On completeness of logical relations for monadic types [⋆]

Sławomir Lasota[1] [⋆⋆]   David Nowak[2]   Yu Zhang[3] [⋆ ⋆ ⋆]

[1] Institute of Informatics, Warsaw University, Warszawa, Poland
sl@mimuw.edu.pl
[2] Department of Information Science, The University of Tokyo, Japan
nowak@yl.is.s.u-tokyo.ac.jp
[3] LSV, CNRS & INRIA Futurs & ENS Cachan, Cachan, France
zhang@lsv.ens-cachan.fr

Abstract. Interesting properties of programs can be expressed using contextual equivalence. The latter is difficult to prove directly, hence (pre-)logical relations are often used as a tool to prove it. Whereas pre-logical relations are complete at all types, logical relations are only complete up to first-order types. We propose a notion of contextual equivalence for Moggi's computational lambda calculus, and define pre-logical and logical relations for this calculus. Monads introduce new difficulties: in particular the usual proofs of completeness up to first-order types do not go through. We prove completeness up to first order for several of Moggi's monads. In the case of the non-determinism monad we obtain, as a corollary, completeness of strong bisimulation w.r.t. contextual equivalence in lambda calculus with monadic non-determinism.

1   Introduction

Interesting properties of programs can be expressed using the notion of contextual equivalence. While contextual equivalence is difficult to prove directly because of the universal quantification over contexts, logical relations and pre-logical relations [4] are powerful tools that allow us to deduce contextual equivalence in typed lambda-calculi. They are easily proved sound w.r.t. contextual equivalence using the so-called Basic Lemma. Whereas pre-logical relations are complete at all types, logical relations are only complete up to first-order types.

On the other hand, Moggi's computational lambda-calculus [7] has proved useful to define various notions of computations on top of the λ-calculus: partial computation, exception, non-determinism, state transformer and continuation in particular. Moggi's insight is based on categorical semantics: while categorical models of the λ-calculus are cartesian closed categories (CCCs), the computational lambda-calculus requires CCCs with a strong monad. A natural notion of logical relations able to deal with the monadic types was proposed in [2].

In this paper, we propose a notion of contextual equivalence for Moggi's computational lambda calculus, and we define pre-logical relations and logical relations for this calculus. Monads introduce new difficulties: in particular the usual proofs of completeness up to first order do not go through. We prove completeness up to first order for several of Moggi's monads. For some monads we require the presence of specific constants. In the case of the non-determinism monad, we need to restrict ourselves to a subset of first-order types. As a corollary, we prove that strong bisimulation is complete w.r.t. contextual equivalence in a lambda calculus with monadic non-determinism.

Section 2 is devoted to preliminaries. We prove in Section 3 that pre-logical relations are complete at all types in the computational lambda-calculus. In Section 4, we look at completeness of logical relations for various Moggi's monads.

2   Preliminaries

2.1   Logical relations and contextual equivalence for λ-calculus

We consider the simply typed λ-calculus, where a type is either a base types b (booleans, integers, etc.), or a function type $\tau \to \tau'$. Terms are:

$$t ::= x \mid c \mid \lambda x \cdot t \mid tt'$$

where c ranges over a set of constants and x over a set of variables. Notations and typing rules are as usual.

We consider set theoretical semantics of the simply types λ-calculus. A Γ-*environment* ρ is a map such that, for every $x : \tau$ in Γ, $\rho(x)$ is an element of $[\![\tau]\!]$. Let t be a term such that $\Gamma \vdash t : \tau$ is derivable. The denotation of t, w.r.t. a Γ-environment ρ, is given as usual by an element $[\![t]\!]\rho$ of $[\![\tau]\!]$. We write $[\![t]\!]$ instead of $[\![t]\!]\rho$ when ρ is irrelevant.

Contextual equivalence. Let **Obs** be a subset of base types, called *observation types*. An observation type is any base type with decidable equality, e.g. booleans, integers, etc. A *context* $\mathbb{C}$ is a term such that $x : \tau \vdash \mathbb{C} : o$ is derivable, where τ is a type and o is an observation type. Two elements a_1 and a_2 of $[\![\tau]\!]$, are *contextually equivalent* (written as $a_1 \approx_\tau a_2$), if and only if for any context $\mathbb{C}$ such that $x : \tau \vdash \mathbb{C} : o$ ($o \in$ **Obs**) is derivable, $[\![\mathbb{C}]\!][x := a_1] = [\![\mathbb{C}]\!][x := a_2]$. We say that two closed terms t_1 and t_2 of the same type τ are *contextually equivalent* (written as $t_1 \approx_\tau t_2$) whenever $[\![t_1]\!] \approx_\tau [\![t_2]\!]$.

Logical relations. Essentially, a (binary) *logical relation* [5] is a family $(\mathcal{R}_\tau)_{\tau \text{ type}}$ of relations, one for each type τ, on $[\![\tau]\!]$ such that related functions map related arguments to related results. More formally, it is a family $(\mathcal{R}_\tau)_{\tau \text{ type}}$ of relations such that for every $f_1, f_2 \in [\![\tau \to \tau']\!]$,

$$f_1 \, \mathcal{R}_{\tau \to \tau'} \, f_2 \quad \Longleftrightarrow \quad \forall a_1, a_2 \in [\![\tau]\!] \cdot a_1 \, \mathcal{R}_\tau \, a_2 \implies f_1(a_1) \, \mathcal{R}_{\tau'} \, f_2(a_2)$$

There is no constraint on relations at base types. In the simply typed λ-calculus, once the relations at base types are fixed, the above condition forces $(\mathcal{R}_\tau)_{\tau \text{ type}}$ to be uniquely determined by induction on types. It is certainly possible to have other complex types, e.g., products. In general, relations of these complex types should be also uniquely determined by relations of their type components. For instance, pairs are related when their elements are pairwise related.

In this setting, what is crucial for deducing contextual equivalence is the so-called *Basic Lemma*. It states that if $\Gamma \vdash t : \tau$ is derivable, and ρ_1, ρ_2 are two related Γ-environments, then $[\![t]\!]\rho_1 \, \mathcal{R}_\tau \, [\![t]\!]\rho_2$. Here two Γ-environments ρ_1, ρ_2 are *related* by the logical relation, if and only if $\rho_1(x) \, \mathcal{R}_\tau \, \rho_2(x)$ for every $x : \tau$ in Γ. Basic Lemma implies that, for any logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ which is a partial equality on observation types, logically related values are necessarily contextually equivalent, i.e., $\mathcal{R}_\tau \subseteq \approx_\tau$ for any type τ.

Take any two values $a_1, a_2 \in [\![\tau]\!]$ such that $a_1 \, \mathcal{R}_\tau \, a_2$. By Basic Lemma, for every context $\mathbb{C}$ such that $x : \tau \vdash \mathbb{C} : o$ is derivable ($o \in$ **Obs**), $[\![\mathbb{C}]\!][x := a_1] \, \mathcal{R}_o \, [\![\mathbb{C}]\!][x := a_2]$, i.e., $[\![\mathbb{C}]\!][x := [\![a_1]\!]] = [\![\mathbb{C}]\!][x := [\![a_2]\!]]$ since $\mathcal{R}_o$ is partial equality.

Completeness is another important aspect of logical relations. We say a logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ is *complete* if every contextually equivalent values are related by this logical relation, i.e., $\approx_\tau \subseteq \mathcal{R}_\tau$ for every type τ. However, completeness for logical relations is hard to achieve. Usually we are only able to prove completeness for types up to first order (the order of types is defined inductively: $\mathbf{ord}(b) = 0$ for any base type b; $\mathbf{ord}(\tau \to \tau') = \max(\mathbf{ord}(\tau) + 1, \mathbf{ord}(\tau'))$ for function types).

Proposition 1 *There exists a logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ with partial equality on observation types, such that if $\vdash t_1 : \tau$ and $\vdash t_2 : \tau$ are derivable, for any type τ up to first order, $t_1 \approx_\tau t_2 \implies [\![t_1]\!] \, \mathcal{R}_\tau \, [\![t_2]\!]$.*

Definability is largely involved in the proof of completeness. We say that a value $a \in [\![\tau]\!]$ is *definable* if and only if there exists a closed term t such that $\vdash t : \tau$ is derivable and $a = [\![t]\!]$. We define a relation $\sim_\tau$ by $a_1 \sim_\tau a_2$ (for $a_1, a_2 \in [\![\tau]\!]$) if and only if a_1, a_2 are definable and $a_1 \approx_\tau a_2$. Now let $(\mathcal{R}_\tau)_{\tau \text{ type}}$ be the logical relation induced by $\mathcal{R}_b = \sim_b$ at all base types b and we can show that it is complete for types up to first order, by induction over types (see Appendix A for the details).

