle \\ v &\simeq_S sub_{\mathbf{V}}(v) \end{aligned}$$

Theorem 1. *The abstract machines from Sections 2.2 and 3.2 are weakly bisimilar with respect to $\simeq_S$.*

Proof. By co-induction on the execution of the abstract machine for the $\varsigma\rho$-calculus [30].

5 Related Work

The ς-calculus has already proved a fruitful playground. For example, Kesner and López [33] have defined a set of contraction rules for the ς-calculus based on explicit substitutions and flat closures. Due to the dynamic nature of the ς-calculus, and as already pointed out in Section 3.4, managing flat closures requires the evaluator to recompute sets of free variables dynamically during evaluation. In contrast, we opted for deep closures here. For another example, Gordon, Hankin and Lassen [29] have considered an imperative version of the

ς-calculus extended with λ-terms. They have defined a natural semantics based on explicit substitutions for their extended calculus, and proved it equivalent to substitution-based big-step and small-step semantics. In addition, they also provided a compiler to and a decompiler from a ZINC-like virtual machine [35]. Our approach is more inter-derivational and mechanical.

6 Conclusion and Issues

We have presented an abstract machine that mediates between Abadi and Cardelli's reduction semantics and natural semantics for the ς-calculus. We have then presented a version of the ς-calculus with explicit substitutions, the $\varsigma\rho$-calculus, and inter-derived a natural semantics, an abstract machine, and a reduction semantics for it. By construction, each of these three semantic artifacts is sound with respect to the two others. We have also shown that the abstract machines for the ς-calculus and for the $\varsigma\rho$-calculus are bisimilar, thereby establishing a coherence between the ς-calculus and the $\varsigma\rho$-calculus.

In the conclusion of "A Syntactic Correspondence between Context-Sensitive Calculi and Abstract Machines" [11], Biernacka and Danvy listed 16 distinct, independently published specifications of the control operator call/cc, and candidly asked whether all these artifacts define the same call/cc. It is the authors' belief that inter-deriving these artifacts using correct transformations puts one in position to answer this question.

As a side benefit, the nature of each inter-derivation makes it possible to pinpoint the specific goodness of each of the semantic artifacts. For example, a calculus in the form of a reduction semantics makes it possible to state equations to reason about programs; an abstract machine gives one some idea about the implementation requirements of a run-time system; and an interpreter in the form of a natural semantics is well suited for prototyping. We have illustrated these issues here with Abadi and Cardelli's untyped calculus of objects.

Acknowledgments: This article was written while the first author was visiting the PPS lab at the Université Paris 7 – Denis Diderot on a 'poste rouge' from the CNRS, in the winter of 2007–2008. The first author is grateful to his office mate at PPS, Paul-André Melliès, for insightful discussions and for the description of his completeness proof of the ς-calculus.

This work is partly supported by the Danish Natural Science Research Council, Grant no. 21-03-0545.

References

1. Abadi, M., Cardelli, L.: A Theory of Objects. In: Monographs in Computer Science. Springer, Heidelberg (1996)
2. Abadi, M., Cardelli, L., Curien, P.-L., Lévy, J.-J.: Explicit substitutions. Journal of Functional Programming 1(4), 375–416 (1991); A preliminary version was presented at the Seventeenth Annual ACM Symposium on Principles of Programming Languages (POPL 1990) (1990)

3. Ager, M.S.: Partial Evaluation of String Matchers & Constructions of Abstract Machines. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark (January 2006)
4. Ager, M.S., Biernacki, D., Danvy, O., Midtgaard, J.: A functional correspondence between evaluators and abstract machines. In: Miller, D. (ed.) Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP 2003), Uppsala, Sweden, August 2003, pp. 8–19. ACM Press, New York (2003)
5. Ager, M.S., Danvy, O., Midtgaard, J.: A functional correspondence between call-by-need evaluators and lazy abstract machines. Information Processing Letters 90(5), 223–232 (2004); Extended version available as the research report BRICS RS-04-3
6. Ager, M.S., Danvy, O., Midtgaard, J.: A functional correspondence between monadic evaluators and abstract machines for languages with computational effects. Theoretical Computer Science 342(1), 149–172 (2005); Extended version available as the research report BRICS RS-04-28
7. Barendregt, H.: The Lambda Calculus: Its Syntax and Semantics. Studies in Logic and the Foundation of Mathematics, vol. 103, revised edn. North-Holland, Amsterdam (1984)
8. Biernacka, M.: A Derivational Approach to the Operational Semantics of Functional Languages. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark (January 2006)
9. Biernacka, M., Biernacki, D., Danvy, O.: An operational foundation for delimited continuations in the CPS hierarchy. Logical Methods in Computer Science 1(2:5), 1–39 (2005); A preliminary version was presented at the Fourth ACM SIGPLAN Workshop on Continuations (CW 2004) (2004)
10. Biernacka, M., Danvy, O.: A concrete framework for environment machines. ACM Transactions on Computational Logic 9(1), 1–30, Article #6 (2007); Extended version available as the research report BRICS RS-06-3
11. Biernacka, M., Danvy, O.: A syntactic correspondence between context-sensitive calculi and abstract machines. Theoretical Computer Science 375(1-3), 76–108 (2007); Extended version available as the research report BRICS RS-06-18
12. Biernacki, D.: The Theory and Practice of Programming Languages with Delimited Continuations. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark (December 2005)
13. Cardelli, L.: Compiling a functional language. In: Steele Jr., G.L. (ed.) Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming, Austin, Texas, August 1984, pp. 208–217. ACM Press, New York (1984)
14. Church, A.: The Calculi of Lambda-Conversion. Princeton University Press, Princeton (1941)
15. Consel, C., Danvy, O.: Tutorial notes on partial evaluation. In: Graham, S.L. (ed.) Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages, Charleston, South Carolina, January 1993, pp. 493–501. ACM Press, New York (1993)
16. Cousineau, G., Curien, P.-L., Mauny, M.: The Categorical Abstract Machine. Science of Computer Programming 8(2), 173–202 (1987)
17. Curien, P.-L.: An abstract framework for environment machines. Theoretical Computer Science 82, 389–402 (1991)
18. Curien, P.-L., Hardin, T., Lévy, J.-J.: Confluence properties of weak and strong calculi of explicit substitutions. Journal of the ACM 43(2), 362–397 (1996)

