HAL
archives-ouvertes.fr

Applicative Bisimulations for Delimited-Control Operators

Dariusz Biernacki, Sergueï Lenglet

▶ **To cite this version:**

Dariusz Biernacki, Sergueï Lenglet. Applicative Bisimulations for Delimited-Control Operators. Lars Birkedal. Foundations of Software Science and Computation Structures (FoSSaCS 2012), Mar 2012, Tallinn, Estonia. Springer, LNCS 7213, pp.119 - 134, 2012, <10.1007/978-3-642-28729-9_8>. <hal-01399945>

HAL Id: hal-01399945
https://hal.inria.fr/hal-01399945

Submitted on 21 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Applicative Bisimulations for Delimited-Control Operators

Dariusz Biernacki and Sergueï Lenglet *

University of Wrocław

Abstract. We develop a behavioral theory for the untyped call-by-value λ-calculus extended with the delimited-control operators shift and reset. For this calculus, we discuss the possible observable behaviors and we define an applicative bisimilarity that characterizes contextual equivalence. We then compare the applicative bisimilarity and the CPS equivalence, a relation on terms often used in studies of control operators. In the process, we illustrate how bisimilarity can be used to prove equivalence of terms with delimited-control effects.

1 Introduction

Morris-style contextual equivalence [21] is usually regarded as the most natural behavioral equivalence for functional languages based on λ-calculi. Roughly, two terms are equivalent if we can exchange one for the other in a bigger program without affecting its behavior (i.e., whether it terminates or not). The quantification over program contexts makes contextual equivalence hard to use in practice and, therefore, it is common to look for more effective characterizations of this relation. One approach is to rely on coinduction, by searching for an appropriate notion of *bisimulation*. The bisimulation has to be defined in such a way that its resulting behavioral equivalence, called *bisimilarity*, is *sound* and *complete* with respect to contextual equivalence (i.e., it is included and contains contextual equivalence, respectively).

The problem of finding a sound and complete bisimilarity in the λ-calculus has been well studied and usually leads to the definition of an *applicative* bisimilarity [1, 12, 11] (or, more recently, *environmental* bisimilarity [23]). The situation is more complex in λ-calculi extended with *control operators* for first-class continuations—so far, only a few works have been conducted on the behavioral theory of such calculi. A first step can be found for the $\lambda\mu$-calculus (a calculus that mimics abortive control operators such as *call/cc* [22]) in [3] and [9], where it is proved that the definition of contextual equivalence can be slightly simplified by quantifying over evaluation contexts only; such a result is usually called a *context lemma*. In [25], Støvring and Lassen define an *eager normal form bisimilarity* (based on the notion of Lévy-Longo tree equivalence) [15–17] which is sound for the $\lambda\mu$-calculus, and which becomes sound and complete when a

* The author is supported by the Alain Bensoussan Fellowship Programme.

notion of state is added to the $\lambda\mu$-calculus. In [19], Merro and Biasi define an applicative bisimilarity which characterizes contextual equivalence in the *CPS calculus* [26], a minimal calculus which models the control features of functional languages with imperative jumps. As for the λ-calculus extended with control only, however, no sound and complete bisimilarities have been defined.

In this article, we present a sound and complete applicative bisimilarity for a λ-calculus extended with Danvy and Filinski's static delimited-control operators *shift* and *reset* [8]. In contrast to abortive control operators, delimited-control operators allow to delimit access to the current continuation and to compose continuations. The operators shift and reset were introduced as a direct-style realization of the traditional success/failure continuation model of backtracking otherwise expressible only in continuation-passing style. The numerous theoretical and practical applications of shift and reset (see, e.g., [5] for an extensive list) include the seminal result by Filinski showing that a programming language endowed with shift and reset is monadically complete [10].

The λ-calculi with static delimited-control operators have been an active research topic from the semantics as well as type- and proof-theoretic point of view (see, e.g., [5, 4, 2]). However, to our knowledge, no work has been carried out on the behavioral theory of such λ-calculi. In order to fill this void, we present a study of the behavioral theory of an untyped, call-by-value λ-calculus extended with shift and reset [8], called $\lambda_{\mathcal{S}}$. In Section 2, we give the syntax and reduction semantics of $\lambda_{\mathcal{S}}$, and discuss the possible observable behaviors for the calculus. In Section 3, we define an applicative bisimilarity, based on a labelled transition semantics, and prove it characterizes contextual equivalence, using an adaptation of Howe's congruence proof method [12]. As a byproduct, we also prove a context lemma for $\lambda_{\mathcal{S}}$. In Section 4, we study the relationship between applicative bisimilarity and an equivalence based on translation into continuation-passing style (CPS), a relation often used in works on control operators and CPS. In the process, we show how applicative bisimilarity can be used to prove equivalence of terms. Section 5 concludes the article and gives ideas for future work. Most of the proofs missing from the article are avalaible in [7].

2 The Language $\lambda_{\mathcal{S}}$

In this section, we present the syntax, reduction semantics, and contextual equivalence of the language $\lambda_{\mathcal{S}}$ used throughout this article.

2.1 Syntax

The language $\lambda_{\mathcal{S}}$ extends the call-by-value λ-calculus with the delimited-control operators *shift* and *reset* [8]. We assume we have a set of term variables, ranged over by x and k. We use two metavariables to distinguish term variables bound with a λ-abstraction from variables bound with a shift; we believe such distinction helps to understand examples and reduction rules. The syntax of terms and

values is given by the following grammars:

$$\begin{array}{ll} \text{Terms:} & t ::= x \mid \lambda x.t \mid t\,t \mid \mathcal{S}k.t \mid \langle t \rangle \\ \text{Values:} & v ::= \lambda x.t \end{array}$$

The operator *shift* ($\mathcal{S}k.t$) is a capture operator, the extent of which is determined by the delimiter *reset* ($\langle\cdot\rangle$). A λ-abstraction $\lambda x.t$ binds x in t and a shift construct $\mathcal{S}k.t$ binds k in t; terms are equated up to α-conversion of their bound variables. The set of free variables of t is written $\mathsf{fv}(t)$; a term is *closed* if it does not contain any free variable. Because we work mostly with closed terms, we consider only λ-abstractions as values.

