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
[2] RCIS, National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
[3] Project Everest, INRIA Sophia-Antipolis, France

Abstract. Software security can be ensured by specifying and verifying security properties of software using formal methods with strong theoretical bases. In particular, programs can be modeled in the framework of lambda-calculi, and interesting properties can be expressed formally by contextual equivalence (a.k.a. observational equivalence). Furthermore, imperative features, which exist in most real-life software, can be nicely expressed in the so-called computational lambda-calculus. Contextual equivalence is difficult to prove directly, but we can often use logical relations as a tool to establish it in lambda-calculi. We have already defined logical relations for the computational lambda-calculus in previous work. We devote this paper to the study of their completeness w.r.t. contextual equivalence in the computational lambda-calculus.

1 Introduction

Contextual equivalence. Two programs are contextually equivalent (a.k.a. observationally equivalent) if they have the same observable behavior, i.e. an outsider cannot distinguish them. Interesting properties of programs can be expressed using the notion of contextual equivalence. For example, to prove that a program does not leak a secret, such as the secret key used by an ATM to communicate with the bank, it is sufficient to prove that if we change the secret, the observable behavior will not change [13, 3, 14]: whatever experiment a customer makes with the ATM, he or she cannot guess information about the secret key by observing the reaction of the ATM. Another example is to specify functional properties by contextual equivalence. For example, if sorted is a function which checks that a list is sorted and sort is a function which sorts a list, then, for all list l, you want the expression $\mathsf{sorted}(\mathsf{sort}(l))$ to be contextually equivalent to the expression true. Finally, in the context of parameterized verification, contextual equivalence allows the verification for all instantiations of the parameter to be reduced to the verification for a finite number of instantiations (See e.g. [5] where logical relations are one of the essential ingredients).

⋆ Partially supported by the RNTL project Prouvé, the ACI Sécurité Informatique Rossignol, the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions", and the ACI Cryptologie "PSI-Robuste".

⋆⋆ Partially supported by the Polish KBN grant No. 4 T11C 042 25 and by the European Community Research Training Network *Games*. This work was performed in part during the author's stay at LSV.

⋆⋆⋆ This work was mainly done when the author was a PhD student under an MENRT grant on ACI Cryptologie funding, École Doctorale Sciences Pratiques (Cachan).

Logical relations. While contextual equivalence is difficult to prove directly because of the universal quantification over contexts, logical relations [11, 6] are powerful tools that allow us to deduce contextual equivalence in typed λ-calculi. With the aid of the so-called Basic Lemma, one can easily prove that logical relations are sound w.r.t. contextual equivalence. However, completeness of logical relations is much more difficult to achieve: usually we can only show the completeness of logical relations for types up to first order.

The computational λ-calculus [8] has proved useful to define various notions of computations on top of the λ-calculus, using monadic types. Logical relations for monadic types can be derived by the construction defined in [2] where soundness of logical relations is guaranteed. However, monadic types introduce new difficulties. In particular, contextual equivalence becomes subtler due to the different semantics of different monads: equivalent programs in one monad are not necessarily equivalent in another! This accordingly makes completeness of logical relations more difficult to achieve in the computational λ-calculus. In particular the usual proofs of completeness up to first order do not go through.

Contributions. We propose in this paper a notion of contextual equivalence for the computational λ-calculus. Logical relations for this language are defined according to the general derivation in [2]. We then explore the completeness and we prove that for the partial computation monad, the exception monad and the state transformer monad, logical relations are still complete up to first-order types. In the case of the non-determinism monad, we need to restrict ourselves to a subset of first-order types.

Not like previous work on using logical relations to study contextual equivalence in models with computational effects [12, 10, 9], most of which focus on computations with local states, our work in this paper is based on a more general framework for describing computations, namely the computational λ-calculus. In particular, very different forms of computations like non-determinism are studied, not just those for local states.

Note that all proofs that are omitted in this short paper, can be found in the full version [4].

