On Completeness of Logical Relations for Monadic Types*

Sławomir Lasota1,**, David Nowak2, and Yu Zhang3,***

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 [13,14]: 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 sorted(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 \[1,6\] are powerful tools that allow us to deduce contextual equivalence in typed $\lambda$-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 $\lambda$-calculus \[8\] has proved useful to define various notions of computations on top of the $\lambda$-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 $\lambda$-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 $\lambda$-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 $\lambda$-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 $\lambda$-Calculus

Let $\lambda \to$ be a simple version of typed $\lambda$-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 $\Gamma$-environment $\rho$ is a map such that, for every $x : \tau$ in $\Gamma$, $\rho(x)$ 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 $\Gamma$-environment $\rho$, is given as usual by an element $[t]\rho$ of $[\tau]$. We write $[t]$ instead of $[t]\rho$ when $\rho$ 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 $\text{Obs}$ be a subset of base types, called observation types, such as booleans, integers, etc. A context $C$ is a term such that $x : \tau \vdash C : o$ is derivable, where $o$ is an observation type. Wespell the standard notion of contextual equivalence in a denotational setting: two elements $a_1$ and $a_2$ of $[\tau]$, are contextually equivalent (written as...