We distinguish several kinds of contexts, defined below, which all can be seen as terms with a hole.

$$\begin{array}{ll} \text{Pure evaluation contexts:} & E ::= \Box \mid v\,E \mid E\,t \\ \text{Evaluation contexts:} & F ::= \Box \mid v\,F \mid F\,t \mid \langle F \rangle \\ \text{Contexts:} & C ::= \Box \mid \lambda x.C \mid t\,C \mid C\,t \mid \mathcal{S}k.C \mid \langle C \rangle \end{array}$$

Regular contexts are ranged over by C. The pure evaluation contexts[1] (abbreviated as pure contexts), ranged over by E, represent delimited continuations and can be captured by the shift operator. The call-by-value evaluation contexts, ranged over by F, represent arbitrary continuations and encode the chosen reduction strategy. Following the correspondence between evaluation contexts of the reduction semantics and control stacks of the abstract machine for shift and reset, established by Biernacka et al. [5], we interpret contexts inside-out, i.e., $\Box$ stands for the empty context, $v\,E$ represents the "term with a hole" $E[v\,[\,]]$, $E\,t$ represents $E[[\,]\,t]$, $\langle F \rangle$ represents $F[\langle [\,] \rangle]$, etc. (This choice does not affect the results presented in this article in any way.) Filling a context C (respectively E, F) with a term t produces a term, written $C[t]$ (respectively $E[t]$, $F[t]$); the free variables of t can be captured in the process. A context is *closed* if it contains only closed terms.

2.2 Reduction Semantics

Let us first briefly describe the intuitive semantics of shift and reset by means of an example written in SML using Filinski's implementation of shift and reset [10].

Example 1. The following function copies a list [6] (the SML expression `shift (fn k => t)` corresponds to $\mathcal{S}k.t$ and `reset (fn () => t)` corresponds to $\langle t \rangle$):

```
fun copy xs =
  let fun visit nil = nil
        | visit (x::xs) = visit (shift (fn k => x :: (k xs)))
  in reset (fn () => visit xs) end
```

This program illustrates the main ideas of programming with shift and reset:

[1] This terminology comes from Kameyama (e.g., in [13]).

– Reset delimits continuations. Control effects are local to `copy`.
– Shift captures delimited continuations. Each, but last, recursive call to `visit` abstracts the continuation `fn v => reset (fn () => visit v)` and binds it to `k`.
– Captured continuations are statically composed. When applied in the expression `k xs`, the captured continuation becomes the current delimited continuation that is isolated from the rest of the program by a control delimiter—witness the reset expression in the captured continuation.

Formally, the call-by-value reduction semantics of $\lambda_{\mathcal{S}}$ is defined by the following rules, where $t\{v/x\}$ is the usual capture-avoiding substitution of v for x in t:

$$\begin{array}{lr} (\beta_v) & F[(\lambda x.t)\, v] \to_{\mathrm{v}} F[t\{v/x\}] \\ (\mathit{shift}) & F[\langle E[\mathcal{S}k.t]\rangle] \to_{\mathrm{v}} F[\langle t\{\lambda x.\langle E[x]\rangle/k\}\rangle] \text{ with } x \notin \mathsf{fv}(E) \\ (\mathit{reset}) & F[\langle v\rangle] \to_{\mathrm{v}} F[v] \end{array}$$

The term $(\lambda x.t)\, v$ is the usual call-by-value redex for β-reduction (rule (β_v)). The operator $\mathcal{S}k.t$ captures its surrounding context E up to the dynamically nearest enclosing reset, and substitutes $\lambda x.\langle E[x]\rangle$ for k in t (rule (*shift*)). If a reset is enclosing a value, then it has no purpose as a delimiter for a potential capture, and it can be safely removed (rule (*reset*)). All these reductions may occur within a metalevel context F. The chosen call-by-value evaluation strategy is encoded in the grammar of the evaluation contexts.

Example 2. Let $i = \lambda x.x$ and $\omega = \lambda x.x\, x$. We present the sequence of reductions initiated by $\langle((\mathcal{S}k_1.i\, (k_1\, i))\, \mathcal{S}k_2.\omega)\, (\omega\, \omega)\rangle$. The term $\mathcal{S}k_1.i\, (k_1\, i)$ is within the pure context $E = (\Box\, (\omega\, \omega))\, \mathcal{S}k_2.\omega$ (remember that we represent contexts inside-out), enclosed in a delimiter $\langle\cdot\rangle$, so E is captured according to rule (*shift*).

$$\langle((\mathcal{S}k_1.i\, (k_1\, i))\, \mathcal{S}k_2.\omega)\, (\omega\, \omega)\rangle \to_{\mathrm{v}} \langle i\, ((\lambda x.\langle(x\, \mathcal{S}k_2.\omega)\, (\omega\, \omega)\rangle)\, i)\rangle$$

The role of reset in $\lambda x.\langle E[x]\rangle$ becomes clearer after reduction of the (β_v)-redex $(\lambda x.\langle E[x]\rangle)\, i$.

$$\langle i\, ((\lambda x.\langle(x\, \mathcal{S}k_2.\omega)\, (\omega\, \omega)\rangle)\, i)\rangle \to_{\mathrm{v}} \langle i\, \langle(i\, \mathcal{S}k_2.\omega)\, (\omega\, \omega)\rangle\rangle$$

When the captured context E is reactivated, it is not *merged* with the context $i\ \Box$, but *composed* thanks to the reset enclosing E. As a result, the capture triggered by $\mathcal{S}k_2.\omega$ leaves the term i outside the first enclosing reset untouched.

$$\langle i\, \langle(i\, \mathcal{S}k_2.\omega)\, (\omega\, \omega)\rangle\rangle \to_{\mathrm{v}} \langle i\, \langle\omega\rangle\rangle$$

Because k_2 does not occur in ω, the context $i\ (\Box\ (\omega\, \omega))$ is discarded when captured by $\mathcal{S}k_2.\omega$. Finally, we remove the useless delimiter $\langle i\, \langle\omega\rangle\rangle \to_{\mathrm{v}} \langle i\, \omega\rangle$ with rule (*reset*), and we then (β_v)-reduce and remove the last delimiter $\langle i\, \omega\rangle \to_{\mathrm{v}} \langle\omega\rangle \to_{\mathrm{v}} \omega$. Note that, while the reduction strategy is call-by-value, some function arguments are not evaluated, like the non-terminating term $\omega\, \omega$ in this example.

There exist terms which are not values and which cannot be reduced any further; these are called *stuck terms*.

Definition 1. *A closed term t is stuck if t is not a value and $t \not\to_{\mathrm{v}}$.*

For example, the term $E[\mathcal{S}k.t]$ is stuck because there is no enclosing reset; the capture of E by the shift operator cannot be triggered. In fact, closed stuck terms are easy to characterize.

Lemma 1. *A closed term t is stuck iff $t = E[\mathcal{S}k.t']$ for some E, k, and t'.*

We call *redexes* (ranged over by r) the terms of the form $(\lambda x.t)\, v$, $\langle E[\mathcal{S}k.t]\rangle$, and $\langle v\rangle$. Thanks to the following unique-decomposition property, the reduction $\to_{\mathrm{v}}$ is deterministic.

Lemma 2. *For all closed terms t, either t is a value, or it is a stuck term, or there exist a unique redex r and a unique context F such that $t = F[r]$.*

Given a relation $\mathcal{R}$ on terms, we write $\mathcal{R}^*$ for the transitive and reflexive closure of $\mathcal{R}$. We define the evaluation relation of $\lambda_{\mathcal{S}}$ as follows.