2 Logical relations for the simply typed λ-calculus

Let $\lambda^{\to}$ be a simple version of typed λ-calculus with only base types b (booleans, integers, etc.) and function types $\tau \to \tau'$. Terms consist of variables, constants, abstractions and applications. Notations and typing rules are as usual. We consider the set theoretical semantics of $\lambda^{\to}$. 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, e.g., when t is a closed term. When given a value $a \in [\![\tau]\!]$, we say that it is *definable* if and only if there exists a closed term t such that $\vdash t : \tau$ is derivable and $a = [\![t]\!]$.

Let $\mathbf{Obs}$ be a subset of base types, called *observation types*, such as booleans, integers, etc. A *context* $\mathbb{C}$ is a term such that $x : \tau \vdash \mathbb{C} : o$ is derivable, where o is an observation type. We spell the standard notion of *contextual equivalence* in a

denotational setting: 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 $x : \tau \vdash \mathbb{C} : o$ ($o \in \mathbf{Obs}$) 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* whenever $[\![t_1]\!] \approx_\tau [\![t_2]\!]$. We also define a relation $\sim_\tau$: for every pair of values $a_1, a_2 \in [\![\tau]\!]$, $a_1 \sim_\tau a_2$ if and only if a_1, a_2 are definable and $a_1 \approx_\tau a_2$.

Essentially, a (binary) *logical relation* [6] 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 $f_1, f_2 \in [\![\tau \to \tau']\!]$,

$$f_1 \; \mathcal{R}_{\tau\to\tau'} \; f_2 \iff \forall a_1, a_2 \in [\![\tau]\!] \; . \; a_1 \; \mathcal{R}_\tau \; a_2 \implies f_1(a_1) \; \mathcal{R}_{\tau'} \; f_2(a_2)$$

There is no constraint on relations at base types. In $\lambda^\to$, 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.

A so-called *Basic Lemma* comes along with logical relations since Plotkin's work [11]. It states that if $\Gamma \vdash t : \tau$ is derivable, ρ_1, ρ_2 are two related Γ-environments, and every constant is related to itself, 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 is crucial for proving various properties using logical relations [6]. In the case of establishing contextual equivalence, it implies that, for every logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that $\mathcal{R}_o$ is the equality for every observation type o, logically related values are necessarily contextually equivalent, i.e., $\mathcal{R}_\tau \subseteq \approx_\tau$ for any type τ.

Completeness states the inverse: 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. Completeness for logical relations is hard to achieve, even in a simple version of λ-calculus like $\lambda^\to$. 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). The following proposition states the completeness of logical relations in $\lambda^\to$, for types up to first order:

Proposition 1. *There exists a logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ for $\lambda^\to$, 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, $t_1 \approx_\tau t_2 \implies [\![t_1]\!] \; \mathcal{R}_\tau \; [\![t_2]\!]$.*

3 Logical relations for the computational λ-calculus

3.1 The computational λ-calculus λ_{Comp}

Moggi's computational λ-calculus has proved a nice framework for expressing various forms of side effects (exceptions, non-determinism, etc.) [8]. The computational λ-calculus, denoted by λ_{Comp}, extends $\lambda^\to$ with a unary type constructor T: $\mathsf{T}\tau$ denotes the type of computations which return values of type τ. It also introduces two extra term constructs $\mathtt{val}(t)$ and $\mathtt{let}\ x \Leftarrow t\ \mathtt{in}\ t'$, representing respectively the trivial computation and the sequential computation, with the typing rules:

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

Moggi also builds a categorical model for the computational λ-calculus, using the notion of monads [8]. We shall focus on Moggi's monads defined over the category $\mathcal{S}et$ of sets and functions. For instance, the non-determinism monad is defined by $[\![\mathsf{T}\tau]\!] = \mathbb{P}_{\mathrm{fin}}([\![\tau]\!])$, with

$$[\![\mathtt{val}(t)]\!]\rho = \{[\![t]\!]\rho\},$$
$$[\![\mathtt{let}\ x \Leftarrow t_1\ \mathtt{in}\ t_2]\!]\rho = \bigcup_{a \in [\![t_1]\!]\rho} [\![t_2]\!]\rho[x := a].$$

In the rest of this paper, we shall restrict ourselves to four concrete monads: partial computations, exceptions, state transformers and non-determinism. Definitions of these monads can be found in [8, 4]. We write λ_{Comp}^{PESN} for the particular version of λ_{Comp} where the monad is restricted to be one of these four monads.

