Effectful applicative similarity for call-by-name lambda calculi

Ugo Dal Lago[1], Francesco Gavazzo[2], and Ryo Tanaka[3]

[1] Università di Bologna & INRIA Sophia Antipolis
ugo.dallago@unibo.it
[2] Università di Bologna & INRIA Sophia Antipolis
francesco.gavazzo2@unibo.it
[3] The University of Tokyo
anaphalis.alpicola@gmail.com

Abstract. We introduce a notion of applicative similarity in which not terms but *monadic values* arising from the evaluation of effectful terms, can be compared. We prove this notion to be fully abstract whenever terms are evaluated in call-by-name order. This is the first full-abstraction result for such a generic, coinductive methodology for program equivalence.

1 Introduction

Program equivalence is a crucial notion in the theory of programming languages, and its study is at the heart of programming language semantics. When higher-order functions are available, programs have a non-trivial interactive behaviour, and giving a *satisfactory* and at the same time *handy* definition of program equivalence becomes complicated. The problem has been approached, in many different ways, e.g. by denotational semantics or by contextual equivalence. These two approaches have their drawbacks, the first one relying on the existence of a denotational model, the latter quantifying over all contexts, thus making the task of proving programs equivalent quite hard. Handier methodologies for proving programs equivalent have been introduced along the years based on logical relations and applicative bisimilarity. Logical relations were originally devised for typed, normalising languages, but later generalised to more expressive formalisms, e.g., through step-indexing [2] and biorthogonality [3]. Starting from Abramsky's pioneering work on applicative bisimilarity [1], coinduction has also been proved to be a useful methodology for program equivalence, and has been applied to a variety of calculi and language features [17].

The just described scenario also holds when the underlying calculus is not pure, but effectful. There have been many attempts to study effectful λ-calculi [19,16] by way of denotational semantics [9,8], logical relations [4], and applicative bisimilarity [14,7,5]. But while the denotational and logical relation semantics of effectful calculi have been studied in the abstract [11,13], the same cannot be said about applicative bisimilarity and related coinductive techniques. There is

a growing body of literature on applicative bisimilarity for calculi with, e.g., nondeterministic [14], and probabilistic effects [7], but each notion of an effect has been studied independently, often getting different results. Distinct proofs of congruence for applicative bisimilarity, even if done through a common methodology, namely the so-called Howe's method [12], do not at all have the same difficulty in each of the cases cited above.

The observations above naturally lead to some questions. Is there any way to factor out the common part of the congruence proof for applicative bisimilarity in effectful calculi? Where do the limits on the correctness of applicative bisimilarity lie, in presence of effects?

This paper is part of a longstanding research effort directed to giving answers to the questions above. The first two authors, together with Paul Blain Levy, have recently introduced a general notion of applicative bisimilarity for lambda-calculi based on monads and relators [6]. This provides, under mild conditions, a sound methodology for checking contextual equivalence of programs. The central idea is to take simulations as relations on terms and to use a *relator* to lift a (candidate) simulation to a relation on the monadic images of values. There is however little hope to prove a generic full-abstraction result in such a setting, although for certain notions of an effect, full abstraction is already known to hold [1,5].

In this paper, we study a different notion of simulation, which puts in relation not terms but their semantics. This way, interaction between terms and the environment can be modeled by an ordinary, deterministic, transition system and, with minimal side conditions, similarity can be proved to be not only a *sound*, but also to coincide with the contextual preorder. This, however, only holds when terms are call-by-*name* evaluated.

We first of all introduce a computational λ-calculus in which general algebraic effects can be represented, and give a monadic call-by-name operational semantics for it, showing how the latter coincides with the expected one in many distinct concrete examples. We then show how applicative bisimilarity can be defined for any instance of such a monadic λ-calculus, and that in all cases it is also fully-abstract. Along the lines, we also prove a CIU Theorem.

2 Mathematical Preliminaries

In this section, we recall some basic definitions and results on complete partial orders, categories, and monads. Due to space constraints, there is no hope to be comprehensive. The reader can refer to [6] for details and further references.