Definition 2. *We write $t \Downarrow_{\mathrm{v}} t'$ if $t \to_{\mathrm{v}}^* t'$ and $t' \not\to_{\mathrm{v}}$.*

The result of the evaluation of a closed term, if it exists, is either a value or a stuck term. If a term t admits an infinite reduction sequence, we say it *diverges*, written $t \Uparrow_{\mathrm{v}}$. In the rest of the article, we use extensively $\Omega = (\lambda x.x\, x)\, (\lambda x.x\, x)$ as an example of such a term.

2.3 Contextual Equivalence

In this section, we discuss the possible definitions of a Morris-style contextual equivalence for the calculus $\lambda_{\mathcal{S}}$. As usual, the idea is to express that two terms are equivalent iff they cannot be distinguished when put in an arbitrary context. The question is then what kind of behavior we want to observe. As in the regular λ-calculus we could observe only termination (i.e., does a term reduce to a value or not), leading to the following relation.

Definition 3. *Let t_0, t_1 be closed terms. We write $t_0 \approx_c^1 t_1$ if for all closed C, $C[t_0] \Downarrow_{\mathrm{v}} v_0$ implies $C[t_1] \Downarrow_{\mathrm{v}} v_1$, and conversely for $C[t_1]$.*

This definition does not mention stuck terms; as a result, they can be equated with diverging terms. For example, let $t_0 = (\mathcal{S}k.k\ \lambda x.x)\ \Omega$, $t_1 = \Omega$, and C be a closed context. If $C[t_0] \Downarrow_{\mathrm{v}} v_0$, then we can prove that for all closed t, there exists v such that $C[t] \Downarrow_{\mathrm{v}} v$ (roughly, because t is never evaluated; see [7] for further details). In particular, we have $C[t_1] \Downarrow_{\mathrm{v}} v_1$. Hence, we have $t_0 \approx_c^1 t_1$.

A more fine-grained analysis is possible, by observing stuck terms.

Definition 4. *Let t_0, t_1 be closed terms. We write $t_0 \approx_c^2 t_1$ if for all closed C,*

- *$C[t_0] \Downarrow_{\mathrm{v}} v_0$ implies $C[t_1] \Downarrow_{\mathrm{v}} v_1$;*
- *$C[t_0] \Downarrow_{\mathrm{v}} t_0'$, where t_0' is stuck, implies $C[t_1] \Downarrow_{\mathrm{v}} t_1'$, with t_1' stuck as well;*

and conversely for $C[t_1]$.

The relation $\approx_c^2$ distinguishes the terms t_0 and t_1 defined above. We believe $\approx_c^2$ is more interesting because it gives more information on the behavior of terms; consequently, we use it as the contextual equivalence for $\lambda_{\mathcal{S}}$. Henceforth, we simply write $\approx_c$ for $\approx_c^2$.

The relation $\approx_c$, like the other equivalences on terms defined in this article, can be extended to open terms in the following way.

Definition 5. *Let* $\mathcal{R}$ *be a relation on closed terms. The* open extension *of* $\mathcal{R}$*, written* $\mathcal{R}^\circ$*, is defined on open terms as: we write* $t_0\ \mathcal{R}^\circ\ t_1$ *if for every substitution* σ *which closes* t_0 *and* t_1*,* $t_0\sigma\ \mathcal{R}\ t_1\sigma$ *holds.*

Remark 1. Contextual equivalence can be defined directly on open terms by requiring that the context C binds the free variables of the related terms. The resulting relation would be equal to ${\approx_c}^\circ$ [11].

3 Bisimilarity for $\lambda_{\mathcal{S}}$

In this section, we define an applicative bisimilarity and prove it equal to contextual equivalence.

3.1 Labelled Transition System

To define the bisimilarity for $\lambda_{\mathcal{S}}$, we propose a labelled transition system (LTS), where the possible interactions of a term with its environment are encoded in the labels. Figure 1 defines a LTS $t_0 \xrightarrow{\alpha} t_1$ with three kinds of transitions. An *internal action* $t \xrightarrow{\tau} t'$ is an evolution from t to t' without any help from the surrounding context; it corresponds to a reduction step from t to t'. The transition $v_0 \xrightarrow{v_1} t$ expresses the fact that v_0 needs to be applied to another value v_1 to evolve, reducing to t. Finally, the transition $t \xrightarrow{E} t'$ means that t is stuck, and when t is put in a context E enclosed in a reset, the capture can be triggered, the result of which being t'.

Most rules for internal actions (Fig. 1) are straightforward; the rules (β_{v}) and (reset) mimic the corresponding reduction rules, and the compositional rules (right_τ), (left_τ), and ($\langle\cdot\rangle_\tau$) allow internal actions to happen within any evaluation context. The rule ($\langle\cdot\rangle_{\mathcal{S}}$) for context capture is explained later. Rule (val) defines the only possible transition for values. Note that while both rules (β_{v}) and (val) encode β-reduction, they are quite different in nature; in the former, the term $(\lambda x.t)\ v$ can evolve by itself, without any help from the surrounding context, while the latter expresses the possibility for $\lambda x.t$ to evolve only if a value v is provided by the environment.

The rules for context capture are built following the principles of complementary semantics developed in [18]. The label of the transition $t \xrightarrow{E} t'$ contains what the environment needs to provide (a context E, but also an enclosing reset, left implicit) for the stuck term t to reduce to t'. Hence, the transition $t \xrightarrow{E} t'$

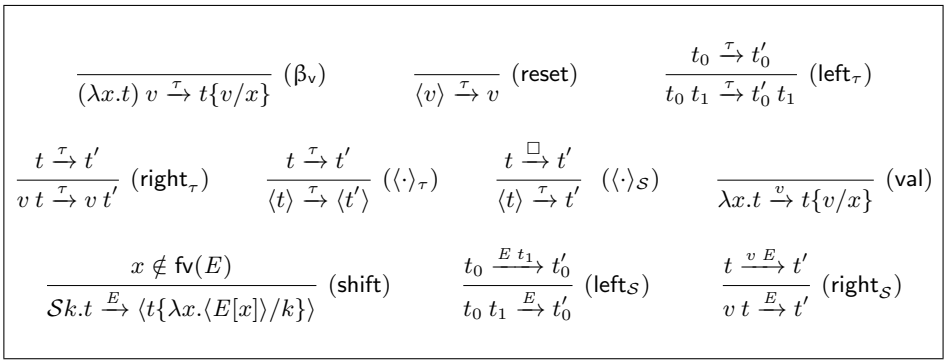

Fig. 1: Labelled Transition System

means that we have $\langle E[t]\rangle \xrightarrow{\tau} t'$ by context capture. For example, in the rule (shift), the result of the capture of E by $\mathcal{S}k.t$ is $\langle t\{\lambda x.\langle E[x]\rangle/k\}\rangle$.