The computational λ-calculus is strongly normalizing [1]. In fact, every term of a monadic type can be written in the following βc-normal-η-long form, w.r.t. the βc-reduction rules and η-equalities in [1] (see the proof in the full version [4]):

$$\mathtt{let}\ x_1 \Leftarrow d_1 u_{11} \cdots u_{1k_1}\ \mathtt{in} \cdots \mathtt{let}\ x_n \Leftarrow d_n u_{n1} \cdots u_{nk_n}\ \mathtt{in}\ \mathtt{val}(u),$$

where $n = 0, 1, 2, \ldots$, every d_i ($1 \leq i \leq n$) is either a constant or a variable, u and u_{ij} ($1 \leq i \leq n, 1 \leq j \leq k_j$) are all βc-normal terms or βc-normal-η-long terms (of monadic types).

As argued in [3], the standard notion of contextual equivalence does not fit in the setting of the computational λ-calculus. In order to define contextual equivalence for λ_{Comp}, we have to consider contexts $\mathbb{C}$ of type $\mathsf{T}o$ (o is an observation 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, because the typing rule for the `let` construct only allows us to use computations to build other computations, never values. Taking this into account, we get the following definition:

Definition 1 (Contextual equivalence for λ_{Comp}). *In λ_{Comp}, 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 if and only if $[\![t_1]\!] \approx_\tau [\![t_2]\!]$. We use the same notation*

3.2 Logical relations for λ_{Comp}

A uniform framework for defining logical relations relies on the categorical notion of subscones [7], and a natural extension of logical relations able to deal with monadic types was introduced in [2]. The construction consists in lifting the CCC structure and the strong monad from the categorical model to the subscone. We reformulate this construction in the category $\mathcal{S}et$. 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. $a\ R\ b \Rightarrow f(a)\ R'\ g(b)$.

The lifting of the CCC structure gives rise to the standard logical relations given in Section 2 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 $(c_1, c_2) \in \mathcal{R}_{\mathsf{T}\tau} \iff (c_1, c_2) \in \tilde{T}(\mathcal{R}_\tau)$. This construction guarantees that Basic Lemma always holds provided that every constant is related to itself [2]. A list of instantiations of the above definition in concrete monads is also given in [2]. For instance, the logical relation for the non-determinism monad is defined by:

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

Definitions of logical relations for other monads in λ_{Comp}^{PESN} can be found in [4].

We restrict our attention to 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.

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

3.3 Toward a proof on completeness of logical relations for λ_{Comp}

Completeness of logical relations for λ_{Comp} is much subtler than in $\lambda^{\to}$ due to the introduction of monadic types. We were expecting to find a general proof following the general construction defined in [2]. However, this turns out extremely difficult although it might not be impossible with certain restrictions, on types for example. The difficulty arises mainly from the different semantics for different forms of computations, which actually do not ensure that equivalent programs in one monad are necessarily equivalent in another. Consider two programs in λ_{Comp}: $\texttt{let}\; x \Leftarrow t_1 \;\texttt{in let}\; y \Leftarrow t_2 \;\texttt{in}\; \texttt{val}(x)$ and $\texttt{let}\; y \Leftarrow t_2 \;\texttt{in let}\; x \Leftarrow t_1 \;\texttt{in}\; \texttt{val}(x)$, where both t_1 and t_2 are closed term. We can conclude that they are equivalent in the non-determinism monad — they return the same set of possible results of t_1, no matter what results t_2 produces, but this is not the case in, e.g., the exception monad when t_1 and t_2 throw different exceptions.