Pre-logical relations. In order to get completeness at all types, we appeal to the notion of *pre-logical relations* [4]. A *pre-logical relation* is any family $(\mathcal{R}_\tau)_{\tau \text{ type}}$ of relations (between two values of $[\![\tau]\!]$ in our case) such that:

(i) for every $f_1, f_2 \in [\![\tau \to \tau']\!]$, if $f_1 \, \mathcal{R}_{\tau \to \tau'} \, f_2$ and $a_1 \, \mathcal{R}_\tau \, a_2$ then $f_1(a_1) \, \mathcal{R}_{\tau'} \, f_2(a_2)$;
(ii) $\mathsf{K} \, \mathcal{R}_{\tau \to \tau' \to \tau} \, \mathsf{K}$;
(iii) $\mathsf{S} \, \mathcal{R}_{(\tau \to \tau' \to \tau'') \to (\tau \to \tau') \to \tau \to \tau''} \, \mathsf{S}$;
(iv) and for every constant $c : \tau$, $[\![c]\!] \, \mathcal{R}_\tau \, [\![c]\!]$.

where K is the function mapping $x \in [\![\tau]\!]$, $y \in [\![\tau']\!]$ to x, and S is the function mapping $x \in [\![\tau \to \tau' \to \tau'']\!]$, $y \in [\![\tau \to \tau']\!]$, $z \in [\![\tau]\!]$ to $x(z)(y(z))$; Compared to logical relations, the main difference of pre-logical relations is that relations $\mathcal{R}_{\tau \to \tau'}$ for functions are no more determined uniquely by relations $\mathcal{R}_\tau$ and $\mathcal{R}_{\tau'}$.

Basic Lemma for pre-logical relations [4, Lemma 4.1] is stronger than for logical relations: pre-logical relations are *exactly* those families of relations such that Basic Lemma holds.

With respect to contextual equivalence, pre-logical relations are not just sound, but also complete *at all types*. More precisely, there exists a pre-logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ which is partial equality on observation types, and for every closed terms t_1, t_2 of type τ, if $t_1 \approx_\tau t_2$ then $[\![t_1]\!] \, \mathcal{R}_\tau \, [\![t_2]\!]$. An instance of such a pre-logical relation is the relation $\sim_\tau$ we defined above. As shown in [3, Theorem 3], $\sim_\tau$ is indeed a pre-logical relation. Although the argument of [3] is in the setting of cryptographic lambda-calculi, the proof does not depend on any particular constants or types, and can be applied here without any change.

2.2 Logical relations and contextual equivalence for computational λ-calculus

Moggi defines a language called the computational λ-calculus and uses monads to model computational types, in order to give semantics to programming languages which include side effects such as exceptions, non-determinism, and so on [7]. This was done in a categorical setting. While categorical models of the simply typed λ-calculus are cartesian closed categories (CCCs), the computational λ-calculus requires CCCs with a strong monad $(T, \eta, \mu, \mathsf{t})$.

An extra unary type constructor T is introduced in the computational λ-calculus. Intuitively, a type $\mathsf{T}\tau$ is the type of computations of type τ. We call $\mathsf{T}\tau$ a *monadic type* in the sequel. Compared to the λ-calculus, the computational λ-calculus is extended with two constructs (and corresponding typing rules):

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathtt{val}(t) : \mathsf{T}\tau} \qquad\qquad \frac{\Gamma \vdash t_1 : \mathsf{T}\tau \quad \Gamma, x : \tau \vdash t_2 : \mathsf{T}\tau'}{\Gamma \vdash \mathtt{let}\ x \Leftarrow t_1\ \mathtt{in}\ t_2 : \mathsf{T}\tau'}$$

Each computational λ-term has a unique interpretation as a morphism in a CCC with a strong monad. In particular, the interpretation of terms in the computational λ-calculus must satisfy the following equations:

$$[\![\mathtt{let}\ x \Leftarrow \mathtt{val}(t_1)\ \mathtt{in}\ t_2]\!]\rho = [\![t_2[t_1/x]]\!]\rho \tag{1}$$

$$[\![\mathtt{let}\ x \Leftarrow t\ \mathtt{in}\ \mathtt{val}(x)]\!]\rho = [\![t]\!]\rho \tag{2}$$

$$[\![\mathtt{let}\ x_2 \Leftarrow (\mathtt{let}\ x_1 \Leftarrow t_1\ \mathtt{in}\ t_2)\ \mathtt{in}\ t_3]\!]\rho = [\![\mathtt{let}\ x_1 \Leftarrow t_1\ \mathtt{in}\ \mathtt{let}\ x_2 \Leftarrow t_2\ \mathtt{in}\ t_3]\!]\rho \tag{3}$$

Indeed, every term of a monadic type can be written in some canonical form (respecting these equations):

Definition 1 (Computational canonical form). *A term t of a monadic type $\mathsf{T}\tau$ in the computational λ-calculus is said to be a* computational canonical term *if it is of the form*

$$\mathtt{let}\ x_1 \Leftarrow t_1\ \mathtt{in}\cdots\mathtt{let}\ x_n \Leftarrow t_n\ \mathtt{in}\ \mathtt{val}(u) \quad (n = 0, 1, 2, \ldots)$$

where u is a term of type τ, x_1, ..., x_n are variables and every t_i ($i = 1, \ldots, n$) is a weak head normal form, i.e., $t_i = u_i w_{i1} \cdots w_{ik_i}$ and each u_i is either a variable or a constant.

Proposition 2 *For every term t of a monadic type $\mathsf{T}\tau$ in the computational λ-calculus, there exists a computational canonical term t' such that $[\![t']\!]\rho = [\![t]\!]\rho$, for every valid interpretation $[\![\_]\!]\rho$ (i.e., interpretations satisfying the equations (1-3)).*

Proof. The computational λ-calculus is strongly normalizing [1], so we consider the β-normal form of term t and prove it by induction on t. See Appendix A for details. □

Contextual equivalence. In order to define contextual equivalence for the computational λ-calculus, we have to consider contexts $\mathbb{C}$ of type $\mathsf{T}o$ (where o is an observable type), not of type o. Indeed, contexts should be allowed to do some computations. If they were of type o, they could only return values. In particular, a context $\mathbb{C}$ such that $x : \mathsf{T}\tau \vdash \mathbb{C} : o$ is derivable, meant to observe computations of type τ, cannot observe anything. This is because the typing rule for the $\mathtt{let}$ construct only allows us to use computations to build other computations, never values.

Taking this into account, we are led to the following definition:

Definition 2 (Contextual equivalence). *For any type τ, we say that two values $a_1, a_2 \in [\![\tau]\!]$ are contextually equivalent, written as $a_1 \approx_\tau a_2$, if and only if, for all observable types $o \in \mathbf{Obs}$ and contexts $\mathbb{C}$ such that $x : \tau \vdash \mathbb{C} : \mathsf{T}o$ is derivable, $[\![\mathbb{C}]\!][x := a_1] = [\![\mathbb{C}]\!][x := a_2]$.*

Two closed terms t_1 and t_2 of type τ are contextually equivalent (written as $t_1 \approx_\tau t_2$) if and only if $[\![t_1]\!] \approx_\tau [\![t_2]\!]$.

Logical and pre-logical relations. A natural extension of logical relations able to deal with monadic types was introduced in [2]. It relies on the categorical notion of subscones [6] — a uniform framework for defining logical relations. The construction consists of lifting the CCC structure and the strong monad from the categorical model to the subscone.

We shall focus in this paper on Moggi's monads [7] defined over the category $\mathcal{S}et$ of sets and functions. In this case, the subscone is the category whose objects are binary relations $(A, B, R \subseteq A \times B)$ where A and B are sets; and a morphism between two objects $(A, B, R \subseteq A \times B)$ and $(A', B', R' \subseteq A' \times B')$ is a pair of functions $(f : A \to A', g : B \to B')$ preserving relations, i.e. whenever $a\ R\ b$, then $f(a)\ R'\ g(b)$.

The lifting of the CCC structure gives rise to the standard logical relations given in Section 2.1 and the lifting of the strong monad will give rise to relations for monadic types. We write $\tilde{T}$ for the lifting of the strong monad T. Given a relation $R \subseteq A \times B$ and two computations $a \in TA$ and $b \in TB$, $(a, b) \in \tilde{T}(R)$ if and only if there exists a computation $c \in T(R)$ (i.e. c computes pairs in R) such that $a = T\pi_1(c)$ and $b = T\pi_2(c)$. The standard definition of logical relation for the simply typed λ-calculus is then extended with:

$$(c_1, c_2) \in \mathcal{R}_{\mathsf{T}\tau} \iff (c_1, c_2) \in \tilde{T}(\mathcal{R}_\tau) \tag{4}$$

The construction guarantees that Basic Lemma still holds.

Pre-logical relations for computational λ-calculus are obtained in a natural way. Namely, we extend the definition from Section 2.1 with the condition (4) above. Note that relation $\mathcal{R}_{\mathsf{T}\tau}$ on each monadic type is determined uniquely by $\mathcal{R}_\tau$, in contrast to function types.

3 Completeness of pre-logical relations

Let T be an arbitrary monad in $\mathcal{S}et$. We restrict our attention to pre-logical relations $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that, for any observation type $o \in \mathbf{Obs}$, $\mathcal{R}_{\mathsf{T}o}$ is a partial equality. Such relations are called *observational* in the rest of the paper.

