

Descriptive and Relative Completeness of Logics for Higher-Order Functions*

Kohei Honda¹, Martin Berger¹, and Nobuko Yoshida²

¹ Department of Computer Science, Queen Mary, University of London

² Department of Computing, Imperial College London

Abstract. This paper establishes a strong completeness property of compositional program logics for pure and imperative higher-order functions introduced in [18, 16, 17, 19, 3]. This property, called *descriptive completeness*, says that for each program there is an assertion fully describing the program's behaviour up to the standard observational semantics. This formula is inductively calculable from the program text alone. As a consequence we obtain the first relative completeness result for compositional logics of pure and imperative call-by-value higher-order functions in the full type hierarchy.

1 Introduction

Program logics such as Hoare logic are a means to *describe* abstract behaviours of programs as logical assertions; to *verify* that a given program satisfies a specified property; and to *define* axiomatic semantics in the sense that the assertions assign meaning to a program with respect to its observable properties. Because of this strong match with observable and operational semantics of programs in a simple and intuitive manner, many engineering activities ranging from static analyses to program testing increasingly use program logics as their theoretical foundation.

For describing properties of first-order imperative programs, Hoare logic uses a pair of assertions in number theory. For example, in the partial correctness judgement $\{x = i\}x := x + 1\{x = i + 1\}$, the pair of assertions $x = i$ and $x = i + 1$ describes a property of the program $x := x + 1$ by saying: *whatever the initial content of x would be, if this program terminates, then the final content of x is the increment of its initial one*. Here a *property* is a subset of programs taken modulo an observational congruence: for example, in *while* programs, we consider programs up to partial functions on store they represent. Since the collection of all properties is uncountable, no standard logical language can represent all properties of any non-trivial programming language. Then what classes of properties should a program logic represent and prove?

In this paper, we focus on *descriptive completeness*, a strong completeness property, which is about representability of behaviour *as a canonical formula*: given a program P , we can always find a unique assertion pair in Hoare logic which represents (pinpoints) P 's behaviour. For partial correctness, the best assertion pair for P describes all partial functions equal to or less defined than P . For example, the pair " $x = i$ " and " $x = i + 1$ "

* Work is partially supported by EPSRC GR/R03075/01, GR/T04236/01, GR/S55538/01, GR/T04724/01, GR/T03208/01 and IST-2005-015905 MOBIUS.

are also satisfied by a diverging program. Dually for total correctness. A related concept are the *characteristic formulae* of Hennessy-Milner logics, which precisely characterise a CCS process up to bisimilarity [13, 31, 32]. We shift this notion from a process logic to a program logic, establishing descriptive completeness of Hoare logics for pure and imperative higher-order functions introduced in [18, 16, 19, 3].

In first-order Hoare logic, a program defines a partial function from states to states, so that the existence of characteristic formulae is not hard to establish. When we move to higher-order programs, a logic needs to describe how a program *transforms behaviour*. For example $\lambda x^{\text{Nat} \Rightarrow (\alpha \Rightarrow \beta)}.x1$ is a function which receives a function and returns another function. The logics for higher-order functions and their imperative extensions [16, 19, 3, 18] involve direct description of such applicative behaviour. Due to complexity of the underlying semantic universe, it is not immediately obvious if a single pair of formulae can fully describe the behaviour of an arbitrary higher-order program. In the present paper we *construct* a characteristic formula of a program compositionally and algorithmically, following its syntactic structure, and inductively verify that the derived formula has the required properties. The induced algorithm is implemented as a prototype (1,250 LOC in Ocaml) [2]. The size of the resulting formula is asymptotically almost linear to the size of a program under a certain condition.

The generated characteristic assertions clarify the relationship between total and partial correctness for higher-order objects, following early observations [29, 28], but in the context of concrete assertion methods and proof rules. We use the duality between total and partial correctness [28] to derive descriptive completeness for partial correctness from its total variant. A total correctness property denotes an upward closed set of semantic points, representing liveness, while a partial correctness formula stands for a downward closed set, representing safety [28, 26, 23]. This duality subsumes the corresponding notions in the original Hoare logic, and offers a key insight into the nature of assertions for higher-order objects and their derivation. Finally, relative completeness [6] of proof rules is an immediate consequence of descriptive completeness. To our knowledge this work is the first to obtain descriptive and relative completeness in Hoare logics for (imperative) higher-order functions in the full type hierarchy.

In the remainder, Section 2 establishes descriptive and relative completeness for the logic of call-by-value PCF. Section 3 discusses the corresponding results for an imperative extension. Section 4 gives comparisons with related work. Section 5 concludes with further topics. All proofs are omitted, relegated to the long version [1].

2 Descriptive Completeness for PCFv

Call-by-value PCF. The syntax of PCFv is standard [27], and is briefly reviewed below (we can easily treat, but omit, other standard types such as sums and products [18]).

$$\begin{aligned} \alpha, \beta, \dots &::= \text{Bool} \mid \text{Nat} \mid \alpha \Rightarrow \beta & V, W, \dots &::= x^\alpha \mid c \mid \lambda x^\alpha. M \mid \mu f^{\alpha \Rightarrow \beta}. \lambda x^\alpha. M \\ M, N, \dots &::= V \mid \text{op}(\vec{M}) \mid MN \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \end{aligned}$$

We use numerals (0,1,2,...) and booleans (**t** and **f**) as constants (*c* above) and standard first-order operations ($\text{op}(\vec{M})$ where \vec{M} denotes a vector). V, V', \dots denote values. The typing is standard; henceforth we only consider well-typed programs. A *basis* (Γ, Δ, \dots)