Being with such an obstacle, we shall switch our effort to case studies in Section 4 and we explore the completeness of logical relations for a list of common monads, precisely, all the monads in λ_{Comp}^{PESN}. But, let us sketch out here a general structure for proving completeness of logical relations in λ_{Comp}. In particular, our study is still restricted to first-order types, which, in λ_{Comp}, are defined by the grammar: $\tau^1 ::= b \mid \mathsf{T}\tau^1 \mid b \to \tau^1$, where b ranges over the set of base types.

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, $\sim_\tau \subseteq \mathcal{R}_\tau$, where $\sim_\tau$ is the relation defined in Section 2. As in the proof of 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 the completeness for an arbitrary monad T?

Note that we should also check that the logical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$, induced by $\mathcal{R}_b = \sim_b$, is observational, i.e., a partial equality on $\mathsf{T}o$, for any observable type o.

Consider any pair $(a, b) \in \mathcal{R}_{\mathsf{T}o} = \tilde{T}(\mathcal{R}_o)$. By definition of the lifted monad $\tilde{T}$, there exists a computation $c \in T\mathcal{R}_o$ such that $a = T\pi_1(c)$ and $b = T\pi_2(c)$. But $\mathcal{R}_o = \sim_o \subseteq \mathrm{id}_{[\![o]\!]}$, hence the two projections $\pi_1, \pi_2 : \mathcal{R}_o \to [\![o]\!]$ are the same function, $\pi_1 = \pi_2$, and consequently $a = T\pi_1(c) = T\pi_2(c) = b$. This proves that $\mathcal{R}_{\mathsf{T}o}$ is a partial equality.

As usual, the proof of completeness would go by induction over τ, to show $\sim_\tau \subseteq \mathcal{R}_\tau$ for each first-order type τ. Cases $\tau = b$ and $\tau = b \to \tau'$ go identically as in $\lambda^{\to}$. The only difficult case is $\tau = \mathsf{T}\tau'$, i.e., $(\star)$ $\sim_\tau \subseteq \mathcal{R}_\tau \implies \sim_{\mathsf{T}\tau} \subseteq \mathcal{R}_{\mathsf{T}\tau}$. We did not find any general way to show $(\star)$ for an arbitrary monad. Instead, in the next section we prove it by cases, for all the monads in λ^{PESN}_{Comp} except the non-determinism monad. The non-determinism monad is an exceptional case where we do not have completeness for all first-order types but a subset of them. This will be studied separately in Section 4.3.

At the heart of the difficulty of showing $(\star)$ we find an issue of definability at monadic types in the set-theoretical model. We write def_τ for the subset of definable elements in $[\![\tau]\!]$, and we eventually show that the relation between $\mathsf{def}_{\mathsf{T}\tau}$ and def_τ can be shortly spelled-out $(\star\star)$: $\mathsf{def}_{\mathsf{T}\tau} \subseteq T\mathsf{def}_\tau$, for all the monads we consider in this paper. This is a crucial argument for proving completeness of logical relations for monadic types, but to show $(\star\star)$, we need different proofs for different monads. This is detailed in Section 4.1.

4 Completeness of logical relations for monadic types

4.1 Definability in the set-theoretical model of λ^{PESN}_{Comp}

As we have seen in $\lambda^{\to}$, definability is involved largely in the proof of completeness of logical relations (for first-order types). This is also the case in λ_{Comp} and it apparently needs more concern due to the introduction of monadic types.