We assume basic familiarity with domain theory. In particular, we do not define the notions of directed complete partial order (dcpo for short), and of pointed dcpo (dcppo for short). Similarly, we do not give the notions of monotone, continuous, and strict functions, which are standard. Following [19,20], we consider operations producing effects coming from a signature. Semantically, dealing with operation symbols requires the introduction of appropriate algebraic structures interpreting such operation symbols as suitable functions. Combining

the algebraic and the order theoretic structures just described, leads to consider algebras carrying a domain structure (dcppo, in this paper), and with all function symbols interpreted as continuous functions. The formal notion capturing all these desiderata is the one of a continuous Σ-algebra [10]. Recall that a signature $\Sigma = (\mathcal{F}, \alpha)$ consists of a set $\mathcal{F}$ of operation symbols and a map $\alpha : \mathcal{F} \to \mathbb{N}$, assigning to each operation symbol a (finite) arity.

Definition 1. *Given a signature Σ, a continuous Σ-algebra is a dcppo $(D, \sqsubseteq, \bot)$ such that for any function symbol σ in Σ there is an associated continuous function $\sigma_D : D^{\alpha(\sigma)} \to D$.*

Example 1. Let X be a set: the following are examples of dcppo (see [6] for further examples).

1. The flat lifting X_{undef} of X, defined as $X + \{\mathsf{undef}\}$, ordered as follows: $x \sqsubseteq y$ iff $x = \mathsf{undef}$ or $x = y$.
2. The set $(X+E)_{\mathsf{undef}}$ (think to E as a set of exceptions), ordered as in previous example. We can consider the signature $\Sigma = \{\mathtt{raise}(e) \mid e \in E\}$, where each operation symbol $\mathtt{raise}(e)$ is interpreted as the constant $\mathsf{in}_l(\mathsf{in}_r(e))$.
3. The powerset $\mathcal{P}X$, ordered by inclusion. The least upper bound of a chain of sets is their union, whereas the bottom is the empty set. We can consider the signature $\Sigma = \{\oplus\}$ containing a binary operation symbol for nondeterministic choice. The latter can be interpreted as (binary) union, which is indeed continuous.
4. The set of discrete subdistributions $\mathcal{D}X = \{\mu : X \to [0,1] \mid \sum_{x \in X} \mu(x) \leq 1\}$ over X, ordered pointwise: $\mu \sqsubseteq \nu$ iff $\forall x \in X.\ \mu(x) \leq \nu(x)$. The dcppo structure is pointwise induced by the one of $[0,1]$ with the natural ordering. The least element is the always zero distribution $x \mapsto 0$ (note that the latter is a subdistribution, and not a distribution). We can consider the signature $\Sigma = \{\oplus_p \mid p \in [0,1]\}$ with a family of probabilistic choice operations indexed by real numbers in $[0,1]$. We can interpret $\oplus_p$ as the binary operation $(x, y) \mapsto p \cdot x + (1-p) \cdot y$, which is indeed continuous.

Since we are interested in effectul calculi, we rely on monads $\langle T, \eta, \mu \rangle$ [15] (on the category of sets and functions) to model the kind of effects we are interested in. For a function $f : X \to TY$ we denote by $f^\dagger$ its Kleisli lifting $\mu_Y \circ Tf : TX \to TY$. We are interested in monads carrying a continuous Σ-algebra structure. That is, for a fixed signature Σ, we require a monad T to come with a map associating to any set X an order $\sqsubseteq_X$ on TX and an element $\bot_X \in TX$ such that $(TX, \sqsubseteq_X, \bot_X)$ is a continuous Σ-algebra wrt the order $\sqsubseteq_X$. We also require Kleisli lifting to satisfy specific continuity conditions[4]. The reader can consult [6] for details.

Example 2. It is easy to see that all the constructions in Example 1 carry a monad structure.

[4] For all sets X, Y we require (notice that TY being a dcppo, so is $X \to TY$) both $\sup \mathcal{F}^\dagger = \sup_{f \in \mathcal{F}} f^\dagger$ and $f^\dagger(\sup \mathcal{T}) = \sup_{t \in \mathcal{T}} f^\dagger(t)$ to hold, where $\mathcal{F} \subseteq X \to TY$ and $\mathcal{T} \subseteq TX$ are directed.

Finally, following [19] we require monads to properly interact with algebraic operations.