In rule ($\mathsf{left}_{\mathcal{S}}$), we want to know the result of the capture of E by the term $t_0\,t_1$, assuming t_0 contains an operator shift. Under this hypothesis, the capture of E by $t_0\,t_1$ comes from the capture of $E\;t_1$ by t_0. Therefore, as premise of the rule ($\mathsf{left}_{\mathcal{S}}$), we check that t_0 is able to capture $E\;t_1$, and the result t_0' of this transition is exactly the result we want for the capture of E by $t_0\,t_1$. The rule ($\mathsf{right}_{\mathcal{S}}$) follows the same pattern. Finally, a stuck term t enclosed in a reset is able to perform an internal action (rule ($\langle\cdot\rangle_{\mathcal{S}}$)); we obtain the result t' of the transition $\langle t\rangle \xrightarrow{\tau} t'$ by letting t capture the empty context, i.e., by considering the transition $t \xrightarrow{\square} t'$.

Example 3. With the same notations as in Example 2, we illustrate how the LTS handles capture by considering the transition from $\langle (i\;\mathcal{S}k.\omega)\;(\omega\;\omega)\rangle$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{S}k.\omega \xrightarrow{i\;(\square\;(\omega\;\omega))} \langle\omega\rangle}\;(\mathsf{shift})}{i\;\mathcal{S}k.\omega \xrightarrow{\square\;(\omega\;\omega)} \langle\omega\rangle}\;(\mathsf{right}_{\mathcal{S}})}{(i\;\mathcal{S}k.\omega)\;(\omega\;\omega) \xrightarrow{\square} \langle\omega\rangle}\;(\mathsf{left}_{\mathcal{S}})}{\langle (i\;\mathcal{S}k.\omega)\;(\omega\;\omega)\rangle \xrightarrow{\tau} \langle\omega\rangle}\;(\langle\cdot\rangle_{\mathcal{S}})$$

Reading the tree from bottom to top, we see that the rules ($\langle\cdot\rangle_{\mathcal{S}}$), ($\mathsf{left}_{\mathcal{S}}$), and ($\mathsf{right}_{\mathcal{S}}$) build the captured context in the label by deconstructing the initial term. Indeed, the rule ($\langle\cdot\rangle_{\mathcal{S}}$) removes the outermost reset, and initiates the context in the label with $\square$. The rules ($\mathsf{left}_{\mathcal{S}}$) and ($\mathsf{right}_{\mathcal{S}}$) then successively remove the outermost application and store it in the context. The process continues until a shift operator is found; then we know the captured context is completed, and the rule (shift) computes the result of the capture. This result is then simply propagated from top to bottom by the other rules.

The LTS corresponds to the reduction semantics and exhibits the observable terms (values and stuck terms) of the language in the following way.

Lemma 3. *The following hold:*

- *We have* $\xrightarrow{\tau} = \rightarrow_{\mathrm{v}}$.
- *If* $t \xrightarrow{E} t'$, *then* t *is a stuck term, and* $\langle E[t] \rangle \xrightarrow{\tau} t'$.
- *If* $t \xrightarrow{v} t'$, *then* t *is a value, and* $t\ v \xrightarrow{\tau} t'$.

3.2 Applicative Bisimilarity

We now define the notion of applicative bisimilarity for $\lambda_{\mathcal{S}}$. We write $\Rightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$. We define the weak delay[2] transition $\overset{\alpha}{\Rightarrow}$ as $\Rightarrow$ if $\alpha = \tau$ and as $\Rightarrow\xrightarrow{\alpha}$ otherwise. The definition of the (weak delay) bisimilarity is then straightforward.

Definition 6. *A relation* $\mathcal{R}$ *on closed terms is an applicative simulation if* $t_0\ \mathcal{R}\ t_1$ *implies that for all* $t_0 \xrightarrow{\alpha} t'_0$, *there exists* t'_1 *such that* $t_1 \overset{\alpha}{\Rightarrow} t'_1$ *and* $t'_0\ \mathcal{R}\ t'_1$.

A relation $\mathcal{R}$ *on closed terms is an applicative bisimulation if* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are simulations. Applicative bisimilarity* $\approx$ *is the largest applicative bisimulation.*

In words, two terms are equivalent if any transition from one is matched by a weak transition with the same label from the other. As in the λ-calculus [1, 11], it is not mandatory to test the internal steps when proving that two terms are bisimilar, because of the following result.

Lemma 4. *If* $t \xrightarrow{\tau} t'$ *(respectively* $t \Downarrow_{\mathrm{v}} t'$*) then* $t \approx t'$.

Lemma 4 holds because $\{(t, t')\,,\ t \xrightarrow{\tau} t'\}$ is an applicative bisimulation. Consequently, applicative bisimulation can be defined in terms of big-step transitions as follows.

Definition 7. *A relation* $\mathcal{R}$ *on closed terms is a big-step applicative simulation if* $t_0\ \mathcal{R}\ t_1$ *implies that for all* $t_0 \overset{\alpha}{\Rightarrow} t'_0$ *with* $\alpha \neq \tau$, *there exists* t'_1 *such that* $t_1 \overset{\alpha}{\Rightarrow} t'_1$ *and* $t'_0\ \mathcal{R}\ t'_1$.

A relation $\mathcal{R}$ *on closed terms is a big-step applicative bisimulation if* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are big-step applicative simulations. Big-step applicative bisimilarity* $\dot{\approx}$ *is the largest big-step applicative bisimulation.*

Henceforth, we drop the adjective "applicative" and refer to the two kinds of relations simply as "bisimulation" and "big-step bisimulation".

Lemma 5. *We have* $\approx = \dot{\approx}$.

The proof is by showing that $\approx$ is a big step bisimulation, and that $\dot{\approx}$ is a bisimulation (using a variant of Lemma 4 involving $\dot{\approx}$). As a result, if $\mathcal{R}$ is a big-step bisimulation, then $\mathcal{R} \subseteq \dot{\approx} \subseteq \approx$. We work with both styles (small-step and big-step), depending on which one is easier to use in a given proof.

Example 4. Assuming we add lists and recursion to the calculus, we informally prove that the function `copy` defined in Example 1 is bisimilar to its effect-free variant, defined below.