Note that we require partial identity on $\mathsf{T}o$, not on o. But if we assume that denotation of $\mathtt{val}(\_)$, i.e., the unit operation η, is injective, then that $\mathcal{R}_{\mathsf{T}o}$ is a partial equality implies that $\mathcal{R}_o$ is a partial equality as well. Indeed, let $a_1 \, \mathcal{R}_o \, a_2$, and by Basic Lemma, $[\![\mathtt{val}(x)]\!][x := a_1] \, \mathcal{R}_{\mathsf{T}o} \, [\![\mathtt{val}(x)]\!][x := a_2]$, that is to say $\eta_{[\![o]\!]}(a_1) = \eta_{[\![o]\!]}(a_2)$. By injectivity of η, $a_1 = a_2$.

Theorem 1 (Soundness). *If $(\mathcal{R}_\tau)_{\tau \text{ type}}$ is an observational pre-logical relation, then $\mathcal{R}_\tau \subseteq \approx_\tau$ for every type τ.*

Proof. By Basic Lemma. $\qquad\square$

Obviously, soundness holds for observational logical relations as well.

As in Section 2.1, for every type τ, we define the binary relation $(\sim_\tau)_{\tau \text{ type}}$ by: for any $a_1, a_2 \in [\![\tau]\!]$, $a_1 \sim_\tau a_2$ if and only if a_1 and a_2 are definable and $a_1 \approx_\tau a_2$.

Theorem 2 (Completeness). *There exists an observational pre-logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that, for all terms t_1 and t_2 such that $\vdash t_1 : \tau$ and $\vdash t_2 : \tau$ are derivable, $t_1 \approx_\tau t_2 \implies [\![t_1]\!] \, \mathcal{R}_\tau \, [\![t_2]\!]$.*

Proof. Obviously, $t_1 \approx_\tau t_2 \implies [\![t_1]\!] \sim_\tau [\![t_2]\!]$, so we will prove that $(\sim_\tau)_{\tau \text{ type}}$ is a pre-logical relation. The conditions (ii) – (iv), for $(\sim_\tau)_{\tau \text{ type}}$, are clearly satisfied: $[\![t]\!] \sim_\tau [\![t]\!]$ holds for every closed term t of type τ, hence *a fortiori* for K, S and any constant c. To prove the condition (i), we use similar techniques as in Proposition 1 (see Appendix A for details). $\qquad\square$

4 Completeness of logical relations for monadic types

This section is devoted to the study on the completeness of logical relations for the computational λ-calculus. However, it seems difficult to get a general result on completeness for all monads, since specific properties of particular monads (and corresponding logical relations) are quite different. Furthermore, an important role is played by the language constants since our discussion necessarily involves contexts, and these constants vary widely as well. Hence, instead of a general approach, we shall check the completeness in some important examples, notably partial computation, exception, nondeterminism, state transformer and continuation.

We restrict ourselves to types up to *first order* in the computational lambda-calculus:

$$\tau^1 ::= b \mid \mathsf{T}\tau^1 \mid b \to \tau^1,$$

where b ranges over a set of base types. However, completeness for types up to first-order does not hold for every monad. In the case of the non-determinism monad, it holds only for a subset of first-order types.

Similarly as in Proposition 1 in Section 2.1, we investigate completeness in a strong sense. We aim at finding an observational logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that if $\vdash t_1 : \tau$ and $\vdash t_2 : \tau$ are derivable and $t_1 \approx_\tau t_2$, for any type τ up to first order, then $[\![t_1]\!] \, \mathcal{R}_\tau \, [\![t_2]\!]$. Or briefly, $\sim_\tau \subseteq \mathcal{R}_\tau$. As in Proposition 1, the logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ will be induced by $\mathcal{R}_b = \sim_b$, for any base type b. Then how to prove completeness, for an arbitrary monad T?

As usual, the proof would go by induction over τ, to show $\sim_\tau \subseteq \mathcal{R}_\tau$ for each first-order type τ. Now, cases $\tau = b$ and $\tau = b \to \tau'$ go identically as in the simply typed λ-calculus. The only difficult case is $\tau = \mathsf{T}\tau'$, i.e.,

$$\sim_\tau \, \subseteq \mathcal{R}_\tau \implies \sim_{\mathsf{T}\tau} \, \subseteq \mathcal{R}_{\mathsf{T}\tau} \tag{5}$$

We did not find any general way to show (5) for an arbitrary monad. Instead, in the following subsections we shall prove it for particular ones.

There is also another subtle point – notice that it is even not true in general that $(\mathcal{R}_\tau)_{\tau \text{ type}}$ is observational, i.e., it is not necessarily partial identity on $\mathsf{T}o$. Fortunately, this can be solved in general, under some mild assumptions on the monad T, fulfilled by all the monads investigated in the sequel. A detailed treatment is to be found in the full version of the paper.

At the heart of the difficulty of showing (5), one finds an issue of definability at monadic types. By definition, an element c of $[\![\mathsf{T}\tau]\!]$ is definable if and only if there is a close term t such that $\vdash t : \mathsf{T}\tau$ is derivable and $[\![t]\!] = c$. But this does not state anything on the connection between the definability of a computation and its corresponding "result". Intuitively, if a value of $[\![\mathsf{T}\tau]\!]$ is definable, either it corresponds to a computation which "returns a definable result" (necessarily of type τ), or there is a specific constant in the language defining this value. This argument is by no means formal and we shall make it more precise for each monad, in Propositions 3, 4, 5, 6 and 7. Interestingly, all of them can be spelled-out shortly by $\mathsf{def}_{\mathsf{T}\tau} \subseteq T\mathsf{def}_\tau$, where by $\mathsf{def}_\tau \subseteq [\![\tau]\!]$ we mean the subset of definable elements of $[\![\tau]\!]$. But even though stated easily in general, this fact needs substantially different proofs for different monads.

Before we move on to discussions of concrete monads, we define a $\mathcal{P}$-*form* for closed terms, parameterized by a predicate $\mathcal{P}$ on terms.

Definition 3. *For any predicate $\mathcal{P}$ on terms, we say that a closed term (necessarily of a computation type $\mathsf{T}\tau$ for some τ) is in $\mathcal{P}$-*form *if and only if it is of the form*

$$\texttt{let } x_1 \Leftarrow t_1 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u) \quad (n = 0, 1, \ldots),$$

where $\mathcal{P}$ is a predicate on closed terms, $t_i = u_i w_{i1} \cdots w_{ik_i}$ ($1 \leq i \leq n$), u_i is either a variable x_l ($1 \leq l \leq i - 1$) or any closed term such that $\mathcal{P}(u_i)$ holds, w_{im} ($1 \leq m \leq k_i$) is a term whose free variables must be in $\{x_1, \ldots, x_{i-1}\}$ and u is any term of type τ with free variables in $\{x_1, \ldots, x_n\}$.

For a concrete monad, we shall define a predicate **Cond** on closed terms, which is inclined to act as a condition to be satisfied by constants.

4.1 Partial computation

$$[\![\mathsf{T}\tau]\!] = [\![\tau]\!] \cup \{\bot\},$$
$$[\![\texttt{val}(t)]\!]\rho = [\![t]\!]\rho,$$
$$[\![\texttt{let } x \Leftarrow t_1 \texttt{ in } t_2]\!]\rho = \begin{cases} [\![t_2]\!]\rho[x := [\![t_1]\!]\rho] & \text{if } [\![t_1]\!]\rho \neq \bot \\ \bot & \text{if } [\![t_1]\!]\rho = \bot \end{cases},$$

where $\bot$ is a distinguished element denoting non-terminating computations. Logical relations at monadic types are given by [2]:

$$c_1 \mathcal{R}_{\mathsf{T}\tau} c_2 \iff c_1 \mathcal{R}_\tau c_2 \text{ or } c_1 = c_2 = \bot \tag{6}$$

Let **Cond** be the smallest set of closed terms such that, for any closed term t of type $\tau_1 \to \cdots \to \tau_n \to \mathsf{T}\tau$, **Cond**$(t)$ holds if and only if: for any closed terms $\vdash t_1 : \tau_1, \cdots, \vdash t_n : \tau_n$, $[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$ is either:

- equal to $\bot$, or
- definable at type τ (by some closed term t'), and, if τ is of the form $\tau_1' \to \cdots \to \tau_m' \to \mathsf{T}\tau'$, then **Cond**$(t')$ holds.

We assume that for any constant d, **Cond**(d) holds. We also assume that there is, for every τ, a constant Ω_τ of type $\mathsf{T}\tau$ such that $[\![\Omega_\tau]\!] = \bot$. Clearly **Cond**$(\Omega_\tau)$ holds.

Lemma 1. *For any closed term t (of type $\mathsf{T}\tau$) in the **Cond**-form, $[\![t]\!]$ is either $\bot$, or a definable value at type τ.*

Proof. By induction on n of the **Cond**-form (see Appendix A for details). $\qquad\square$

Proposition 3 *A value $c \in [\![\mathsf{T}\tau]\!]$ is definable if and only if, either c is definable at type τ, or $c = \bot$, i.e.:* $\mathsf{def}_{\mathsf{T}\tau}(c) \Longleftrightarrow \mathsf{def}_\tau(c)$ *or* $c = \bot$.