19. Danvy, O.: Back to direct style. Science of Computer Programming 22(3), 183–195 (1994); A preliminary version was presented at the Fourth European Symposium on Programming (ESOP 1992) (1992)
20. Danvy, O.: From reduction-based to reduction-free normalization. In: Antoy, S., Toyama, Y. (eds.) Proceedings of the Fourth International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2004), Invited talk, Aachen, Germany, May 2004. Electronic Notes in Theoretical Computer Science, vol. 124(2), pp. 79–100. Elsevier Science, Amsterdam (2004)
21. Danvy, O.: An Analytical Approach to Program as Data Objects. DSc thesis, Department of Computer Science, University of Aarhus, Aarhus, Denmark (October 2006)
22. Danvy, O., Millikin, K.: Refunctionalization at work. Science of Computer Programming (in press); A preliminary version is available as the research report BRICS RS-07-7
23. Danvy, O., Millikin, K.: On the equivalence between small-step and big-step abstract machines: a simple application of lightweight fusion. Information Processing Letters 106(3), 100–109 (2008)
24. Danvy, O., Nielsen, L.R.: Defunctionalization at work. In: Søndergaard, H. (ed.) Proceedings of the Third International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2001), Firenze, Italy, September 2001, pp. 162–174. ACM Press, New York (2001); Extended version available as the research report BRICS RS-01-23
25. Danvy, O., Nielsen, L.R.: Refocusing in reduction semantics. Research Report BRICS RS-04-26, DAIMI, Department of Computer Science, University of Aarhus, 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 59(4)
26. Dybvig, R.K.: The development of Chez Scheme. In: Lawall, J.L. (ed.) Proceedings of the 2006 ACM SIGPLAN International Conference on Functional Programming (ICFP 2006), Keynote talk, Portland, Oregon, September 2006. SIGPLAN Notices, vol. 41(9), pp. 1–12. ACM Press, New York (2006)
27. Felleisen, M.: The Calculi of λ-v-CS Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages. PhD thesis, Computer Science Department, Indiana University, Bloomington, Indiana (August 1987)
28. Felleisen, M., Flatt, M.: Programming languages and lambda calculi (1989-2001) (last accessed, April 2008), unpublished lecture notes available at `http://www.ccs.neu.edu/home/matthias/3810-w02/readings.html`
29. Gordon, A.D., Hankin, P.D., Lassen, S.B.: Compilation and equivalence of imperative objects. Journal of Functional Programming 9(4), 373–426 (1999); Extended version available as the technical report BRICS RS-97-19
30. Johannsen, J.: Master's thesis, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark (forthcoming, 2008)
31. Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation. Prentice-Hall International, London (1993), `http://www.dina.kvl.dk/~sestoft/pebook/`
32. Kahn, G.: Natural semantics. In: Brandenburg, F.J., Wirsing, M., Vidal-Naquet, G. (eds.) STACS 1987. LNCS, vol. 247, pp. 22–39. Springer, Heidelberg (1987)
33. Kesner, D., López, P.E.M.: Explicit substitutions for objects and functions. Journal of Functional and Logic Programming Special issue 2 (1999); A preliminary version was presented at PLILP 1998/ALP 1998 (1998)

34. Landin, P.J.: The mechanical evaluation of expressions. The Computer Journal 6(4), 308–320 (1964)
35. Leroy, X.: The Zinc experiment: an economical implementation of the ML language. Rapport Technique 117, INRIA Rocquencourt, Le Chesnay, France (February 1990)
36. Marlow, S., Peyton Jones, S.L.: Making a fast curry: push/enter vs. eval/apply for higher-order languages. In: Fisher, K. (ed.) Proceedings of the 2004 ACM SIGPLAN International Conference on Functional Programming (ICFP 2004), Snowbird, Utah, September 2004. SIGPLAN Notices, vol. 39(9), pp. 4–15. ACM Press, New York (2004)
37. Midtgaard, J.: Transformation, Analysis, and Interpretation of Higher-Order Procedural Programs. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark (June 2007)
38. Millikin, K.: A Structured Approach to the Transformation, Normalization and Execution of Computer Programs. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark (May 2007)
39. Nielsen, L.R.: A study of defunctionalization and continuation-passing style. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, BRICS DS-01-7 (July 2001)
40. Nielson, H.R., Nielson, F.: Semantics with Applications, a formal introduction. Wiley Professional Computing. John Wiley and Sons, Chichester (1992)
41. Reynolds, J.C.: Definitional interpreters for higher-order programming languages. In: Proceedings of 25th ACM National Conference, Boston, Massachusetts, pp. 717–740 (1972); reprinted in Higher-Order and Symbolic Computation 11(4), 363–397 (1998) with a foreword [42]
42. Reynolds, J.C.: Definitional interpreters revisited. Higher-Order and Symbolic Computation 11(4), 355–361 (1998)
43. Rose, K.H.: Explicit substitution – tutorial & survey. BRICS Lecture Series LS-96-3, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark (September 1996)