[2] where internal steps are allowed before, but not after a visible action

```
fun copy2 nil = nil
  | copy2 (x::xs) = x::(copy2 xs)
```

To this end, we define the relations (where we let l range over lists, and e over their elements)

$$\mathcal{R}_1 = \{(\langle e_1 :: \langle e_2 :: \ldots \langle e_n :: \langle \texttt{visit}\ l\rangle\rangle\rangle\rangle,\ e_1 :: (e_2 :: \ldots e_n :: (\texttt{copy2}\ l)))\}$$
$$\mathcal{R}_2 = \{(\langle e_1 :: \langle e_2 :: \ldots \langle e_n :: \langle l\rangle\rangle\rangle\rangle,\ e_1 :: (e_2 :: \ldots e_n :: l))\}$$

and we prove that $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \{(l, l)\}$ is a bisimulation. First, let $t_0\ \mathcal{R}_1\ t_1$. If l is empty, then both $\texttt{visit}\ l$ and $\texttt{copy2}\ l$ reduce to the empty list, and we obtain two terms related by $\mathcal{R}_2$. Otherwise, we have $l = e_{n+1} :: l'$, $\langle \texttt{visit}\ l\rangle$ reduces to $\langle e_{n+1} :: \langle \texttt{visit}\ l'\rangle\rangle$, $\texttt{copy2}\ l$ reduces to $e_{n+1} :: (\texttt{copy2}\ l')$, and therefore t_0 and t_1 reduce to terms that are still in $\mathcal{R}_1$. Now, consider $t_0\ \mathcal{R}_2\ t_1$; the transition from t_0 removes the delimiter surrounding l, giving a term related by $\mathcal{R}_2$ to t_1 if there are still some delimiters left, or equal to t_1 if all the delimiters are removed. Finally, two identical lists are clearly bisimilar.

3.3 Soundness

To prove soundness of $\approx$ w.r.t. contextual equivalence, we show that $\approx$ is a congruence using *Howe's method*, a well-known congruence proof method initially developed for the λ-calculus [12, 11]. We briefly sketch the method and explain how we apply it to $\approx$; the complete proof can be found in [7].

The idea of the method is as follows: first, prove some basic properties of *Howe's closure* $\approx^\bullet$, a relation which contains $\approx$ and is a congruence by construction. Then, prove a simulation-like property for $\approx^\bullet$. From this result, prove that $\approx^\bullet$ and $\approx$ coincide on closed terms. Because $\approx^\bullet$ is a congruence, it shows that $\approx$ is a congruence as well. The definition of $\approx^\bullet$ relies on the notion of *compatible refinement*; given a relation $\mathcal{R}$ on open terms, the compatible refinement $\widehat{\mathcal{R}}$ relates two terms iff they have the same outermost operator and their immediate subterms are related by $\mathcal{R}$. Formally, it is inductively defined by the following rules.

$$\frac{}{x\ \widehat{\mathcal{R}}\ x} \qquad \frac{t_0\ \mathcal{R}\ t_1}{\lambda x.t_0\ \widehat{\mathcal{R}}\ \lambda x.t_1} \qquad \frac{t_0\ \mathcal{R}\ t_1 \qquad t_0'\ \mathcal{R}\ t_1'}{t_0\ t_0'\ \widehat{\mathcal{R}}\ t_1\ t_1'} \qquad \frac{t_0\ \mathcal{R}\ t_1}{\mathcal{S}k.t_0\ \widehat{\mathcal{R}}\ \mathcal{S}k.t_1} \qquad \frac{t_0\ \mathcal{R}\ t_1}{\langle t_0\rangle\ \widehat{\mathcal{R}}\ \langle t_1\rangle}$$

Howe's closure $\approx^\bullet$ is inductively defined as the smallest congruence containing $\approx^\circ$ and closed under right composition with $\approx^\circ$.

Definition 8. *Howe's closure $\approx^\bullet$ is the smallest relation satisfying:*

$$\frac{t_0 \approx^\circ t_1}{t_0 \approx^\bullet t_1} \qquad \frac{t_0 \approx^\bullet \approx^\circ t_1}{t_0 \approx^\bullet t_1} \qquad \frac{t_0\ \widehat{\approx^\bullet}\ t_1}{t_0 \approx^\bullet t_1}$$

By construction, $\approx^\bullet$ is a congruence (by the third rule of the definition), and composing on the right with $\approx^\circ$ gives some transitivity properties to $\approx^\bullet$. In particular, it helps in proving the following classical results (see [11] for the proofs).

Lemma 6 (Basic properties of $\approx^\bullet$). *The following hold:*

- *For all t_0, t_1, v_0, and v_1, $t_0 \approx^\bullet t_1$ and $v_0 \approx^\bullet v_1$ implies $t_0\{v_0/x\} \approx^\bullet t_1\{v_1/x\}$.*
- *The relation $(\approx^\bullet)^*$ is symmetric.*

The first item states that $\approx^\bullet$ is substitutive. This property helps in establishing the simulation-like property of $\approx^\bullet$ (second step of the method). Let $(\approx^\bullet)^c$ be the restriction of $\approx^\bullet$ to closed terms. We cannot prove directly that $(\approx^\bullet)^c$ is a bisimulation, so we prove a stronger result instead. We extend $\approx^\bullet$ to labels, by defining $E \approx^\bullet E'$ as the smallest congruence extending $\approx^\bullet$ with the relation $\Box \approx^\bullet \Box$, and by adding the relation $\tau \approx^\bullet \tau$.

Lemma 7 (Simulation-like property). *If $t_0\ (\approx^\bullet)^c\ t_1$ and $t_0 \xrightarrow{\alpha} t'_0$, then for all $\alpha\ (\approx^\bullet)^c\ \alpha'$, there exists t'_1 such that $t_1 \stackrel{\alpha'}{\Longrightarrow} t'_1$ and $t'_0\ (\approx^\bullet)^c\ t'_1$.*

Using Lemma 7 and the fact that $((\approx^\bullet)^c)^*$ is symmetric (by the second item of Lemma 6), we can prove that $((\approx^\bullet)^c)^*$ is a bisimulation. Therefore, we have $((\approx^\bullet)^c)^* \subseteq \approx$, and because $\approx \subseteq (\approx^\bullet)^c \subseteq ((\approx^\bullet)^c)^*$ holds by construction, we can deduce $\approx = (\approx^\bullet)^c$. Because $(\approx^\bullet)^c$ is a congruence, we have the following result.

Theorem 1. *The relation $\approx$ is a congruence.*

As a corollary, $\approx$ is sound w.r.t. contextual equivalence.

Theorem 2. *We have $\approx \subseteq \approx_c$.*

3.4 Completeness and Context Lemma

In this section, we prove that $\approx$ is complete w.r.t. $\approx_c$. To this end, we use an auxiliary relation $\dot{\approx}_c$, defined below, which refines contextual equivalence by testing terms with evaluation contexts only. While proving completeness, we also prove $\dot{\approx}_c = \approx_c$, which means that testing with evaluation contexts is as discriminative as testing with any contexts. Such a simplification result is similar to Milner's context lemma [20].