Proof. The "if" direction: For any value $c \in [\![\mathsf{T}\tau]\!]$, if $c = \bot$, it is obvious (Ω_τ defines it); if $c \in [\![\tau]\!]$ and $\mathsf{def}_\tau(c)$ holds, suppose c is defined by some closed term t of type τ, then c is also definable at type $\mathsf{T}\tau$ (by the term $\mathtt{val}(t)$), i.e., $\mathsf{def}_{\mathsf{T}\tau}(c)$ holds.

The "only if" direction: Suppose that there is a value $c \in [\![\mathsf{T}\tau]\!]$ which is definable by some closed term t of type $\mathsf{T}\tau$. Consider the computational canonical form of t:

$$u_t \equiv \mathtt{let}\ x_1 \Leftarrow t_1\ \mathtt{in} \cdots \mathtt{let}\ x_n \Leftarrow t_n\ \mathtt{in}\ \mathtt{val}(u), \qquad (n = 0, 1, \dots)$$

where $t_i = y_i w_{i1} \cdots w_{ik_i}$ $(1 \le i \le n)$, y_i is either a constant or a variable x_l $(1 \le l \le i-1,$ if $i \ge 2)$, and w_{im} $(1 \le m \le k_i)$ is a term with free variables all in $\{x_1, \cdots, x_{i-1}\}$. u_t is in the **Cond**-form, because for any constant d, **Cond**(d) holds. Hence by Lemma 1, the denotation of term t (the value c) is either $\bot$ or a definable value of type τ. $\qquad\square$

Lemma 2. *For any logical relation $\mathcal{R}$, $\sim_\tau\ \subseteq \mathcal{R}_\tau \implies\ \sim_{\mathsf{T}\tau}\ \subseteq \mathcal{R}_{\mathsf{T}\tau}$.*

Proof. we take any two elements $(c_1, c_2) \notin \mathcal{R}_{\mathsf{T}\tau}$ and show that there exist contexts that can distinguish them (see Appendix A for details). $\qquad\square$

Theorem 3. *Logical relations for the partial computation monad are complete up to first-order types, in the strong sense that there exists an observational logical relation $\mathcal{R}$ such that for any closed terms t_1, t_2 of a type τ^1 up to first order, if $t_1 \approx_{\tau^1} t_2$, then $[\![t_1]\!]\ \mathcal{R}_{\tau^1}\ [\![t_2]\!]$.*

Proof. Take $(\mathcal{R}_\tau)_{\tau\ type}$ induced by $\mathcal{R}_b =\sim_b$, for any base type b. It is then proved by induction on types that $\sim_{\tau^1}\subseteq \mathcal{R}_{\tau^1}$ for any type τ^1 up to first order. In particular, Lemma 2 shows the inductions step for monadic types. $\qquad\square$

4.2 Exception

The exception monad can be seen as the generalization of the partial computation monad.

$$[\![\mathsf{T}\tau]\!] = [\![\tau]\!] \cup E$$
$$[\![\mathtt{val}(t)]\!]\rho = [\![t]\!]\rho$$
$$[\![\mathtt{let}\ x \Leftarrow t_1\ \mathtt{in}\ t_2]\!]\rho = \begin{cases} [\![t_2]\!]\rho[x := [\![t_1]\!]\rho] & \text{if } [\![t_1]\!]\rho \notin E \\ [\![t_1]\!]\rho & \text{if } [\![t_1]\!]\rho \in E \end{cases}$$

where E is a fixed set of exceptions. Logical relations at monadic types are given by [2]:

$$c_1\ \mathcal{R}_{\mathsf{T}\tau}\ c_2 \Longleftrightarrow c_1\ \mathcal{R}_\tau\ c_2\ \text{ or }\ c_1 = c_2 \in E$$

Let **Cond** be the smallest set of closed terms such that, for any closed term t of type $\tau_1 \to \cdots \to \tau_n \to \mathsf{T}\tau$, **Cond**($t$) holds if and only if, for any terms $\vdash t_1 : \tau_1, \cdots, \vdash t_n : \tau_n$, $[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$ is either:

- an exception e in E, or
- definable at type τ (by some closed term t'), and, if τ is again of the form $\tau_1' \to \cdots \to \tau_m' \to \mathsf{T}\tau'$, then **Cond**($t'$) holds.

We assume that for any constant d, **Cond**(d) holds. We also assume that there is, for every type τ and every exception $e \in E$, a constant $\mathtt{raise}_\tau^e$ of type $\mathsf{T}\tau$ such that $[\![\mathtt{raise}_\tau^e]\!] = e$. Clearly, **Cond**($\mathtt{raise}_\tau^e$) holds.

Proposition 4 *A value $c \in [\![\mathsf{T}\tau]\!]$ is definable at type $\mathsf{T}\tau$, if and only if, either $c \in [\![\tau]\!]$ and c is definable at type τ, or $c = e$ for some $e \in E$.*

Lemma 3. *For any logical relation $\mathcal{R}$, $\sim_\tau \subseteq \mathcal{R}_\tau \implies \sim_{\mathsf{T}\tau} \subseteq \mathcal{R}_{\mathsf{T}\tau}$.*

Theorem 4. *Logical relations for the exception monad are complete up to first-order types, in the strong sense that there exists an observational logical relation $(\mathcal{R}_\tau)_{\tau \ type}$ such that for any closed terms t_1, t_2 of any type τ^1 up to first order, if $t_1 \approx_{\tau^1} t_2$, then $[\![t_1]\!] \ \mathcal{R}_{\tau^1} \ [\![t_2]\!]$.*

4.3 Non-determinism

$$[\![\mathsf{T}\tau]\!] = \mathbb{P}_{\text{fin}}([\![\tau]\!])$$
$$[\![\mathtt{val}(t)]\!] \rho = \{ [\![t]\!] \rho \}$$
$$[\![\mathtt{let} \ x \Leftarrow t_1 \ \mathtt{in} \ t_2]\!] \rho = \bigcup_{a \in [\![t_1]\!]\rho} [\![t_2]\!] \rho[x := a]$$

where $\mathbb{P}_{\text{fin}}(S)$ is the set of finite subsets of S. Logical relations at monadic types are given by [2]:

$$c_1 \ \mathcal{R}_{\mathsf{T}\tau} \ c_2 \Longleftrightarrow (\forall a_1 \in c_1. \ \exists a_2 \in c_2. \ a_1 \ \mathcal{R}_\tau \ a_2) \ \& \ (\forall a_2 \in c_2. \ \exists a_1 \in c_1. \ a_1 \ \mathcal{R}_\tau \ a_2) \tag{7}$$

Let **Cond** be the smallest set of closed terms such that, for any closed term t of type $\tau_1 \to \cdots \to \tau_n \to \mathsf{T}\tau$, **Cond**$(t)$ holds if and only if:

- for any closed terms $\vdash t_1 : \tau_1, \cdots, \vdash t_n : \tau_n$, $[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$ is a finite set where each element is definable at type τ (by a closed term t'), and,
- if τ is again of the form $\tau'_1 \to \cdots \to \tau'_m \to \mathsf{T}\tau'$, then, for every t', **Cond**(t') holds.

We assume that for any constant $\mathtt{d}$, **Cond**$(\mathtt{d})$ holds. We also assume there is, for every τ, a constant $+_\tau$ of type $\tau \to \tau \to \mathsf{T}\tau$ and a constant Φ_τ of type $\mathsf{T}\tau$ such that for any $a_1, a_2 \in [\![\tau]\!]$, $[\![+_\tau]\!](a_1, a_2) = \{a_1\} \cup \{a_2\}$ and $[\![\Phi_\tau]\!] = \emptyset$. Obviously, **Cond**$(+_\tau)$ and **Cond**(Φ_τ) hold.

Proposition 5 *A value $c \in [\![\mathsf{T}\tau]\!]$ is definable if and only if, for any $a \in c$, a is definable at type τ.*

However, in the case of non-determinism, we are not able to achieve the completeness of logical relations for any type up to first order. There is indeed counterexamples where two first-order programs are contextually equivalent but they are not related by the logical relations as defined by (7).
Consider the following two programs:

$$\vdash \mathtt{val}(\lambda x.(\mathtt{true} +_{\mathsf{bool}} \mathtt{false})) : \mathsf{T}(\mathsf{bool} \to \mathsf{Tbool}),$$
$$\vdash \lambda x.\mathtt{val}(\mathtt{true}) \ +_{\mathsf{bool} \to \mathsf{Tbool}} \ \lambda x.\mathtt{val}(\mathtt{false}) : \mathsf{T}(\mathsf{bool} \to \mathsf{Tbool}).$$

The two programs are contextually equivalent: what contexts can do is to apply the functions to some arguments and observe the results. While doing so, we always get the same set of possible values ($\{\mathtt{true}, \mathtt{false}\}$), although the two programs contain different set of possible functions. So there is no way to distinguish them with a context. But they cannot be related by a logical relation, because the function $[\![\lambda x.(\mathtt{true} +_{\mathsf{bool}} \mathtt{false})]\!]$ from the first program is not related to any function from the second program.

Indeed, if we assume that for every non-observable base type b, there is an equality test constant $\mathtt{test}_b : b \to b \to \mathsf{bool}$ (clearly, **Cond**$(\mathtt{test}_b)$ holds), logical relations for the non-determinism monad are then complete for a set of *weak first-order types*:

$$\tau_{\mathrm{w}}^1 ::= b \mid \mathsf{T}b \mid b \to \tau_{\mathrm{w}}^1.$$

Compared to all types up to first oder, weak first-order types do not contain monadic types of functions.