Despite we did not find a general proof for $(\star\star)$ it does hold for all the concrete monads in λ^{PESN}_{Comp}. To state it formally, let us first define a predicate $\mathcal{P}_\tau$ on elements of $[\![\tau]\!]$, by induction on types: $\mathcal{P}_b = \mathsf{def}_b$, for every base type b; $\mathcal{P}_{\mathsf{T}\tau} = T(\mathsf{def}_\tau \cap \mathcal{P}_\tau)$; $\mathcal{P}_{\tau\to\tau'} = \{f \in \mathcal{P}_{\tau\to\tau'} \mid \forall a \in \mathsf{def}_\tau, f(a) \in \mathcal{P}_{\tau'}\}$. We say that a constant c (of type τ) is logical if and only if τ is a base type or $[\![c]\!] \in P_\tau$. We then require that λ^{PESCN}_{Comp} contains only logical constants. Note that this restriction is valid because the predicates $P_{T\tau}$ and $P_{\tau\to\tau'}$ depend only on definability at type τ. Some typical logical constants for monads in λ^{PESN}_{Comp} are as follows:

- Partial computation: a constant Ω_τ of type $\mathsf{T}\tau$, for every τ. Ω_τ denotes the non-termination, so $[\![\Omega_\tau]\!] = \bot$.
- Exception: a constant $\mathtt{raise}^e_\tau$ of type $\mathsf{T}\tau$ for every type τ and every exception $e \in E$. $\mathtt{raise}^e_\tau$ does nothing but raises the exception e, so $[\![\mathtt{raise}^e_\tau]\!] = e$.
- State transformer: a constant $\mathtt{update}_s$ of type $\mathsf{T}\mathsf{unit}$ for every state $s \in St$, where unit is the base type which contains only a dummy value $*$. $\mathtt{update}_s$ simply changes the current state to s, so for any $s' \in St$, $[\![\mathtt{update}_s]\!](s') = (*, s)$.
- Non-determinism: a constant $+_\tau$ of type $\tau \to \tau \to \mathsf{T}\tau$ for every non-monadic type τ. $+_\tau$ takes two arguments and returns randomly one of them — it introduces the non-determinism, so for any $a_1, a_2 \in [\![\tau]\!]$, $[\![+_\tau]\!](a_1, a_2) = \{a_1, a_2\}$.

We assume in the rest of this paper that the above constants are present in λ_{Comp}^{PESN}. It is clear that each of these constants is related to itself.

Note that $\mathcal{P}_\tau$ being a predicate on elements of $[\![\tau]\!]$ is equivalent to say that $\mathcal{P}_\tau$ can be seen as subset of $[\![\tau]\!]$, but in the case of monadic types, $\mathcal{P}_{\mathsf{T}\tau}$ (i.e., $T(\mathsf{def}_\tau \cap \mathcal{P}_\tau)$) is not necessary a subset of $[\![\mathsf{T}\tau]\!]$ (i.e., $T[\![\tau]\!]$). Fortunately, we prove that all the monads in λ_{Comp}^{PESN} preserves inclusions, which ensures that the predicate $\mathcal{P}$ is well-defined:

Proposition 2. *All the monads in λ_{Comp}^{PESN} preserve inclusions: $A \subseteq B \Rightarrow TA \subseteq TB$.*

Introducing such a constraint on constants is mainly for proving ($\star\star$). In fact, we can prove that the denotation of every closed βc-normal-η-long computation term t (of type $\mathsf{T}\tau$), in λ_{Comp}^{PESN} with logical constants, is an element of $T\mathsf{def}_\tau$, i.e., $[\![t]\!] \subseteq T\mathsf{def}_\tau$ (see [4] for details).

Proposition 3. $\mathsf{def}_{\mathsf{T}\tau} \subseteq T\mathsf{def}_\tau$ *holds in the set-theoretical model of λ_{Comp}^{PESN} with logical constants.*

4.2 Completeness of logical relations in λ_{Comp}^{PES} for first-order types

We prove ($\star$) in this section for the partial computation monad, the exception monad, the state monad and the continuation monad. We write λ_{Comp}^{PES} for λ_{Comp} where the monad is restricted to one of these four monads.