Definition 2. *Let T be a monad carrying a continuous Σ-algebra structure. We say that T is algebraic if for any function $f : X \to TY$, $f^\dagger$ is Σ-algebra homomorphism. That is,*

$$f^\dagger(\sigma_X(t_1, \ldots, t_n)) = \sigma_Y(f^\dagger(t_1), \ldots, f^\dagger(t_n)).$$

We treat $\bot_X$ as an algebraic constant, so that the above definition implies $f^\dagger(\bot_X) = \bot_Y$, for any $f : X \to TY$.

3 Syntax and Operational Semantics

In this section we introduce a computational λ-calculus and give it *call-by-name* monadic operational semantics following [19]. We fix a signature Σ and a monad $\langle T, \eta, \mu\rangle$ satisfying the conditions of previous section.

Definition 3. *For a fixed signature Σ the sets Λ° and $\mathcal{V}^\circ$ of* terms *and* values *are defined as follows:*

$$M, N \ ::= \ V \ \big| \ MN \ \big| \ \sigma(M, \ldots, M) \qquad\qquad V \ ::= \ x \ \big| \ \lambda x.M.$$

We let letters $M, N \ldots$ and $V, W, \ldots$ to range over terms and values, respectively. We denote by $\Lambda(\bar{x})$ the set of terms with free variables from $\bar{x}$ and write Λ for the set $\Lambda(\emptyset)$. Similar conventions apply to the set of values.

We give operational semantics to our calculus as in [19] (which we refer to for details). An indexed function symbol σ_i consists of a function symbol $\sigma \in \Sigma$ together with an index $i \in \{0, \ldots, \alpha(\sigma) - 1\}$. *Evaluation contexts* are defined by the simple grammar: $E \ ::= \ [\cdot] \ \big| \ EM$. Small-step semantics is defined by means of a binary relation $\to$ on closed terms and of a ternary relation $\stackrel{\cdot}{\to}$ on closed terms and indexed function symbols, according to the rules in Figure 1.

$$(\lambda x.M)N \to M[N/x] \qquad \frac{M \to N}{E[M] \to E[N]} \qquad \frac{M \stackrel{\sigma_i}{\to} N}{E[M] \stackrel{\sigma_i}{\to} E[N]}$$
$$\sigma(M_0, \ldots, M_{n-1}) \stackrel{\sigma_i}{\to} M_i$$

Fig. 1. Small-step Semantics.

We now give a monadic multi-step operational semantics to the above calculus. Multi-step operational semantics relates a closed term M to a set of elements in $T\mathcal{V}$, representing finite approximations of its evaluation. We refer to elements in $T\mathcal{V}$ as *monadic values*.

Definition 4. *Define the relation* $\Rightarrow$ *between closed terms and monadic values by the rules in Figure 2, where* $\bot$ *is the bottom element of* $T\mathcal{V}$ *and* $\sigma_{\mathcal{V}}$ *is the interpretation of* σ *in* $T\mathcal{V}$.

$$\frac{}{M \Rightarrow \bot} \qquad \frac{}{V \Rightarrow \eta(V)} \qquad \frac{M \to N \qquad N \Rightarrow t}{M \Rightarrow t} \qquad \frac{M \xrightarrow{\sigma_i} N_i \qquad N_i \Rightarrow t_i}{M \Rightarrow \sigma_{\mathcal{V}}(t_1, \ldots, t_n)}$$

Fig. 2. Big-step Semantics.

It is easy to prove that the set $\{t \in T\mathcal{V} \mid M \Rightarrow t\}$ of finite approximations of (the execution of) M is directed, meaning that it has a least upper bound.

Definition 5. *Define the* evaluation function $[\![\cdot]\!] : \Lambda_0 \to T(\mathcal{V})$ *by* $[\![M]\!] = \sup_{M \Rightarrow t} t$.

4 Observational Equivalence

In this section we give a general notion of observation on top of which we define several behavioural preorders. We rely on the basic assumption that we can only observe side effects.

Intuitively, an observation is a function that takes an element in $T\mathcal{V}$ representing the computation of a term and returns what we can observe about it. Since we assume that the observable part of a computation consists only of the side effects that could occur during such computation, such observable part is uniquely determined by the semantics of the operations considered.

Since we are working with an untyped calculus, it is customary *not* to observe values[5].

Definition 6. *Let* $1 = \{*\}$ *be the one element set and* $!_X : X \to 1$ *be the unique function mapping each element in* X *to* $*$. *Define the observation function* $Obs : T\mathcal{V} \to T1$ *as* $T(!_{\mathcal{V}})$. *Observations trivially extends to terms by defining* $Obs(M)$ *as* $Obs([\![M]\!])$.