Theorem 5. *Logical relations for the non-determinism monad are complete up to weak first-order types. in the strong sense that there exists an observational logical relation $\mathcal{R}$ such that for any closed terms t_1, t_2 of a weak first-order type τ_w^1, if $t_1 \approx_{\tau_w^1} t_2$, then $[\![t_1]\!] \mathcal{R}_{\tau_w^1} [\![t_2]\!]$.*

Proof. Take the logical relation $\mathcal{R}$ induced by $\mathcal{R}_b = \sim_b$, for any base type b. $\qquad\qquad\square$

Now let state and label be base types such that label is an observation type, whereas state is not. Using non-determinism monad, we can define labeled transition systems as elements of $[\![\text{state} \to \text{label} \to \text{Tstate}]\!]$, with states in $[\![\text{state}]\!]$ and labels in $[\![\text{label}]\!]$, as functions mapping states a and labels l to the set of states b such that $a \xrightarrow{\;l\;} b$. The logical relation at type $\text{state} \to \text{label} \to \text{Tstate}$ is given by [2]:

$$(f_1, f_2) \in \mathcal{R}_{\text{state} \to \text{label} \to \text{Tstate}} \iff$$
$$\forall a_1, a_2, l_1, l_2 \cdot (a_1, a_2) \in \mathcal{R}_{\text{state}} \;\&\; (l_1, l_2) \in \mathcal{R}_{\text{label}} \implies$$
$$(\forall b_1 \in f_1(a_1, l_1) \cdot \exists b_2 \in f_2(a_2, l_2) \cdot (b_1, b_2) \in \mathcal{R}_{\text{state}})$$
$$\&\; (\forall b_2 \in f_2(a_2, l_2) \cdot \exists b_1 \in f_1(a_1, l_1) \cdot (b_1, b_2) \in \mathcal{R}_{\text{state}})$$

In case $\mathcal{R}_{\text{label}}$ is equality, f_1 and f_2 are logically related if and only if $\mathcal{R}_{\text{state}}$ is a *strong bisimulation* between the labeled transition systems f_1 and f_2.

Sometimes we explicitly specify an initial state for certain labeled transition system. In this case, the encoding of the labeled transition system in the nondeterminism monad is a pair (q, f) of $[\![\text{state} \times (\text{state} \to \text{label} \to \text{Tstate})]\!]$, where q is the initial state and f is the transition relation as defined above. Then (q_1, f_1) and (q_2, f_2) are logically related if and only if they are strongly bisimular, i.e., $\mathcal{R}_{\text{state}}$ is a strong bisimulation between the two labeled transition systems and $q_1 \mathcal{R}_{\text{state}} q_2$.

Corollary 1 (Soundness of strong bisimulation). *Let f_1 and f_2 be transition systems. If there exists a strong bisimulation between f_1 and f_2, then f_1 and f_2 are contextually equivalent.*

Proof. There exists a strong bisimulation between f_1 and f_2, therefore f_1 and f_2 are logically related. By Theorem 1, f_1 and f_2 are thus contextually equivalent. $\qquad\qquad\square$

In order to prove completeness, we need to assume that label has no *junk*, in the sense that every value of $[\![\text{label}]\!]$ is definable.

Corollary 2 (Completeness of strong bisimulation). *Let f_1 and f_2 be transition systems which are definable. If f_1 and f_2 are contextually equivalent and label has no junk, then there exists a strong bisimulation between f_1 and f_2.*

Proof. Let $\mathcal{R}$ be the logical relation given by By Theorem 5. f_1 and f_2 are definable and contextually equivalent, therefore $f_1 \; \mathcal{R}_{\text{state} \to \text{label} \to \text{Tstate}} \; f_2$. Moreover, because label has no junk, $\mathcal{R}_{\text{label}}$ is equality. $\mathcal{R}_{\text{state}}$ is thus a strong bisimulation between f_1 and f_2. $\qquad\qquad\square$

4.4 State transformers

$$[\![\text{T}\tau]\!] = ([\![\tau]\!] \times St)^{St}$$
$$[\![\texttt{val}(t)]\!]\rho = s \mapsto ([\![t]\!]\rho, s)$$
$$[\![\texttt{let } x \Leftarrow t_1 \texttt{ in } t_2]\!]\rho = s \mapsto [\![t_2]\!]\rho[x := a_1](s_1)$$
$$\text{where } a_1 = \pi_1([\![t_1]\!]\rho(s)), s_1 = \pi_2([\![t_2]\!]\rho(s))$$

where St is a finite set of states. Logical relations at monadic types are given by [2]:

$$c_1 \, \mathcal{R}_{\text{T}\tau} \, c_2 \iff \forall s \in St. \; \pi_1(c_1 s) \, \mathcal{R}_\tau \, \pi_1(c_2 s) \;\&\; \pi_2(c_1 s) = \pi_2(c_2 s)$$

Let **Cond** be the smallest set of closed terms such that, for any closed term t of type $\tau_1 \to \cdots \to \tau_n \to \text{T}\tau$, **Cond**$(t)$ holds if and only if,

- for any closed terms $\vdash t_1 : \tau_1, \cdots, \vdash t_n : \tau_n$, $[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$ is a function such that for any $s \in St$, $[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])(s) = (a, s')$ where $s' \in St$ and a is definable at type τ (by some closed term t'), and
- if τ is of the form $\tau'_1 \to \cdots \to \tau'_m \to \mathsf{T}\tau'$, then $\mathbf{Cond}(t')$ holds.

We assume that for any constant d, $\mathbf{Cond}(\mathsf{d})$ holds. Let unit be the base type which contains only a dummy value $*$. We assume that there is, for each $s \in St$, a constant update_s of type Tunit such that for any $s' \in St$, $[\![\mathsf{update}_s]\!](s') = (*, s)$. This constant does nothing but change the current state to s. Clearly, $\mathbf{Cond}(\mathsf{update}_s)$ holds.

Proposition 6 *If a value $c \in [\![\mathsf{T}\tau]\!]$ is definable, then for every $s \in St$, $\pi_1(cs)$ is definable at type τ.*

Lemma 4. *For any logical relation $(\, \mathcal{R}_\tau \,)_{\tau \text{ type}}$, $\sim_\tau \subseteq \mathcal{R}_\tau \implies \sim_{\mathsf{T}\tau} \subseteq \mathcal{R}_{\mathsf{T}\tau}$.*

Theorem 6. *Logical relations for the state monad are complete up to first-order types, in the strong sense that there exists an observational logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that for any closed terms t_1, t_2 of any type τ^1 up to first order, if $t_1 \approx_{\tau^1} t_2$, then $[\![t_1]\!] \, \mathcal{R}_{\tau^1} \, [\![t_2]\!]$.*

4.5 Continuation

The semantics of monadic types and constructs are given by

$$[\![\mathsf{T}\tau]\!] = R^{R^{[\![\tau]\!]}}$$
$$[\![\mathsf{val}(t)]\!]\rho = k^{[\![\tau]\!] \to R} := k([\![t]\!]\rho)$$
$$[\![\mathsf{let}\ x \Leftarrow t_1\ \mathsf{in}\ t_2]\!]\rho = k^{[\![\tau_2]\!] \to R} := [\![t_1]\!]\rho(k')$$
$$\text{where } k' \text{ is a function: } v^{[\![\tau_1]\!]} := [\![t_2]\!]\rho[x := v](k)$$

R is a set of possible results. Logical relations for monadic types are given by [2]:

$$c_1 \, \mathcal{R}_{\mathsf{T}\tau} \, c_2 \iff (\forall k_1, k_2. \ (\forall a_1, a_2. \ a_1 \, \mathcal{R}_\tau \, a_2 \implies k_1(a_1) = k_2(a_2)) \implies c_1(k_1) = c_2(k_2))$$

Intuitively, we can always observe the results returned by any continuation, or test the equality between these results, so when we call a continuation with two contextually equivalent arguments, it should return the same result. Furthermore, if two continuations agree on any contextually equivalent argument, we shall regard them as equivalent. Precisely, Two continuations $k_1, k_2 \in R^{[\![\tau]\!]}$ are *equivalent* if and only if, for any $a_1, a_2 \in [\![\tau]\!]$, $a_1 \sim_\tau a_2 \implies k_1(a_1) = k_2(a_2)$.

Let $\mathbf{Cond}$ be the set of closed terms such that, for any closed term t of type $\tau_1 \to \cdots \to \tau_n \to \mathsf{T}\tau$, $\mathbf{Cond}(t)$ holds if and only if,

- for any closed terms $\vdash t_1 : \tau_1, \cdots, \vdash t_n : \tau_n$, $[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$ is a function such that for any equivalent continuations $k_1, k_2 \in R^{[\![\tau]\!]}$,

$$[\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])(k_1) = [\![t]\!]([\![t_1]\!], \cdots, [\![t_n]\!])(k_2).$$

We assume that for any constant d, $\mathbf{Cond}(\mathsf{d})$ holds. We also assume that there is, for each τ and each continuation $k \in R^{[\![\tau]\!]}$, a constant call_τ^k of type $\tau \to \mathsf{T}$bool such that for any $a \in [\![\tau]\!]$ and any continuation $k' \in R^{\mathsf{bool}}$, $[\![\mathsf{call}_\tau^k]\!](a)(k') = k(a)$. $\mathbf{Cond}(\mathsf{call}_\tau^k)$ holds.