Proofs depend typically on the particular semantics of every form of computation, but a common technique is used frequently: given two definable but non-related elements of $[\![\mathsf{T}\tau]\!]$, one can find a context to distinguish the programs (of type $\mathsf{T}\tau$) that define the two given elements, and such a context is usually built based on another context that can distinguish programs of type τ.

Theorem 2. *In λ_{Comp}^{PES}, if all constants are logical and in particular, if the constant $\mathtt{update}_s$ is present for the state transformer monad, then logical relations 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 Completeness of logical relations for the non-determinism monad

The non-determinism monad is an interesting case: the completeness of logical relations for this monad does not hold for all first-order types! To state it, consider the following two programs of a first-order type that break the completeness of logical relations:

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

Recall the logical constant $+_\tau$ of type $\tau \to \tau \to \mathsf{T}\tau$: $[\![+_\tau]\!](a_1, a_2) = \{a_1, a_2\}$ for every $a_1, a_2 \in [\![\tau]\!]$. The two programs are contextually equivalent: what contexts can do is to apply the functions to some arguments and observe the results. But no matter how many time we apply these two functions, we always get the same set of possible values ($\{\mathtt{true}, \mathtt{false}\}$), so there is no way to distinguish them with a context.

Recall the logical relation for non-determinism monad in Section 3.2. Clearly the denotations of the above two programs are not related by that relation because the function $[\![\lambda x.\mathtt{val}(\mathtt{true})]\!]$ from the second program is not related to the function in the first.

However, 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, $\mathcal{P}(\mathtt{test}_b)$ holds), logical relations for the non-determinism monad are then complete for a set of *weak first-order types*: $\tau^1_{\mathrm{w}} ::= b \mid \mathsf{T}b \mid b \to \tau^1_{\mathrm{w}}$. Compared to all types up to first order, weak first-order types do not contain monadic types of functions, so it immediately excludes the two programs in the above counterexample.

Theorem 3. *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 $(\mathcal{R}_\tau)_{\tau\ \mathrm{type}}$ such that for any closed terms t_1, t_2 of a weak first-order type τ^1_w, if $t_1 \approx_{\tau^1_w} t_2$, then $[\![t_1]\!]\ \mathcal{R}_{\tau^1_w}\ [\![t_2]\!]$.*

References

1. P. N. Benton, G. M. Bierman, and V. C. V. de Paiva. Computational types from a logical perspective. *J. Functional Programming*, 8(2):177–193, 1998.
2. J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Proceedings of CSL'2002*, volume 2471 of *LNCS*, pages 553–568. Springer, 2002.
3. J. Goubault-Larrecq, S. Lasota, D. Nowak, and Y. Zhang. Complete lax logical relations for cryptographic lambda-calculi. In *Proceedings of CSL'2004*, volume 3210 of *LNCS*, pages 400–414. Springer, 2004.
4. S. Lasota, D. Nowak, and Y. Zhang. On completeness of logical relations for monadic types. Research Report cs.LO/0612106, arXiv, 2006.
5. R. Lazić and D. Nowak. A unifying approach to data-independence. In *Proceedings of CONCUR'2000*, volume 1877 of *LNCS*, pages 581–595. Springer, 2000.
6. J. C. Mitchell. *Foundations of Programming Languages*. MIT Press, 1996.
7. J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In *Proceedings of CSL'1992*, volume 702 of *LNCS*, pages 352–378. Springer, 1993.
8. E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
9. P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, 1995.
10. A. Pitts and I. Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998.
11. G. D. Plotkin. Lambda-definability in the full type hierarchy. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980.
12. K. Sieber. Full abstraction for the second order subset of an algol-like language. *Theoretical Computer Science*, 168(1):155–212, 1996.
13. E. Sumii and B. C. Pierce. Logical relations for encryption. *J. Computer Security*, 11(4):521–554, 2003.
14. Y. Zhang. *Cryptographic logical relations*. Ph. d. dissertation, ENS Cachan, France, 2005.