Note that by very definition of monad we have $T(!_{\mathcal{V}}) = (\eta_1 \circ !)^\dagger$. As a consequence, since T is algebraic an easy calculation shows that we have the following equalities:

$$\begin{aligned} Obs(\bot) &= \bot_{T(1)}; \\ Obs(\eta_{\mathcal{V}}(V)) &= \eta_1(*); \\ Obs(\sigma_{\mathcal{V}}(t_1, \ldots, t_n) &= \sigma_1(Obs(t_1), \ldots, Obs(t_n)). \end{aligned}$$

[5] One is usually more concerned with different notions of convergence, such as may convergence or the probability of convergence.

Remark 1. Notice that the function Obs is uniquely determined by the effect signature. That is because of our choice of not distinguishing between values. Nevertheless, it is easy to see that our definition of Obs can be extended to any function $f : \mathcal{V} \to X$, where the set X represents a set of observables and f encodes a ground observation on values. In fact, we would have $Obs(V) = (\eta \circ f)(V)$.

We can now give a general notion of contextual preorder.

Definition 7. *Define* contexts *by the following two grammars*

$$C ::= \cdot \mid MC \mid CM \mid \lambda x.C$$

The contextual proerder $\leq^{ctx}$ *is defined as follows:*

$$M \leq^{ctx} N \iff \forall C.\ Obs([\![C[M]]\!]) \sqsubseteq Obs([\![C[N]]\!])$$

Example 3. Referring to Example 1:

1. For pure λ-calculus we consider the monad $TX = X_{\mathsf{undef}}$ thus obtaining $Obs : \mathcal{V}_{\mathsf{undef}} \to 1_{\mathsf{undef}}$ satisfying: $Obs(\eta(V)) = *$ and $Obs(\bot) = \mathsf{undef}$. We thus have $Obs(M) = \mathsf{undef}$ iff $M \uparrow$. As a consequence, $M \leq^{ctx} N$ iff for any context C, $C[M] \uparrow \implies C[N] \uparrow$, which is the usual notion of contextual preorder.
2. For the nondeterministic calculus of Example 1 we have $\mathcal{P}(1) = \{\emptyset, *\}$ so that: $Obs(\bot) = \emptyset$, $Obs(V) = \{*\}$ and $Obs(M \oplus N) = Obs(M) \cup Obs(N)$. We thus have $Obs(M) = \{*\}$ iff M may converge. Again, we have recovered the usual notion of contextual preorder.
3. For the probabilistic calculus of Example 1 we first observe that $\mathcal{D}(1) \cong [0, 1]$. It is then easy to see that we obtain $Obs(\bot) = 0$, $Obs(V) = 1$ and $Obs(M \oplus_p N) = p \cdot Obs(M) + (1 - p) \cdot Obs(N)$. We have $Obs(M) = Pr(M \Downarrow)$, meaning that $M \leq^{ctx} N$ holds iff for any context C, $Pr(C[M] \Downarrow) \leq Pr(C[N] \Downarrow)$ holds.

5 Applicative Similarity

In this section we define the notion of applicative simulation and prove full abstraction of applicative similarity wrt the CIU preorder. First of all, we lift the notion of application from terms to monadic values.

Definition 8. *For $t \in T\mathcal{V}$ and $M \in \Lambda$ define the application $t \circ M \in T\mathcal{V}$ of M to t as*[6]*:* $((\lambda x.L) \mapsto [\![L[M/x]]\!])^\dagger(t)$.

It is easy to see that we have:

$$\bot \circ M = \bot;$$
$$\eta(\lambda x.N) \circ M = [\![N[M/x]]\!];$$
$$\sigma_{\mathcal{V}}(t_1, \ldots, t_n) \circ M = \sigma_{\mathcal{V}}(t_1 \circ M, \ldots, t_n \circ M);$$

and that for terms M, N we have $[\![MN]\!] = [\![M]\!] \circ N$.

[6] Note that with a similar strategy it is possible to extend application to $T\mathcal{V}^\circ$.