Proposition 7 *If a value $c \in [\![\mathsf{T}\tau]\!]$ is definable at type $\mathsf{T}\tau$, then for every equivalent continuations $k_1, k_2 \in R^{[\![\tau]\!]}$, $c(k_1) = c(k_2)$.*

Lemma 5. *For any logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$, $\sim_\tau \subseteq \mathcal{R}_\tau \implies \sim_{\mathsf{T}\tau} \subseteq \mathcal{R}_{\mathsf{T}\tau}$.*

Theorem 7. *Logical relations for the continuation monad are complete up to first-order types, in the strong sense that there exists an observational logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that for any closed terms t_1, t_2 of any type τ^1 up to first order, if $t_1 \approx_{\tau^1} t_2$, then $[\![t_1]\!] \, \mathcal{R}_{\tau^1} \, [\![t_2]\!]$.*

References

1. P. N. Benton, G. M. Bierman, and V. C. V. de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, Mar. 1998.
2. J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Proceedings of the 16th International Workshop on Computer Science Logic (CSL'02)*. Springer-Verlag LNCS 2471, 2002.
3. J. Goubault-Larrecq, S. Lasota, D. Nowak, and Y. Zhang. Complete lax logical relations for cryptographic lambda-caculi. In *Proceedings of the 18th International Workshop on Computer Science Logic (CSL'04)*. Springer-Verlag LNCS 3210, 2004.
4. F. Honsell and D. Sannella. Pre-logical relations. In *Proceedings of 13rd International Workshop Computer Science Logica (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 546–561. Springer-Verlag, 1999.
5. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
6. J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In *Proceedings of the 6th International Workshop on Computer Science Logic (CSL'92)*, pages 352–378. Springer Verlag LNCS 702, 1993.
7. E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.
8. G. Plotkin, J. Power, D. Sannella, and R. Tennent. Lax logical relations. In *Proceedings of the 7th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, pages 85–102. Springer Verlag LNCS 1853, 2000.

A Proofs

Proof of Proposition 1 We define the relation $\sim_\tau$ by: $a_1 \sim_\tau a_2$ if and only if a_1, a_2 are definable and $a_1 \approx_\tau a_2$. Let $(\mathcal{R}_\tau)_{\tau \text{ type}}$ be the logical relation induced by $\mathcal{R}_b = \sim_b$ at all base types b.

The proof is by induction over τ. Case $\tau = b$ is obvious. Let $\tau = b \to \tau'$. Take two terms t_1, t_2 of type τ such that $[\![t_1]\!]$ and $[\![t_2]\!]$ are related by $\approx_{b \to \tau'}$. Let $f_1 = [\![t_1]\!]$ and $f_2 = [\![t_2]\!]$. Assume that $a_1, a_2 \in [\![b]\!]$ are related by $\mathcal{R}_b$, therefore $a_1 \sim_b a_2$ since $\mathcal{R}_b = \sim_b$. Clearly, a_1 and a_2 are thus definable, say by terms u_1 and u_2, respectively. Then, for any context $\mathbb{C}$ such that $x : \tau' \vdash \mathbb{C} : o$ ($o \in \mathbf{Obs}$) is derivable,

$$
\begin{aligned}
&[\![\mathbb{C}]\!][x := f_1(a_1)] \\
&= [\![\mathbb{C}[xu_1/x]]\!][x := f_1] \quad (\text{since } a_1 = [\![u_1]\!]) \\
&= [\![\mathbb{C}[xu_1/x]]\!][z := f_2] \quad (\text{since } f_1 \approx_{b \to \tau'} f_2) \\
&= [\![\mathbb{C}]\!][x := f_2(a_1)] \\
&= [\![\mathbb{C}[t_2x/x]]\!][x := a_1] \quad (\text{since } f_2 = [\![t_2]\!]) \\
&= [\![\mathbb{C}[t_2x/x]]\!][x := a_2] \quad (\text{since } a_1 \approx_b a_2) \\
&= [\![\mathbb{C}]\!][x := f_2(a_2)].
\end{aligned}
$$

Hence $f_1(a_1) \approx_{\tau'} f_2(a_2)$. Moreover, $f_1(a_1)$ and $f_2(a_2)$ are therefore definable by t_1u_1 and t_2u_2 respectively. By induction hypothesis, $f_1(a_1) \mathcal{R}_{\tau'} f_2(a_2)$. Because a_1 and a_2 are arbitrary, we conclude that $f_1 \mathcal{R}_{b \to \tau'} f_2$. $\qquad\qquad\square$

Proof of Proposition 2 The computational λ-calculus is strongly normalizing [1], so we consider the β-normal form of term t and prove it by induction on t.

If t is a variable or a constant, then according to the equation (2)

$$[\![t]\!]\rho = [\![\texttt{let } x \Leftarrow t \texttt{ in val}(x)]\!]\rho,$$

where x is not free in t.

If t is an application $t_1t_2 \cdots t_n$, then t_1 is of a functions type and it must be a variable or a constant (it cannot be a λ-abstraction since t is β-normal). Similarly, t is equivalent to the term $\texttt{let } x \Leftarrow t \texttt{ in val}(x)$.

If t is a trivial computation $\texttt{val}(t')$, it is already in the computational canonical form.

If t is a sequential computation $\mathtt{let}\ x \Leftarrow t_1\ \mathtt{in}\ t_2$, by induction, there are computational canonical terms for both t_1 and t_2, namely

$$\mathtt{let}\ x_1^1 \Leftarrow t_1^1\ \mathtt{in} \cdots \mathtt{let}\ x_m^1 \Leftarrow t_m^1\ \mathtt{in}\ \mathtt{val}(u_1)$$

and

$$\mathtt{let}\ x_1^2 \Leftarrow t_1^2\ \mathtt{in} \cdots \mathtt{let}\ x_n^2 \Leftarrow t_n^2\ \mathtt{in}\ \mathtt{val}(u_2),$$

where $x_1^1, \ldots, x_m^1, x_1^2, \ldots, x_n^2$ are not free in t. Replace t_1 and t_2 with these terms in t and we get

$$
\begin{aligned}
[\![t]\!]\rho &= [\![\mathtt{let}\ x \Leftarrow (\mathtt{let}\ x_1^1 \Leftarrow t_1^1\ \mathtt{in} \cdots \mathtt{let}\ x_m^1 \Leftarrow t_m^1\ \mathtt{in}\ \mathtt{val}(u_1))\ \mathtt{in}\ t_2]\!]\rho \\
&= [\![\mathtt{let}\ x_1^1 \Leftarrow t_1^1\ \mathtt{in} \\
&\qquad \mathtt{let}\ x \Leftarrow (\mathtt{let}\ x_2^1 \Leftarrow t_2^1\ \mathtt{in} \cdots \mathtt{let}\ x_m^1 \Leftarrow t_m^1\ \mathtt{in}\ \mathtt{val}(u_1))\ \mathtt{in}\ t_2]\!]\rho \\
&= \quad \cdots \quad \cdots \\
&= [\![\mathtt{let}\ x_1^1 \Leftarrow t_1^1\ \mathtt{in} \cdots \mathtt{let}\ x_m^1 \Leftarrow t_m^1\ \mathtt{in}\ \mathtt{let}\ x \Leftarrow \mathtt{val}(u_1)\ \mathtt{in}\ t_2]\!]\rho \\
&= [\![\mathtt{let}\ x_1^1 \Leftarrow t_1^1\ \mathtt{in} \cdots \mathtt{let}\ x_m^1 \Leftarrow t_m^1\ \mathtt{in}\ \mathtt{let}\ x \Leftarrow \mathtt{val}(u_1)\ \mathtt{in} \\
&\qquad \mathtt{let}\ x_1^2 \Leftarrow t_1^2\ \mathtt{in} \cdots \mathtt{let}\ x_n^2 \Leftarrow t_n^2\ \mathtt{in}\ \mathtt{val}(u_2)]\!]\rho.
\end{aligned}
$$

Because all $t_1^1, \ldots, t_m^1, t_1^2, \ldots, t_n^2$ and $\mathtt{val}(u)$ are weak head normal forms, so the last term in the above equation is computational canonical. $\qquad\square$

Proof of Theorem 2 Obviously, $t_1 \approx_\tau t_2 \implies [\![t_1]\!] \sim_\tau [\![t_2]\!]$, so we will prove that $(\sim_\tau)_{\tau\ \text{type}}$ is a pre-logical relation. The conditions (ii) – (iv), for $(\sim_\tau)_{\tau\ \text{type}}$, are clearly satisfied: $[\![t]\!] \sim_\tau [\![t]\!]$ holds for every closed term t of type τ, hence *a fortiori* for K, S and any constant c.