Definition 9. *Let t_0, t_1 be closed terms. We write $t_0 \dot{\approx}_c t_1$ if for all closed F,*

- *$F[t_0] \Downarrow_{\mathrm{v}} v_0$ implies $F[t_1] \Downarrow_{\mathrm{v}} v_1$;*
- *$F[t_0] \Downarrow_{\mathrm{v}} t'_0$, where t'_0 is stuck, implies $F[t_1] \Downarrow_{\mathrm{v}} t'_1$, with t'_1 stuck as well;*

and conversely for $F[t_1]$.

Clearly we have $\approx_c \subseteq \dot{\approx}_c$ by definition. The relation $\approx$ is complete w.r.t. $\dot{\approx}_c$.

Theorem 3. *We have $\dot{\approx}_c \subseteq \approx$.*

The proof of Theorem 3 is the same as in λ-calculus [11]; we prove that $\dot{\approx}_c$ is a big-step bisimulation, using Lemmas 3, 4, and Theorem 2. The complete proof can be found in [7]. We can now prove that all the relations defined so far coincide.

Theorem 4. *We have $\approx_c = \dot{\approx}_c = \approx$.*

Indeed, we have $\dot{\approx}_c \subseteq \approx$ (Theorem 3), $\approx \subseteq \approx_c$ (Theorem 2), and $\approx_c \subseteq \dot{\approx}_c$ (by definition).

$$
\begin{aligned}
\overline{x} &= \lambda k_1 k_2.k_1\ x\ k_2 \\
\overline{\lambda x.t} &= \lambda k_1 k_2.k_1\ (\lambda x.\overline{t})\ k_2 \\
\overline{t_0\ t_1} &= \lambda k_1 k_2.\overline{t_0}\ (\lambda x_0 k_2'.\overline{t_1}\ (\lambda x_1 k_2''.x_0\ x_1\ k_1\ k_2'')\ k_2')\ k_2 \\
\overline{\langle t \rangle} &= \lambda k_1 k_2.\overline{t}\ \theta\ (\lambda x.k_1\ x\ k_2) \\
\overline{\mathcal{S}k.t} &= \lambda k_1 k_2.\overline{t}\{(\lambda x_1 k_1' k_2'.k_1\ x_1\ (\lambda x_2.k_1'\ x_2\ k_2'))/k\}\ \theta\ k_2 \\
\text{with } \theta &= \lambda x k_2.k_2\ x
\end{aligned}
$$

Fig. 2: CPS translation

4 Relation to CPS Equivalence

In this section we study the relationship between our bisimilarity (and thus contextual equivalence) and an equivalence relation based on translating terms with shift and reset into continuation-passing style (CPS). Such an equivalence has been characterized in terms of direct-style equations by Kameyama and Hasegawa who developed an axiomatization of shift and reset [13]. We show that all but one of their axioms are validated by the bisimilarity of this article, which also provides several examples of use of the bisimilarity. We also pinpoint where the two relations differ.

4.1 Axiomatization of Delimited Continuations

The operators shift and reset have been originally defined by a translation into continuation-passing style [8] that we present in Fig. 2. Translated terms expect two continuations: the delimited continuation representing the rest of the computation up to the dynamically nearest enclosing delimiter and the meta-continuation representing the rest of the computation beyond this delimiter.

It is natural to relate any other theory of shift and reset to their definitional CPS translation. For example, the reduction rules $t \to_{\mathrm{v}} t'$ given in Section 2.2 are sound w.r.t. the CPS because CPS translating t and t' yields $\beta\eta$-convertible terms in the λ-calculus. More generally, the CPS translation for shift and reset induces the following notion of equivalence on terms:

Definition 10. *Terms t and t' are CPS equivalent if their CPS translations are $\beta\eta$-convertible.*

In order to relate the bisimilarity of this article and the CPS equivalence, we use Kameyama and Hasegawa's axioms [13], which characterize the CPS equivalence in a sound and complete way: two terms are CPS equivalent iff one can derive their equality using the equations of Fig. 3. Kameyama and Hasegawa's axioms relate not only closed, but arbitrary terms and they assume variables as values.

4.2 Kameyama and Hasegawa's Axioms through Bisimilarity

We show that closed terms related by all the axioms except for $\mathcal{S}$ `elim` are bisimilar. In the following, we write $\mathcal{I}$ for the bisimulation $\{(t, t)\}$.

$$
\begin{array}{rlrl}
(\lambda x.t)\, v = t\{v/x\} & \beta_{\mathtt{v}} & (\lambda x.E[x])\, t = E[t] \text{ if } x \notin \mathsf{fv}(E) & \beta_\Omega \\
\langle E[\mathcal{S}k.t]\rangle = \langle t\{\lambda x.\langle E[x]\rangle/k\}\rangle & \langle\cdot\rangle\ \mathcal{S} & \langle(\lambda x.t_0)\,\langle t_1\rangle\rangle = (\lambda x.\langle t_0\rangle)\,\langle t_1\rangle & \langle\cdot\rangle\ \mathtt{lift} \\
\langle v\rangle = v & \langle\cdot\rangle\ \mathtt{val} & \mathcal{S}k.\langle t\rangle = \mathcal{S}k.t & \mathcal{S}\ \langle\cdot\rangle \\
\lambda x.v\ x = v \text{ if } x \notin \mathsf{fv}(v) & \eta_{\mathtt{v}} & \mathcal{S}k.k\ t = t \text{ if } k \notin \mathsf{fv}(t) & \mathcal{S}\ \mathtt{elim}
\end{array}
$$

Fig. 3: Axiomatization of $\lambda_{\mathcal{S}}$

Proposition 1. *We have* $(\lambda x.t)\, v \approx t\{v/x\}$, $\langle E[\mathcal{S}k.t]\rangle \approx \langle t\{\lambda x.\langle E[x]\rangle/k\}\rangle$, *and* $\langle v\rangle \approx v$.

Proof. These are direct consequences of the fact that $\xrightarrow{\tau} \subseteq \approx$ (Lemma 4). □

Proposition 2. *If* $x \notin \mathsf{fv}(v)$, *then* $\lambda x.v\ x \approx v$.

Proof. We prove that $\mathcal{R} = \{(\lambda x.(\lambda y.t)\ x, \lambda y.t), x \notin \mathsf{fv}(t)\} \cup \approx$ is a bisimulation. To this end, we have to check that $\lambda x.(\lambda y.t)x \xrightarrow{v_0} (\lambda y.t)v_0$ is matched by $\lambda y.t \xrightarrow{v_0} t\{v_0/y\}$, i.e., that $(\lambda y.t)\ v_0\ \mathcal{R}\ t\{v_0/y\}$ holds for all v_0. We have $(\lambda y.t)\ v_0 \xrightarrow{\tau} t\{v_0/y\}$, and because $\xrightarrow{\tau} \subseteq \approx \subseteq \mathcal{R}$, we have the required result. □

Proposition 3. *We have* $\mathcal{S}k.\langle t\rangle \approx \mathcal{S}k.t$.