Definition 9. *A relation $\mathcal{R} \subseteq T\mathcal{V} \times T\mathcal{V}$ is an applicative simulation if whenever $(t, u) \in \mathcal{R}$ we have*
1. $Obs(t) \sqsubseteq Obs(u)$;
2. For any term M, $(t \circ M, u \circ M) \in \mathcal{R}$.
Applicative similarity $\lesssim$ is defined as the largest applicative simulation. We extend applicative similarity to terms as follows (we overload the notation): $M \lesssim N$ iff $[\![M]\!] \lesssim [\![N]\!]$.

Our main result is a *full abstraction* theorem stating that applicative similarity and the contextual preorder coincide. The proof of such result relies on a CIU Theorem [18].

Definition 10. *The CIU preorder $\leq^{ciu}$ is defined by*

$$M \leq^{ciu} N \iff \forall E.\ Obs([\![E[M]]\!]) \sqsubseteq Obs([\![E[N]]\!])$$

where E ranges over the set of evaluation contexts.

Theorem 1 (CIU Equivalence). *The following holds: $\leq^{ciu} = \leq^{ctx}$.*

We postpone the proof of Theorem 1 to the next section. We spend the rest of this section showing that applicative similarity is fully abstract with respect to the CIU preorder.

Proposition 1 (Soundness). *The following holds: $\lesssim \subseteq \leq^{ciu}$.*

Proof. Clearly $M \lesssim N$ implies $ML \lesssim NL$, for any term L. It is then straightforward to prove that for any evaluation context E, $M \lesssim N$ implies $E[M] \lesssim E[N]$, meaning, in particular, $Obs(E[M]) \sqsubseteq Obs(E[N])$. □

To prove that $\lesssim$ is fully abstract wrt $\leq^{ciu}$ we have to prove $\leq^{ciu} \subseteq \lesssim$. To do so we define a grammar for monadic values as follows:

$$\mathfrak{t} ::= t \mid \mathfrak{t} \bullet M.$$

We refer to terms generated by the above grammar as *monadic terms*. We can give semantics to monadic terms as monadic values via the mapping:

$$\wr t \wr = t;$$
$$\wr \mathfrak{t} \bullet M \wr = \wr \mathfrak{t} \wr \circ M.$$

Monadic term contexts (mt-contexts, for shorts) are defined by:

$$C ::= \cdot \mid C \bullet M.$$

Therefore a mt-context is an expression of the form $\cdot \bullet M_1 \bullet \cdots \bullet M_n$ (where we assume $\bullet$ to be left associative). The lifting $\preccurlyeq^{ciu}$ of the CIU preorder to monadic terms is then defined by

$$\mathfrak{t} \preccurlyeq^{ciu} \mathfrak{u} \iff \forall C.\ Obs(\wr C[\mathfrak{t}] \wr) \sqsubseteq Obs(\wr C[\mathfrak{u}] \wr).$$

It is easy to see that there is a bijective correspondence between evaluation contexts and mt-contexts from which it follows the equation $\leq^{ciu} = \preccurlyeq^{ciu}$.

Proposition 2. *The following holds:* $\preccurlyeq^{ciu} \subseteq \lesssim$.

Proof. We prove that $\preccurlyeq^{ciu}$ is an applicative simulation. Suppose $t \preccurlyeq^{ciu} u$. Obviously $Obs(t) \sqsubseteq Obs(u)$ since $\wr t \int = t$ (and similarly for u). Let now M be a term. We have to prove that $t \circ M \preccurlyeq^{ciu} u \circ M$ holds. Let $C = \cdot \bullet M_1 \bullet \cdots \bullet M_n$ be a mt-context. Then we have to prove: $Obs(\wr C[t \circ M] \int) = Obs(\wr C[u \circ M] \int)$. That immediatly follows from $t \preccurlyeq^{ciu} u$ once one observes that $\wr(\cdot \bullet M_1 \bullet \cdots \bullet M_n)[t \circ M] \int = \wr(\cdot \bullet M \bullet M_1 \bullet \cdots \bullet M_n)[t] \int$ holds. □

Corollary 1. *The following holds:* $\leq^{ciu} \subseteq \lesssim$.

Together with Proposition 1, the above corollary gives a proof of the following:

Proposition 3. *Applicative similarity is fully abstract wrt the CIU preorder. That is:* $\lesssim = \leq^{ciu}$.

What remains to be done in order to prove that applicative similarity is fully abstract wrt to the contextual preorder is to prove Theorem 1.

