Proof Abstraction for Imperative Languages*

William L. Harrison

Dept. of Computer Science, University of Missouri,
Columbia, Missouri, USA

Abstract. Modularity in programming language semantics derives from abstracting over the structure of underlying denotations, yielding semantic descriptions that are more abstract and reusable. One such semantic framework is Liang's modular monadic semantics in which the underlying semantic structure is encapsulated with a monad. Such abstraction can be at odds with program verification, however, because program specifications require access to the (deliberately) hidden semantic representation. The techniques for reasoning about modular monadic definitions of imperative programs introduced here overcome this barrier. And, just like program definitions in modular monadic semantics, our program specifications and proofs are representation-independent and hold for whole classes of monads, thereby yielding proofs of great generality.

Keywords: Monads, Monad Transformers, Language Semantics, Program Specification and Verification.

1 Introduction

Modular monadic semantics (MMS) provides a powerful abstraction principle for denotational definitions via the use of monads and monad transformers [13,2,21] and MMS supports a modular, "mix and match" approach to semantic definition. MMS has been successfully applied to a wide variety of programming languages as well as to language compilers [8,6].

What is not well-recognized is the impact that the semantic factorization by monad transformers in MMS has on program specification and verification. Modularity comes with a price! The monad parameter to an MMS definition is a "black box" (i.e., its precise type structure is unknown) and must remain so if program abstraction is to be preserved. Yet, this makes reasoning with MMS language definitions using standard techniques frequently impossible. How does one reason about MMS specifications without sacrificing modularity and reusability? Furthermore, is there a notion of *proof* abstraction for MMS akin to its notion of *program* abstraction? This paper provides answers in the affirmative to these questions for imperative languages.

* This research supported in part by subcontract GPACS0016, System Information Assurance II, through OGI/Oregon Health & Sciences University.

N. Kobayashi (Ed.): APLAS 2006, LNCS 4279, pp. 97–113, 2006.

This paper presents a novel form of specification for reasoning about MMS definitions called *observational program specification* (OPS), as well as related proof techniques useful for proving such specifications. To reason about MMS definitions (which are parameterized by monads), it is necessary to parameterize the specifications themselves by monads as well. This is precisely what OPS does by lifting predicates to the computational level, and we refer to such lifted predicates as *observations*. Both MMS definitions and OPS specifications are parameterized by a monad that hides underlying denotational structure, thereby allowing greater generality in both programs and proofs alike. And just as MMS provides a notion of program abstraction, OPS provides a notion of *proof* abstraction. Observational program specifications and proofs are representation-independent, holding for whole classes of monads, thereby yielding proofs of great generality.

The methodology pursued here is as follows. Axioms characterizing algebraically the behavior of state monads are defined, and it is demonstrated that these axioms are preserved under monad transformer application. Then, a denotational semantics for the simple imperative language with loops is given in terms of state monads. Using OPS and "observation" computations, Hoare's classic programming logic [9] for this language is embedded into its own state-monadic semantics. Furthermore, it is demonstrated that the inference rules of this logic are derivable from the embedding, relying only on the state monad axioms and facts about observations. This provides a notion of proof abstraction for the simple imperative language because proofs in Hoare logic can now be lifted to any monad with state regardless of other effects it encapsulates!

This paper has the following structure. Section 2 motivates OPS, and Section 3 outlines background material necessary to understand this paper, including overviews of monads and monad transformers. In Section 4, the axiomatization of state monads and their preservation properties with respect to monad transformer application are stated and proved. In Section 5, the notion of observations is made precise. Section 6 presents the embedding of Hoare logic, and also the proof of soundness of this embedding. Section 7 compares the present work with related research. Conclusions and future work are outlined in Section 8.

2 Introducing Observational Specifications

As an example, consider the correctness of an imperative construct $\mathsf{p!}$ defined in a monad with a state Sto. Generally [26,15], a partial correctness specification of an imperative feature like this would take the form of a relation $\Re$ between input and output states σ_0 and σ_1, so that $\sigma_0 \,\Re\, \sigma_1$ means that the state σ_1 may result from the execution of $\mathsf{p!}$ in σ_0. If $\mathsf{p!}$ were defined in the single state monad $\mathsf{St}\, a = Sto \to a \times Sto$, then the correctness of $\mathsf{p!}$ would be written:

$$\forall \sigma_0 : Sto.\ \ \sigma_0 \,\Re\, (\pi_2(\mathsf{p!}\ \sigma_0)) \tag{1}$$

where π_2 is the second projection function $\lambda(-, x).x$. However, if $\mathsf{p!}$ were reinterpreted in the "Environment+State" monad $\mathsf{EnvSt}\, a = Env \to Sto \to a \times Sto$,