Proof. Let $\mathcal{R} = \{(\langle\langle t\rangle\rangle, \langle t\rangle)\}$. We prove that $\{(\mathcal{S}k.\langle t\rangle, \mathcal{S}k.t)\} \cup \mathcal{R} \cup \mathcal{I}$ is a big-step bisimulation. The transition $\mathcal{S}k.\langle t\rangle \xrightarrow{E} \langle\langle t\{\lambda x.\langle E[x]\rangle/k\}\rangle\rangle$ is matched by $\mathcal{S}k.t \xrightarrow{E} \langle t\{\lambda x.\langle E[x]\rangle/k\}\rangle$, and conversely. Let $\langle\langle t\rangle\rangle\ \mathcal{R}\ \langle t\rangle$. It is straightforward to check that $\langle\langle t\rangle\rangle \overset{v_0}{\Rightarrow} v$ iff $t \overset{v_0}{\Rightarrow} v$ iff $\langle t\rangle \overset{v_0}{\Rightarrow} v$. Therefore, any $\overset{v}{\Rightarrow}$ transition from $\langle\langle t\rangle\rangle$ is matched by $\langle t\rangle$, and conversely. If $\langle t\rangle \xrightarrow{\tau} t'$, then t' is a value or $t' = \langle t''\rangle$ for some t''; consequently, neither $\langle t\rangle$ nor $\langle\langle t\rangle\rangle$ can perform a $\xrightarrow{E}$ transition. □

Proposition 4. *We have* $\langle(\lambda x.t_0)\,\langle t_1\rangle\rangle \approx (\lambda x.\langle t_0\rangle)\,\langle t_1\rangle$.

Proof. We prove that $\{(\langle(\lambda x.t_0)\,\langle t_1\rangle\rangle, (\lambda x.\langle t_0\rangle)\,\langle t_1\rangle)\} \cup \mathcal{I}$ is a big-step bisimulation. A transition $\langle(\lambda x.t_0)\,\langle t_1\rangle\rangle \overset{\alpha}{\Rightarrow} t'$ (with $\alpha \neq \tau$) is possible only if $\langle t_1\rangle$ evaluates to some value v. In this case, we have $\langle(\lambda x.t_0)\,\langle t_1\rangle\rangle \overset{\tau}{\Rightarrow} \langle(\lambda x.t_0)\,v\rangle \xrightarrow{\tau} \langle t_0\{v/x\}\rangle$ and $(\lambda x.\langle t_0\rangle)\langle t_1\rangle \overset{\tau}{\Rightarrow} \langle t_0\{v/x\}\rangle$. From this, it is easy to see that $\langle(\lambda x.t_0)\,\langle t_1\rangle\rangle \overset{\alpha}{\Rightarrow} t'$ (with $\alpha \neq \tau$) implies $(\lambda x.\langle t_0\rangle)\,\langle t_1\rangle \overset{\alpha}{\Rightarrow} t'$, and conversely. □

Proposition 5. *If* $x \notin \mathsf{fv}(E)$, *then* $(\lambda x.E[x])\, t \approx E[t]$.

Proof (Sketch). The complete proof, quite technical, can be found in [7]. Let E_0 be such that $\mathsf{fv}(E_0) = \emptyset$. Given two families of contexts $(E_1^i)_i$, $(E_2^i)_i$, we write $\sigma_i^{E_0}$ (resp. $\sigma_i^{\lambda x.E_0[x]}$) the substitution mapping k_i to $\lambda y.\langle E_1^i[E_0[E_2^i[y]]]\rangle$ (resp. $\lambda y.\langle E_1^i[(\lambda x.E_0[x])\ E_2^i[y]]\rangle$). We define

$$
\begin{aligned}
\mathcal{R}_1 = \{(F[(\lambda x.E_0[x])\ t]\sigma_0^{\lambda x.E_0[x]} \dots \sigma_n^{\lambda x.E_0[x]}, F[E_0[t]]\sigma_0^{E_0} \dots \sigma_n^{E_0}),\\
\mathsf{fv}(t, F) \subseteq \{k_0 \dots k_n\}\}\\
\mathcal{R}_2 = \{(t\sigma_0^{\lambda x.E_0[x]} \dots \sigma_n^{\lambda x.E_0[x]}, t\sigma_0^{E_0} \dots \sigma_n^{E_0}), \mathsf{fv}(t) \subseteq \{k_0 \dots k_n\}\}
\end{aligned}
$$

and we prove that $\mathcal{R}_1 \cup \mathcal{R}_2$ is a bisimulation. The relation $\mathcal{R}_1$ contains the terms related by Proposition 5. The transitions from terms in $\mathcal{R}_1$ give terms in $\mathcal{R}_1$, except if a capture happens in t; in this case, we obtain terms in $\mathcal{R}_2$. Similarly, most transitions from terms in $\mathcal{R}_2$ give terms in $\mathcal{R}_2$, except if a term $\lambda y.\langle E_1^i[E_0[E_2^i[y]]]\rangle$ (resp. $\lambda y.\langle E_1^i[(\lambda x.E_0[x])\ E_2^i[y]]\rangle$) is applied to a value (i.e., if $t = F[k_i\ v]$). In this case, the β-reduction generates terms in $\mathcal{R}_1$. □

4.3 Bisimilarity and CPS Equivalence

In Section 4.2, we have considered all the axioms of Fig. 3, except $\mathcal{S}$ `elim`. The terms $\mathcal{S}k.k\,t$ (with $k \notin \mathsf{fv}(t)$) and t are not bisimilar in general, as we can see in the following result.

Proposition 6. *We have $\mathcal{S}k.k\,v \not\approx v$.*

The $\xrightarrow{E}$ transition from $\mathcal{S}k.k\,v$ cannot be matched by v. In terms of contextual equivalence, it is not possible to equate a stuck term and a value (it is also forbidden by the relation $\approx_c^1$ of Section 2.3). The CPS equivalence cannot distinguish between stuck terms and values, because the CPS translation turns all $\lambda_{\mathcal{S}}$ terms into λ-calculus terms of the form $\lambda k_1 k_2.t$, where k_1 is the continuation up to the first enclosing reset, and k_2 is the continuation beyond this reset. Therefore, the CPS translation (and CPS equivalence) assumes that there is always an enclosing reset, while contextual equivalence does not. To be in accordance with CPS, the contextual equivalence should be changed, so that it tests terms only in contexts with an outermost delimiter. We conjecture that the CPS equivalence is included in such a modified contextual equivalence. Note that stuck terms can no longer be observed in such modified relation, because a term within a reset cannot become stuck (see the proof of Proposition 3). Therefore, the bisimilarity of this article is too discriminative w.r.t. to this modified equivalence, and a new complete bisimilarity has to be found.

