A Monad-Based Modeling and Verification Toolbox with Application to Security Protocols*

Christoph Sprenger and David Basin

Department of Computer Science, ETH Zurich, Switzerland
{sprenger,basin}@inf.ethz.ch

Abstract. We present an advanced modeling and verification toolbox for functional programs with state and exceptions. The toolbox integrates an extensible, monad-based, component model, a monad-based Hoare logic and weakest precondition calculus, and proof systems for temporal logic and bisimilarity. It is implemented in Isabelle/HOL using shallow embeddings and incorporates as much modeling and reasoning power as possible from Isabelle/HOL. We have validated the toolbox’s usefulness in a substantial security protocol verification project.

1 Introduction

The choice of a specification formalism with supporting verification methods and tools is critical to the success of substantial verification projects. In order to obtain manageable proofs, models and their properties must be formulated at an appropriate level of abstraction and verification methods and tools must be available for reasoning at that abstraction level. In this paper, we present a comprehensive formalism providing such abstractions for the specification and verification of functional programs with non-pure features, namely state and exceptions. It consists of the following elements: (1) a modeling language capturing the computational structure of the systems under study, (2) a notion of component for structuring models, (3) a notion of component abstraction, including a proof method to establish an abstraction relation between two components, and (4) property specification languages and proof methods. We have implemented this toolbox in Isabelle/HOL [18]. Our aim is to provide a set of practically useful tools rather than to develop meta-theoretical studies about programming languages and logics. Therefore, we work with shallow embeddings and exploit Isabelle/HOL’s expressive specification language (for 1–2) and powerful reasoning tools (for 3–4) as much as possible. We now discuss (1)–(4) in turn.

Ad 1. Modeling state and exception handling directly in HOL results in models that are cluttered with additional state parameters and error-handling branches. This situation calls for an additional layer of abstraction, which can be elegantly modeled using monads [15]. In our case, the monad of choice is a deterministic state-exception monad, which extends HOL with operations for sequential composition, state manipulation, and exception handling. All other control structures, such as pattern matching, if-then-else

* This work was partially supported by the Zurich Information Security Center. It represents the views of the authors.
and function definitions, are borrowed directly from Isabelle/HOL. We would like to stress that most of our method and tools can be easily adapted to other monads.

Ad 2. We take a simple view of components as modules encapsulating a state with interface functions that manipulate this state. We represent the state in an object-oriented style as an extensible record and component composition as record extension and function composition. Using extensible records allows the extended component to automatically inherit properties proved for the base component.

Ad 3. We compare (abstract or refine) components using bisimilarity. Two components are bisimilar if all pairs of equally named interface functions are bisimilar. To show this, we have developed a compositional proof system to establish the bisimilarity of programs. Its judgments and rules resemble those of Hoare logic except that pre- and post-conditions are relations between pairs of states of two programs.

Ad 4. In order to state and prove properties about components, we have embedded several logics in Isabelle/HOL: a weakest pre-condition (WP) calculus, a Hoare logic, and linear-time temporal logic (LTL). All these logics rely on HOL’s set theory as their underlying assertion language. The WP calculus is derived from Pitts’ evaluation logic [20] and provides the basis for the definition of the Hoare logic. Both of these are tailored to the state-exception monad, whereas the temporal logic is interpreted over standard transition systems. The benefit of embedding these logics in Isabelle/HOL is that this enables the use of the corresponding, well-established proof methods. We reduce the verification of temporal properties to pre-/post-condition and assertional reasoning using standard proof rules [14]. The resulting Hoare triples are proved using a combination of Hoare logic and the underlying WP calculus. While Hoare logic provides the full flexibility of interactive proofs, we may unfold Hoare triples at any point during a proof and invoke the WP calculus, using Isabelle’s efficient simplifier to automatically reduce the resulting weakest pre-conditions into simpler HOL assertions.

Our contribution is two-fold. First, we show how to leverage the rich modeling and reasoning infrastructure provided by a theorem prover like Isabelle/HOL for semantic embeddings of various modeling, specification, and verification concepts. This allows us to model, specify and reason at an adequate level of abstraction. We adapt and integrate well-known techniques with less standard and new ones to build a unique, comprehensive, and modular modeling and verification toolbox. Second, we provide a ready-to-use verification toolbox with a wide scope of application. This toolbox has proved its effectiveness in a substantial cryptographic protocol verification project [21]. For a rough idea of the size of the theories involved, our development spans more than 22k lines of Isabelle/HOL sources, organized into more than 50 theories. The toolbox accounts for about 20% of these figures. In particular, the combination of Hoare logic and the WP calculus offers a good degree of proof automation, which was crucial to the successful completion of the sizable verification effort involved. We use examples from our project [21] to illustrate the application of the concepts presented in this paper.

2 Background

Isabelle/HOL notation. In Isabelle/HOL, \( t :: T \) denotes a term \( t \) of type \( T \). The expression \( c \ x \equiv t \) defines the constant \( c \) with the parameter \( x \) as the term \( t \). Definitions