To show (i), assume $f_1 \sim_{\tau\to\tau'} f_2$. In particular f_1 and f_2 are definable, say by closed terms t_1 and t_2, respectively. And for all context $\mathbb{C}_{\text{fun}}$, such that $x : \tau \to \tau' \vdash \mathbb{C}_{\text{fun}} : \mathsf{T}o$ is derivable ($o \in \mathbf{Obs}$), $[\![\mathbb{C}_{\text{fun}}]\!][x := f_1] = [\![\mathbb{C}_{\text{fun}}]\!][x := f_2]$.

Furthermore assume $a_1 \sim_\tau a_2$. Again, a_1 and a_2 are definable, say by respective closed terms u_1 and u_2. And for all context $\mathbb{C}_{\text{arg}}$, such that $x : \tau \vdash \mathbb{C}_{\text{arg}} : \mathsf{T}o$ is derivable ($o \in \mathbf{Obs}$), $[\![\mathbb{C}_{\text{arg}}]\!][x := a_1] = [\![\mathbb{C}_{\text{arg}}]\!][x := a_2]$.

We have to show that $f_1(a_1) \sim_{\tau'} f_2(a_2)$. $f_1(a_1)$ and $f_2(a_2)$ are respectively definable by $t_1 u_1$ and $t_2 u_2$. Let $\mathbb{C}$ be any context such that, for any $o \in \mathbf{Obs}$, $x : \tau' \vdash \mathbb{C} : \mathsf{T}o$ is derivable. The rest of the argument is by context chasing, similarly as in the proof of Proposition 1:

$$
\begin{aligned}
&[\![\mathbb{C}]\!][x := f_1(a_1)] \\
={}& [\![\mathbb{C}[xu_1/x]]\!][x := f_1] && (\text{since } a_1 = [\![u_1]\!]) \\
={}& [\![\mathbb{C}[xu_1/x]]\!][x := f_2] && (\text{since } f_1 \approx_{\tau\to\tau'} f_2) \\
={}& [\![\mathbb{C}]\!][x := f_2(a_1)] \\
={}& [\![\mathbb{C}[t_2x/x]]\!][x := a_1] && (\text{since } f_2 = [\![t_2]\!]) \\
={}& [\![\mathbb{C}[t_2x/x]]\!][x := a_2] && (\text{since } a_1 \approx_\tau a_2) \\
={}& [\![\mathbb{C}]\!][x := f_2(a_2)]
\end{aligned}
$$

Proof of Lemma 1 Because t is of the $\mathbf{Cond}$-form,

$$t \equiv \mathtt{let}\ x_1 \Leftarrow t_1\ \mathtt{in} \cdots \mathtt{let}\ x_n \Leftarrow t_n\ \mathtt{in}\ \mathtt{val}(u), \qquad (n = 0, 1, \ldots).$$

We prove by induction on n:

– In the base case ($n = 0$): $[\![\mathtt{val}(u)]\!] = [\![u]\!]$. It is obvious that $[\![t]\!]$ is definable at type τ (by the term u namely).

– For any $n \geq 1$,

$$[\![t_1]\!] = [\![u_1 w_{11} \cdots w_{1k_1}]\!] = [\![u_1]\!]([\![w_{11}]\!], \cdots, [\![w_{1k_1}]\!]).$$

where $u_1, w_{11}, \cdots, w_{1k_1}$ are all closed terms and $\mathbf{Cond}(u_1)$ holds, so $[\![t_1]\!]$ is either equal to $\bot$ or definable at type τ_1 (suppose t_1 is of type $\mathsf{T}\tau_1$). If $[\![t_1]\!] = \bot$, then the denotation of the whole term is $\bot$, i.e., $[\![t]\!] = \bot$. If $[\![t_1]\!] \neq \bot$, suppose $[\![t_1]\!]$ is defined by a closed term t_1' (of type τ_1). Because $\mathbf{Cond}(u_1)$ holds, so does $\mathbf{Cond}(t_1')$, then $\mathbf{Cond}(u_2[t_1'/x_1])$, ..., $\mathbf{Cond}(u_n[t_1'/x_1])$ hold as well (because $u_2[t_1'/x_1], \ldots, u_n[t_1'/x_1]$ are either t_1' or a constant). Let $t_i' = t_i[t_1'/x_1]$ $(2 \leq i \leq n)$, then

$$[\![\texttt{let } x_1 \Leftarrow t_1 \texttt{ in let } x_2 \Leftarrow t_2 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u)]\!]$$
$$= [\![\texttt{let } x_2 \Leftarrow t_2 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u)]\!][x_1 := [\![t_1']\!]]$$
$$= [\![\texttt{let } x_2 \Leftarrow t_2' \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n' \texttt{ in val}(u[t_1'/x_1])]\!].$$

Clearly, $\texttt{let } x_2 \Leftarrow t_2' \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n' \texttt{ in val}(u[t_1'/x_1])$ is again of the $\mathbf{Cond}$-form, so by induction, its denotation is either $\bot$ or a value definable at type τ. $\qquad\square$

Proof of Lemma 2 We assume that $\sim_\tau \subseteq \mathcal{R}_\tau$. Take any two elements $(c_1, c_2) \notin \mathcal{R}_{\mathsf{T}\tau}$. There are two cases:

– $c_1, c_2 \in [\![\tau]\!]$ but $(c_1, c_2) \notin \mathcal{R}_\tau$, then $c_1 \not\sim_\tau c_2$. If one of these two values is not definable at type τ, by Proposition 3, it is not definable at type $\mathsf{T}\tau$ either. If both values are definable at type τ but they are not contextually equivalent, then there is a context $x : \tau \vdash \mathbb{C} : \mathsf{T}o$ such that $[\![\mathbb{C}]\!][x := c_1] \neq [\![\mathbb{C}]\!][x := c_2]$. Thus, the context $y : \mathsf{T}\tau \vdash \texttt{let } x \Leftarrow y \texttt{ in } \mathbb{C} : \mathsf{T}o$ can distinguish c_1 and c_2 (as two values of type $\mathsf{T}\tau$).
– $c_1 \in [\![\tau]\!]$ and $c_2 = \bot$ (or symmetrically, $c_1 = \bot$ and $c_2 \in [\![\tau]\!]$), then the context $\texttt{let } x \Leftarrow y \texttt{ in val(true)}$ can be used to distinguish them.

In both cases, $c_1 \not\sim_{\mathsf{T}\tau} c_2$, hence $\sim_{\mathsf{T}\tau} \subseteq \mathcal{R}_{\mathsf{T}\tau}$. $\qquad\square$

Proof of Proposition 5 We prove first the following lemma: *In the computational λ-calculus specialized in non-determinism, for any closed term t (of type $\mathsf{T}\tau$) in $\mathbf{Cond}$-form, $[\![t]\!]$ is a finite set of definable values of $[\![\tau]\!]$.*
    Because t is of the $\mathbf{Cond}$-form,

$$t \equiv \texttt{let } x_1 \Leftarrow t_1 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u), \qquad (n = 0, 1, \ldots).$$

We prove by induction on n:

– In the base case $(n = 0)$: $[\![\texttt{val}(u)]\!] = \{[\![u]\!]\}$. It is obvious that $[\![u]\!]$ is definable at type τ (by the term u in particular).
– For any $n \geq 1$,

$$[\![t_1]\!] = [\![u_1 w_{11} \cdots w_{1k_1}]\!] = [\![u_1]\!]([\![w_{11}]\!], \cdots, [\![w_{1k_1}]\!]).$$

where $u_1, w_{11}, \cdots, w_{1k_1}$ are all closed terms and $\mathbf{Cond}(u_1)$ holds, so every element of $[\![t_1]\!]$ is definable at type τ_1 (suppose t_1 is of type $\mathsf{T}\tau_1$). Suppose that for every $a \in [\![t_1]\!]$, there is a closed term t_1^a such that $[\![t_1^a]\!] = a$. Because $\mathbf{Cond}(u_1)$ holds, for every $a \in [\![t_1]\!]$, $\mathbf{Cond}(t_1^a)$ holds as well, hence $\mathbf{Cond}(u_2[t_1^a/x_1])$, ..., $\mathbf{Cond}(u_n[t_1^a/x_1])$ hold (because $u_2[t_1^a/x_1], \ldots, u_n[t_1^a/x_1]$ are either t_1^a or a constant). Let $t_i^a = t_i[t_1^a/x_1]$ $(2 \leq i \leq n)$, then

$$[\![\texttt{let } x_1 \Leftarrow t_1 \texttt{ in let } x_2 \Leftarrow t_2 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u)]\!]$$
$$= \bigcup_{a \in [\![t_1]\!]} [\![\texttt{let } x_2 \Leftarrow t_2 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u)]\!][x_1 := a]$$
$$= \bigcup_{a \in [\![t_1]\!]} [\![\texttt{let } x_2 \Leftarrow t_2^a \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n^a \texttt{ in val}(u[t_1^a/x_1])]\!].$$

Clearly, for every $a \in [\![t_1]\!]$, $\texttt{let } x_2 \Leftarrow t_2^a \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n^a \texttt{ in val}(u[t_1^a/x_1])$ is again in **Cond**-form, so by induction, its denotation is a finite set of definable values of type τ, and so is the union of all these sets, since $[\![t_1]\!]$ is also finite.