Conversely, there exist bisimilar terms that are not CPS equivalent:

Proposition 7. *1. We have $\Omega \approx \Omega\,\Omega$, but Ω and $\Omega\,\Omega$ are not CPS equivalent.*
2. Let $\Theta = \theta\,\theta$, where $\theta = \lambda xy.y\ (\lambda z.x\ x\ y\ z)$, and $\Delta = \lambda x.\delta_x\ \delta_x$, where $\delta_x = \lambda y.x\ (\lambda z.y\ y\ z)$. We have $\Theta \approx \Delta$, but Θ and Δ are not CPS equivalent.

Contextual equivalence puts all diverging terms in one equivalence class, while CPS equivalence is more discriminating. Furthermore, as is usual with equational theories for λ-calculi, CPS equivalence is not strong enough to equate Turing's and Curry's (call-by-value) fixed point combinators.

5 Conclusion

In this article, we propose a first study of the behavioral theory of a λ-calculus with delimited-control operators. We discuss various definitions of contextual equivalence, and we define an LTS-based applicative bisimilarity which is sound and complete w.r.t. the chosen contextual equivalence. Finally, we point out some differences between bisimilarity and CPS equivalence. We believe this work can be pursued in the following directions.

Up-to techniques. Up-to techniques [24, 14, 23] have been introduced to simplify the use of bisimulation in proofs of program equivalences. The idea is to prove terms equivalences using relations that are usually not bisimulations, but are included in bisimulations. The validity of some applicative bisimilarities up-to context remains an open problem in the λ-calculus [14]; nevertheless, we want to see if some up-to techniques can be applied to the bisimulations of this article.

Other forms of bisimilarity. Applicative bisimilarity is simpler to prove than contextual equivalence, but its definition still involves some quantification over values and pure contexts in labels. *Normal bisimilarities* are easier to use because their definitions do not feature such quantification. Lassen has developed a notion of normal bisimilarity, sound in various λ-calculi [15–17], and also complete in the $\lambda\mu$-calculus with state [25]. It would be interesting to see if this equivalence can be defined in a calculus with delimited control, and if it is complete in this setting. Another kind of equivalence worth exploring is *environmental bisimilarity* [23].

Other calculi with control. Defining an applicative bisimilarity for the call-by-name variant of $\lambda_{\mathcal{S}}$ and for the hierarchy of delimited-control operators [8] should be straightforward. We plan to investigate applicative bisimilarities for a typed $\lambda_{\mathcal{S}}$ as well [4]. The problem seems more complex in calculi with abortive operators, such as call/cc. Because there is no delimiter for capture, these languages are not compositional (i.e., $t \to_{\mathrm{v}} t'$ does not imply $E[t] \to_{\mathrm{v}} E[t']$), which makes the definition of a compositional LTS more difficult.

Acknowledgments: We thank Małgorzata Biernacka, Daniel Hirschkoff, Damien Pous, and the anonymous referees for many helpful comments on the presentation of this work.

References

1. S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1993.
2. Z. M. Ariola, H. Herbelin, and A. Sabry. A type-theoretic foundation of delimited continuations. *Higher-Order and Symbolic Computation*, 20(4):403–429, 2007.
3. G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In L. Brim, J. Gruska, and J. Zlatuska, editors, *MFCS'98*, volume 1450 of *LNCS*, pages 336–345, Brno, Czech Republic, Aug. 1998. Springer.
4. M. Biernacka and D. Biernacki. Context-based proofs of termination for typed delimited-control operators. In F. J. López-Fraguas, editor, *PPDP'09*, Coimbra, Portugal, Sept. 2009. ACM Press.
5. M. Biernacka, D. Biernacki, and O. Danvy. An operational foundation for delimited continuations in the CPS hierarchy. *Logical Methods in Computer Science*, 1(2:5):1–39, Nov. 2005.
6. D. Biernacki, O. Danvy, and K. Millikin. A dynamic continuation-passing style for dynamic delimited continuations. Technical Report BRICS RS-05-16, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, May 2005.

7. D. Biernacki and S. Lenglet. Applicative bisimulations for delimited-control operators, Jan. 2012. Available at `http://arxiv.org/abs/1201.0874`.
8. O. Danvy and A. Filinski. Abstracting control. In M. Wand, editor, *LFP'90*, pages 151–160, Nice, France, June 1990. ACM Press.
9. R. David and W. Py. $\lambda\mu$-calculus and Böhm's theorem. *Journal of Symbolic Logic*, 66(1):407–413, 2001.
10. A. Filinski. Representing monads. In H.-J. Boehm, editor, *POPL'94*, pages 446–457, Portland, Oregon, Jan. 1994. ACM Press.
11. A. D. Gordon. Bisimilarity as a theory of functional programming. *Theoretical Computer Science*, 228(1-2):5–47, 1999.
12. D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
13. Y. Kameyama and M. Hasegawa. A sound and complete axiomatization of delimited continuations. In O. Shivers, editor, *ICFP'03*, SIGPLAN Notices, Vol. 38, No. 9, pages 177–188, Uppsala, Sweden, Aug. 2003. ACM Press.
14. S. B. Lassen. Relational reasoning about contexts. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*. Cambridge University Press, 1998. 91-135.
15. S. B. Lassen. Eager normal form bisimulation. In P. Panangaden, editor, *LICS'05*, pages 345–354, Chicago, IL, June 2005. IEEE Computer Society Press.
16. S. B. Lassen. Normal form simulation for McCarthy's amb. In M. Escardó, A. Jung, and M. Mislove, editors, *MFPS'05*, volume 155 of *ENTCS*, pages 445–465, Birmingham, UK, May 2005. Elsevier Science Publishers.
17. S. B. Lassen and P. B. Levy. Typed normal form bisimulation for parametric polymorphism. In F. Pfenning, editor, *LICS'08*, pages 341–352, Pittsburgh, Pennsylvania, June 2008. IEEE Computer Society Press.
18. S. Lenglet, A. Schmitt, and J.-B. Stefani. Howe's method for calculi with passivation. In M. Bravetti and G. Zavattaro, editors, *CONCUR'09*, number 5710 in LNCS, pages 448–462, Bologna, Italy, July 2009. Springer.
19. M. Merro and C. Biasi. On the observational theory of the CPS-calculus: (extended abstract). *ENTCS*, 158:307–330, 2006.
20. R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.
21. J. H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, Massachusets Institute of Technology, 1968.
22. M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *LPAR'92*, number 624 in LNAI, pages 190–201, St. Petersburg, Russia, July 1992. Springer-Verlag.
23. D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In J. Marcinkowski, editor, *LICS'07*, pages 293–302, Wroclaw, Poland, July 2007. IEEE Computer Society Press.
24. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
25. K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In M. Felleisen, editor, *POPL'07*, SIGPLAN Notices, Vol. 42, No. 1, pages 161–172, New York, NY, USA, Jan. 2007. ACM Press.
26. H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1997. ECS-LFCS-97-376.