Contracting and Proving Classes with Models
Extended Abstract

Bernd Schoeller

bernd.schoeller@inf.ethz.ch
Chair of Software Engineering, ETH Zurich, Switzerland

Introduction

Functional specification and verification often requires specialized mathematicians to do the work. Software development and proving correctness are regarded as separate activities.

We want developers to do the specification and verification. One of the promising approaches for letting developers create specifications is the "Design by Contract" methodology [7] as present in the Eiffel language [8]. Contracts document the developer's intend during an early phase of development and make assumptions on the existing system explicit. They establish a functional specification without leaving the notation of the programming language.

The goal of this thesis is to integrate formal methods and Design by Contract into a common proof technology that does not require developers to learn a new notation and at the same time improves the quality of the software by using fully automated proofs. We try to achieve this goal by reducing the conceptual gap between Eiffel and the prover through the use of models and model-based contracts.

Models and model-based contracts for verification

A model is a mathematical structure that captures the state of a computation on an abstract level [6]. We want to enable the developer to formulate contracts using models. To do this, we have to integrate a notation into Eiffel to express and reason about theses mathematical structures.

We can do this without touching the Eiffel language: we have developed a library that captures typed set theory on finite sets. This library provides standard implementations of immutable data structures like sets, relations or sequences. The set theory is derived from the notation used in the B method [1]. B has proved itself to be well suited for complex formalizations.

Contracts that are described using this library can be directly mapped into the mathematical language of a theorem prover. We call such contracts "model-based contracts". They allow us to hide the provers notation from the user.

The result is a major improvement in the expressiveness of contracts. Also, it reduces the conceptual gap between predicates in theorem provers and boolean functions in Eiffel.

A major part of our work is to use these model-based contracts for functional verification. As theorem prover, we use the Boogie tool that is developed for Spec# at Microsoft [2]. Boogie itself uses Simplify [5] as its proof engine.

Boogie defines an intermediate language called BoogiePL [4]. BoogiePL is an imperative language targeted for verification. We translate Eiffel code and contracts into BoogiePL. By using BoogiePL instead of Simplify directly, we can avoid the burden of modeling state change in first-order logic.

Current status

We have developed a model library called "MML" (mathematical model library) [12, 11]. The library has been released and available from our website. Experimental result on model-based specification for existing libraries are being prepared.

Work was started on an automatic translation from Eiffel to BoogiePL called "Ballet". We were able to translate some simple classes and managed to reveal bugs caused by aliasing of references.

Expected results

1. We develop a model library that enables developers to improve their contracts. Together with this library, we provide methodology for implementing models and model-based contracts.
2. We apply the methodology to existing, production quality libraries.
3. A background theory is developed for the direct translation of Eiffel model-based contracts into BoogiePL first-order set theory.
4. Automatic translation tool is provided. This tools compiles Eiffel into BoogiePL, including the identification and translation of model based contracts into the background theory.
5. We integrate the whole process into an interactive development environment with "on the fly" verification capabilities, similar to the ones provided with Spec#.

Scientific contribution

We tightly integrate models and model-based contracts into the specification and verification technology for object-oriented programs. The model library itself is derived from a mathematical theory. Models are directly translated into the language of the underlying proof engine. The mathematical theory is translated into a background theory.

With Eiffel as an established programming language, we can rely on a large set of existing code to produce experimental results. We have started to contract existing libraries using models. This code has been used in large commercial applications for years. We were able to show significant errors in the design and implementation of such libraries using model-based contracts. Some of these errors are documented in [12] and its references.

Open questions

- BoogiePL and existing formalisms rely on the modification clauses to avoid the frame problem [10]. This contradicts the "open world" perspective of modular reasoning. What are the consequences? Can we avoid using modifies clauses? Can the use of models improve this situation?
- Classes that rely on non-trivial pointer structures (for example linked lists) require complex abstraction functions to create the models. How can we describe this abstraction function efficiently?
- How efficient is the prover when working with the background theory? Is the whole approach feasible?

Related work

The work presented here is strongly related to Spec# [2] and its integration into software development. The work on models is related to the efforts done in JML [3]. The overall object-oriented model for verification is based on the work done by Peter Müller on modular specification [9].

References

1. J.-R. Abrial. *The B-Book – assigning programs to meanings*. Cambridge University Press, 1996.
2. M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In *CASSIS 2004*, LNCS 3362. Springer, 2004.
3. Y. Cheon, G. T. Leavons, M. Sitaraman, and S. Edwards. Model variables: Cleanly supporting abstraction in design by contract. Technical Report 03-10, Iowa State University, April 2003.
4. R. DeLine and K. R. M. Leino. BoogiePL: A typed procedural language for checking object-oriented programs. Technical Report MSR-TR-2005-70, Microsoft Research, March 2005.
5. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, July 2003.
6. C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, 1972.
7. B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, Oct. 1992.
8. B. Meyer. *Eiffel: the language*. Object-Oriented Series. Prentice Hall, New York, NY, 1992.
9. P. Müller. *Modular Specification and Verification of Object-Oriented Programs*, volume 2262 of *LNCS*. Springer-Verlag, 2002.
10. P. Müller, A. Poetzsch-Heffter, and G. T. Leavens. Modular specification of frame properties in JML. *Concurrency and Computation: Practice and Experience*, 15:117–154, 2003.
11. B. Schoeller. Strengthening eiffel contracts using models. In Hung Dang Van and Zhiming Liu, editors, *FACS'03*, number 284 in UNU/IIST Report, pages 143–158, September 2003.
12. B. Schoeller, T. Widmer, and B. Meyer. Making specifications complete through models. In R. Reussner, J. Stafford, and C. Szyperski, editors, *Architecting Systems with Trustworthy Components*, volume 3938. Springer-Verlag, 2006.