6 A CIU Theorem

In this section we prove Theorem 1. The proof closely follows [18], and the reader can consult the latter for further details.

We use the notion of *frame stack*.

Definition 11. *The set of frame stacks is defined by the following grammar:*

$$s ::= \mathtt{nil} \mid [\cdot]M :: s.$$

A *configuration* is a pair of the form (s, M), where s is a stack and M a term. To a stack $s = [\cdot]N_1 :: \cdots :: [\cdot]N_n$, we can associate the evaluation context $E_s = \cdot N_1 \cdots N_n$ (where to $\mathtt{nil}$ we associate the empty context). Vice versa, we can associate to any evaluation context E a stack s_E such that $E = E_{s_E}$.

We now give operational semantics to configurations. Small step semantics is defined via a relation $\mapsto$ between configurations, whereas multi-step semantics is defined via a relation $\Mapsto$ between configurations and monadic values. Both these relations are described in Figure 3.

Notice that the set $\{t \in T\mathcal{V} \mid (s, M) \Mapsto t\}$ is directed. As a consequence, we can define the (operational) semantics $(\!|(s, M)|\!)$ of a configuration (s, M) as

$$(\!|(s, M)|\!) = \sup_{(s,M) \Mapsto t} t.$$

Proposition 4. *For any term M, evaluation context E and stack s, we have:*

$$[\![E[M]]\!] = (\!|(s_E, M)|\!);$$
$$(\!|(s, M)|\!) = [\![E_s(M)]\!].$$

In particular, we have $(\!|(\mathtt{nil}, M)|\!) = [\![M]\!]$.

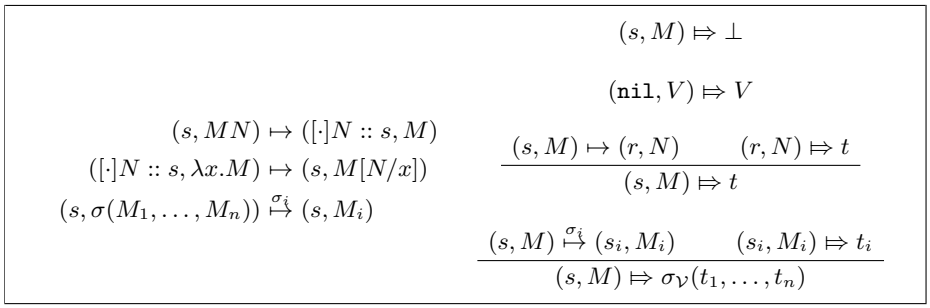

Fig. 3. Stack Semantics.

We can now build up a proof of Theorem 1. The proof is based on an adaptation oh Howe's technique for pure λ-calculus and thus we only sketch its main part. The reader can consult [18] for details. First of all, in light of previous proposition let us characterise the CIU preorder in terms of stacks and configurations.

Definition 12. *Define the CIU order $\preceq^{ciu}$ on terms:*

$$M \preceq^{ciu} N \iff \forall s.\ Obs(\Downarrow\!(s, M)\!\Downarrow) \sqsubseteq Obs(\Downarrow\!(s, N)\!\Downarrow).$$

Proposition 5. *The following hold: $\leq^{ciu} = \preceq^{ciu}$.*

Dealing with Howe's technique, it is convenient to work with generalisations of relations on closed terms called λ-term relations.

Definition 13. *An open relation over terms is a set $\mathcal{R}$ of triples $(\bar{x}, M, N)$ where $M, N \in \Lambda(\bar{x})$. A closed λ-term relation is closed if $\mathcal{R} \subseteq \Lambda \times \Lambda$.*

We will use infix notation and write $\bar{x} \vdash M\ \mathcal{R}\ N$ to indicate that $(\bar{x}, M, N) \in \mathcal{R}$. Finally, we will use the notations $\emptyset \vdash M\ \mathcal{R}\ N$ and $M\ \mathcal{R}\ N$ interchangeably. There is a canonical way to extend a closed relation to an open one.

Definition 14. *Define the open extension operator mapping a closed relation over terms $\mathcal{R}$ to the open relation $\mathcal{R}^\circ$ over terms as follows: $(\bar{x}, M, N) \in \mathcal{R}^\circ$ iff $M, N \in \Lambda(\bar{x})$, and for all $\bar{L}$, $M[\bar{L}/\bar{x}]\ \mathcal{R}\ N[\bar{L}/\bar{x}]$ holds.*

The notions of reflexivity, symmetry and transitivity straightforwardly extend to open λ-term relation (see e.g. [18]).