The proposition follows by considering the computational canonical form of t, as in Proposition 3. $\quad\square$

Proof of Theorem 5 Take the logical relation $\mathcal{R}$ induced by $\mathcal{R}_b = \sim_b$, for any base type b. We prove by induction on types that $\sim_{\tau_w^1} \subseteq \mathcal{R}_{\tau_w^1}$ for any weak first-order type τ_w^1.

Cases b and $b \to \tau_w^1$ go identically as in normal typed lambda-calculi. For monadic types $tcompb$, suppose that $(c_1, c_2) \notin \mathcal{R}_{\mathsf{T}b}$, which means either there is a value in c_1 such that no value of c_2 is related to it, or there is such a value in c_2. We assume that every value in c_1 and c_2 is definable (otherwise it is obvious that $c_1 \not\sim_{\mathsf{T}b} c_2$ because at least one of them is not definable, according to Proposition 5). Suppose there is a value $a \in c_1$ such that no value in c_2 is related to it, and a can be defined by a closed term t of type b. Then the following context can distinguish c_1 and c_2:

$$x : \mathsf{T}\tau \vdash \texttt{let } y \Leftarrow x \texttt{ in test}_b(y, t) : \mathsf{Tbool}$$

since every value in c_2 is not contextually equivalent to a, hence not equal to a. $\quad\square$

Proof of Proposition 6 We prove first the following lemma: *In the computational λ-calculus specialized in state transformers, for any closed term t (of type $\mathsf{T}\tau$) in* **Cond**-*form, and for any $s \in St$, $\pi_1([\![t]\!]s)$ is definable at type τ.*

Because t is of the **Cond**-form,

$$t \equiv \texttt{let } x_1 \Leftarrow t_1 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u), \qquad (n = 0, 1, 2, \ldots).$$

We prove by induction on n:

- In the base case ($n = 0$), for every $s \in St$, $[\![\texttt{val}(u)]\!]s = ([\![u]\!], s)$. It is obvious that $[\![u]\!]$ is definable at type τ (by the term u in particular).
- For any $n \geq 1$,
$$[\![t_1]\!] = [\![u_1 w_{11} \cdots w_{1k_1}]\!] = [\![u_1]\!]([\![w_{11}]\!], \cdots, [\![w_{1k_1}]\!]).$$

where $u_1, w_{11}, \cdots, w_{1k_1}$ are all closed terms and $\mathbf{Cond}(u_1)$ holds, so for every $s \in St$, $\pi_1([\![t_1]\!](s))$ is definable at type τ_1 (suppose t_1 is of type $\mathsf{T}\tau_1$). Suppose that for every $s \in St$, t_1^s is a closed term of type τ_1 such that $\pi_1([\![t_1]\!](s)) = [\![t_1^s]\!]$. Because $\mathbf{Cond}(u_1)$ holds, $\mathbf{Cond}(t_1^s)$ holds as well, hence $\mathbf{Cond}(u_2[t_1^s/x_1])$, ..., $\mathbf{Cond}(u_n[t_1^s/x_1])$ hold (because $u_2[t_1^s/x_1], \ldots, u_n[t_1^s/x_1]$ are either t_1^s or a constant). For every $s \in St$, let $t_i^s = t_i[t_1^s/x_1]$ ($2 \leq i \leq n$), then

$$\begin{aligned}
&[\![\texttt{let } x_1 \Leftarrow t_1 \texttt{ in let } x_2 \Leftarrow t_2 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u)]\!](s) \\
&= [\![\texttt{let } x_2 \Leftarrow t_2 \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n \texttt{ in val}(u)]\!][x := [\![t_1^s]\!]](s') \\
&= [\![\texttt{let } x_2 \Leftarrow t_2^s \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n' \texttt{ in val}(u[t_1^s/x_1])]\!](s')
\end{aligned}$$

where $s' = \pi_2([\![t_1]\!](s))$. Clearly, for every $a \in [\![t_1]\!]$,

$$\texttt{let } x_2 \Leftarrow t_2^s \texttt{ in} \cdots \texttt{let } x_n \Leftarrow t_n^s \texttt{ in val}(u[t_1^s/x_1])$$

is again in **Cond**-form, so by induction, its denotation, when applied to any state, is a pair of a definable value at type τ and a state in St.

The proposition follows by considering the computational canonical form of t, as in Proposition 3. $\quad\square$

Proof of Lemma 4 We assume that $(c_1, c_2) \notin \mathcal{R}_{\mathsf{T}\tau}$, so there exists some $s_0 \in St$ such that

- either $(\pi_1(c_1 s_0), \pi_1(c_2 s_0)) \notin \mathcal{R}_\tau$. Then by induction $\pi_1(c_1 s_0) \not\sim_\tau \pi_1(c_2 s_0)$. If $\pi_1(c_i s_0)$ $(i = 1, 2)$ is not definable, then by Proposition 6, c_i is not definable either. If both $\pi_1(c_1 s_0)$ and $\pi_1(c_2 s_0)$ are definable, but $\pi_1(c_1 s_0) \not\approx_\tau \pi_1(c_2 s_0)$, then there is a context $x : \tau \vdash \mathbb{C} : \mathsf{T}o$ such that $[\![\mathbb{C}]\!][x := \pi_1(c_1 s_0)] \neq [\![\mathbb{C}]\!][x := \pi_1(c_2 s_0)]$, i.e., for some state $s_0' \in St$,

$$[\![\mathbb{C}]\!][x := \pi_1(c_1 s_0)](s_0') \neq [\![\mathbb{C}]\!][x := \pi_1(c_1 s_0)](s_0')$$

Now we can use the following context

$$y : \mathsf{T}\tau \vdash \mathtt{let}\ x \Leftarrow y\ \mathtt{in}\ \mathtt{let}\ z \Leftarrow \mathtt{update}_{s_0'}\ \mathtt{in}\ \mathbb{C} : \mathsf{T}o,$$

Let $f_i = [\![\mathtt{let}\ x \Leftarrow y\ \mathtt{in}\ \mathtt{do}\ \mathbb{C}\ \mathtt{at}\ s_0']\!][y := c_i]$ $(i = 1, 2)$, then for any $s \in St$,

$$f_i(s) = \left[\!\!\left[\mathtt{let}\ z \Leftarrow \mathtt{update}_{s_0'}\ \mathtt{in}\ \mathbb{C}\right]\!\!\right][x := \pi_1(c_i s)](\pi_2(c_i s))$$
$$= [\![\mathbb{C}]\!][x := \pi_1(c_i s)](s_0'), \qquad (i = 1, 2).$$

$f_1 \neq f_2$, because when applied to the state s_0, they will return two different pairs, so the above context can distinguish the two values c_1 and c_2;
- or $\pi_2(c_1 s_0) \neq \pi_2(c_2 s_0)$. we use the context

$$y : \mathsf{T}\tau \vdash \mathtt{let}\ x \Leftarrow y\ \mathtt{in}\ \mathtt{val}(\mathtt{true}) : \mathsf{T}\mathsf{bool},$$

then

$$[\![\mathtt{let}\ x \Leftarrow y\ \mathtt{in}\ \mathtt{val}(\mathtt{true})]\!][y := c_i] = \lambda s.(\mathtt{true}, \pi_2(c_i s)) \quad (i = 1, 2)$$

These two functions are not equal since they return different results when applied to the state s_0.

In both cases, $c_1 \not\approx_{\mathsf{T}\tau} c_2$, hence $\sim_{\mathsf{T}\tau} \subseteq \mathcal{R}_{\mathsf{T}\tau}$. $\qquad\square$

Proof of Lemma 5 Assume $(c_1, c_2) \notin \mathcal{R}_{\mathsf{T}\tau}$, which means that there are two $\mathcal{R}$-related continuations k_1, k_2 such that $c_1(k_1) \neq c_2(k_2)$. Because $\sim_\tau \subseteq \mathcal{R}_\tau$, for any $a_1, a_2 \in [\![\tau]\!]$,

$$a_1 \sim_\tau a_2 \Rightarrow a_1\ \mathcal{R}\ a_2 \Rightarrow k_1(a_1) = k_2(a_2),$$

then k_1 and k_2 are equivalent. Suppose that both c_1 and c_2 are definable, then by Proposition 7, $c_1(k_2) = c_1(k_1)$ and $c_2(k_2) = c_2(k_1)$. Consider the context

$$y : \mathsf{T}\tau \vdash \mathtt{let}\ x \Leftarrow y\ \mathtt{in}\ \mathtt{call}_\tau^{k_1}(a) : \mathsf{T}\mathsf{bool}.$$

For each $k \in R^{[\![\mathsf{bool}]\!]}$,

$$\left[\!\!\left[\mathtt{let}\ x \Leftarrow y\ \mathtt{in}\ \mathtt{call}_\tau^{k_1}\right]\!\!\right][y := c_i](k) = c_i(l) \qquad (i = 1, 2)$$

where for each $a \in [\![\tau]\!]$,

$$l(a) = \left[\!\!\left[\mathtt{call}_\tau^{k_1}\right]\!\!\right](a)(k) = k_1(a),$$

then $l = k_1$ and because $c_1(k_1) \neq c_2(k_2) = c_2(k_1)$, $c_1(l) \neq c_2(l)$. $\qquad\square$