We say that a relation is *compatible* if it is closed under the term constructors of the language. The idea of Howe's technique is to extend a relation $\mathcal{R}$ to a relation $\mathcal{R}^H$ that is compatible by construction. As a consequence, if $M\mathcal{R}^H N$ holds, then $C[M]\mathcal{R}^H C[N]$ holds as well. That means that in order to prove that a relation $\mathcal{R}$ is compatible it is sufficient to prove that $\mathcal{R} = \mathcal{R}^H$ holds. Proving the inclusion $\mathcal{R} \subseteq \mathcal{R}^H$ is often immediate (provided that $\mathcal{R}$ satisfies minimal properties, e.g. reflexivity and transitivity in our case), whereas the other inclusion requires more effort.

Definition 15. *Given a λ-term relation $\mathcal{R}$ the Howe's lifting $\mathcal{R}^H$ of $\mathcal{R}$ is inductively defined by the rules in Figure 4.*

$$\frac{\bar{x} \vdash x\mathcal{R}M}{\bar{x} \vdash x\mathcal{R}^H M}\ \mathsf{How1} \qquad \frac{\bar{x} \cup \{x\} \vdash M\mathcal{R}^H L \qquad \bar{x} \vdash \lambda x.L\mathcal{R}N \qquad x \notin \bar{x}}{\bar{x} \vdash \lambda x.M\mathcal{R}^H N}\ \mathsf{How2}$$

$$\frac{\bar{x} \vdash M\mathcal{R}^H P \qquad \bar{x} \vdash L\mathcal{R}^H \qquad \bar{x} \vdash P\mathcal{R}N}{\bar{x} \vdash ML\mathcal{R}^H N}\ \mathsf{How3}$$

$$\frac{\bar{x} \vdash M_i\mathcal{R}^H N_i \qquad \bar{x} \vdash \sigma(N_1, \ldots, N_n)\mathcal{R}L}{\bar{x} \vdash \sigma(M_1, \ldots, M_n)\mathcal{R}^H L}\ \mathsf{How4}$$

Fig. 4. Howe's Lifting.

The following lemma states some useful properties of Howe's lifting of preorder relations. The proof is standard and can be found in, e.g., [7].

Lemma 1. *Let $\mathcal{R}$ be a preorder. The following hold*
1. *$\mathcal{R} \circ \mathcal{R}^H \subseteq \mathcal{R}^H$.*
2. *$\mathcal{R}^H$ is reflexive.*
3. *For any context C, $\bar{x} \vdash M\mathcal{R}^H N \implies \bar{x} \vdash C[M]\mathcal{R}^H C[N]$.*
4. *$\mathcal{R} \subseteq \mathcal{R}^H$.*

It is easy to see that $\preceq^{ciu}$ is a preorder[7] and thus $(\preceq^{ciu})^H$ is a compatible preorder containing $\preceq^{ciu}$. We extend Howe's construction to frame stacks as described in Figure 5.

$$\frac{}{\mathtt{nil}\mathcal{R}^H\mathtt{nil}}\ \mathsf{HowStk1} \qquad \frac{\emptyset \vdash M\mathcal{R}^H N \qquad s\mathcal{R}^H r}{[\cdot]M :: s\mathcal{R}^H[\cdot]N :: r}\ \mathsf{HowStk2}$$

Fig. 5. Howe's Lifting for Frame Stacks.

The main techincal part of the Howe's method is the so-called Key Lemma, which basically states that the Howe's extension of a relation is contained in the relation itself.

Lemma 2 (Key Lemma). *For all closed stacks s, r and closed terms M, N, if $s(\preceq^{ciu})^H r$, $M(\preceq^{ciu})^H N$, then $Obs(\langle\!|(s, M)|\!\rangle) \sqsubseteq Obs(\langle\!|(r, N)|\!\rangle)$.*

Proof (Sketch.). Since Obs is continuous and $\langle\!|(s, M)|\!\rangle) = \sup_{(s,M)\mapsto t} t$ it is sufficient to prove that for every monadic value t, if $(s, M) \mapsto t$ the $Obs(t) \sqsubseteq \langle\!|(r, N)|\!\rangle$. The latter statement can be proved by induction on the derivation of $(s, M) \mapsto t$ as in [18]. □

[7] We are actually working with $(\preceq^{ciu})^\circ$.

Corollary 2. *The following holds:* $(\preceq^{ciu})^H = \preceq^{ciu}$.

Proof. We already know that $\preceq^{ciu} \subseteq (\preceq^{ciu})^H$ holds. To prove the other inclusion we have to prove $M(\preceq^{ciu})^H N \implies M \preceq^{ciu} N$, for all terms M, N. Suppose $M(\preceq^{ciu})^H N$. Then, for any stack s, we have $s(\preceq^{ciu})^H s$. By previous lemma we have $Obs(\!|(s, M)|\!) \sqsubseteq Obs(\!|(s, N)|\!)$, meaning that $M \preceq^{ciu} N$. □

We can finally prove Theorem 1.

Theorem 1 (CIU Equivalence). *The following holds:* $\leq^{ctx} = \leq^{ciu}$.

Proof. It is immediate to see that $\leq^{ctx} \subseteq \leq^{ciu}$. For the other inclusion, suppose $M \leq^{ciu} N$ and let C be a context. We prove that $C[M] \leq^{ciu} C[N]$ (this will give the thesis simply aplying the definition of $\leq^{ciu}$ with the empty context). Since $\leq^{ciu} = \preceq^{ciu} = (\preceq^{ciu})^H$, we have $M \preceq^{ciu} N$ and thus $C[M] \preceq^{ciu} C[N]$. It follows $C[M] \leq^{ciu} C[N]$. □

7 Conclusions

We have shown how a notion of applicative similarity on call-by-name effectful lambda-calculi can be defined and proved fully-abstract. This is very much in line with logical relations as introduced in [13], but simpler and applicable to untyped λ-calculi. The underlying transition system has monadic values as states, and is essentially deterministic. This is indeed the reason the framework is only applicable to call-by-name (and not to call-by-value) calculi, in contrast with [6]. In fact, in the probabilistic call-by-value case our notion of applicative similarity on monadic values is unsound for the contextual preorder, as witnessed by the terms $\lambda x.(x \oplus_p \Omega)$, $(\lambda x.x) \oplus_p (\lambda x.\Omega)$, and the context $(\lambda x.x(x(\lambda y.y)))[\cdot]$.

References

1. Samson Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.
2. Andrew W. Appel and David A. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5):657–683, 2001.
3. Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann. Relational semantics for effect-based program transformations: higher-order store. In *Proc. of PPDP 2009*, pages 301–312, 2009.
4. Ales Bizjak and Lars Birkedal. Step-indexed logical relations for probability. In *Proc. of FOSSACS 2015*, pages 279–294, 2015.
5. Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In *Proc. of ESOP 2014*, volume 8410 of *LNCS*, pages 209–228. Springer, 2014.
6. Ugo Dal Lago, Francesco Gavazzo, and Paul Levy. Effectful applicative bisimilarity: Monads, relators, and Howe's method (long version). Available at `https://arxiv.org/abs/1704.04647`, 2017.

7. Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL 2014*, pages 297–308, 2014.
8. Vincent Danos and Russell Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic*, 3(3):359–382, 2002.
9. Ugo de'Liguoro and Adolfo Piperno. Non deterministic extensions of untyped lambda-calculus. *Inf. Comput.*, 122(2):149–177, 1995.
10. Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977.
11. Jean Goubault-Larrecq, Slawomir Lasota, and David Nowak. Logical relations for monadic types. *Mathematical Structures in Computer Science*, 18(6):1169–1217, 2008.
12. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.
13. Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for algebraic effects. In *Proc. of LICS 2010*, pages 209–218. IEEE Computer Society, 2010.
14. Søren B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Dept. of Computer Science, University of Aarhus, May 1998.
15. Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.
16. Eugenio Moggi. Computational lambda-calculus and monads. In *Proc. of (LICS 1989*, pages 14–23. IEEE Computer Society, 1989.
17. A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.
18. Andrew M. Pitts. Howe's method for higher-order languages. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, chapter 5, pages 197–232. Cambridge University Press, November 2011.
19. Gordon D. Plotkin and John Power. Adequacy for algebraic effects. In *Proc. of FOSSACS 2001*, pages 1–24, 2001.
20. Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.