Object-Oriented Programming and Verification in Spec#: A Course Proposal for ECI 2007

Mike Barnett and Shaz Qadeer
Microsoft Research
{mbarnett,qadeer}@microsoft.com

0 Description

The Spec# programming system is a new attempt at a more cost effective way to produce high-quality software. For a programming system to be adopted widely, it must provide a complete infrastructure, including libraries, tools, design support, integrated editing capabilities, and most importantly be easily usable by many programmers. Therefore, our approach is to integrate into an existing industrial-strength platform, the .NET Framework. The Spec# programming system rests on the Spec# programming language, which is an extension of the existing object-oriented .NET programming language C#. The extensions over C# consist of specification constructs like pre- and postconditions, non-null types, and some facilities for higher-level data abstractions. In addition, we enrich C# programming constructs whenever doing so supports the Spec# programming methodology. We allow interoperability with existing .NET code and libraries, but we guarantee soundness only as long as the source comes from Spec#. The specifications also become part of program execution, where they are checked dynamically. The Spec# programming system consists not only of a language and compiler, but also an automatic program verifier which checks specifications statically. The Spec# system is fully integrated into the Microsoft Visual Studio environment.

The main contributions of the Spec# programming system are

- a small extension to an already popular language,
- a sound programming methodology that permits specification and reasoning about object invariants even in the presence of callbacks,
- tools that enforce the methodology, ranging from easily usable dynamic checking to high-assurance automatic static verification, and
- a smooth adoption path whereby programmers can gradually start taking advantage of the benefits of specification.

In this course, through hands-on use of the Spec# programming system we present the theory and practice for the verification of object-oriented programs.

Spec# is currently under development at Microsoft Research, Redmond. Much more detailed descriptions of Spec# can be found in the list of references.

1 Purpose and Scope

This course addresses the issue of automatically verifying object-oriented programs. It introduces a new programming system (language and tools) for the .NET Framework that allows specifications to be connected directly to the source language. The language allows the concise recording of detailed design decisions within the source code while the tools enforce these decisions either at compile-time, run-time, or both.

As well as being a presentation of state-of-the-art research, the course is meant to provide an introduction sufficient for attendees to begin using Spec#, either directly for programming, for teaching, or as a research platform. Spec# is freely available for non-commerical purposes from `http://research.microsoft.com/specsharp`.

2 Aims and Objectives

After attending this course, participants should have an understanding of the semantics of object-oriented programs. They will understand how program verification is performed, its inherent tradeoffs, and how modern SMT solvers are implemented. They will understand the design space for solutions to the modular static verification of object-oriented programs. They will be introduced to design-by-contract method specifications and object invariants. They should be able to effectively write specifications for their programs and understand the feedback from the automatic static verification in order to conform to the Spec# programming methodology.

3 Relevance

Object-oriented programming has now become the standard programming metaphor. Despite its advantages, the dynamic nature of most object-oriented systems (late binding, dynamic method dispatch, reflection, etc.) has often introduced additional complexity. Various design methodologies and patterns have been developed to help programmers, but without automatic tool support, nothing prevents the introduction of deep, difficult to find and fix errors.

Analysis tools that provide programmer support must be modular. They must also be integrated into the development process, which entails their being integrated into the development environment. Spec# is an attempt to provide specification and analysis for industrial-strength software systems. At the same time, it is a leading-edge research vehicle, especially in regard to object invariants.

4 Audience

The intended audience should be familiar with procedural programming.

5 Syllabus

5.1 Introduction to Spec# Programming

First Lecture Motivation and demo. Undecidability of verification. Soundness and completeness. Automation versus interactive proving. Whole program versus modular analysis. Bug finding versus verification. Precision.

Second Lecture Basics of procedural semantics: abstract language and wp. Refinement. Loops.

5.2 Semantics of Object-Oriented Programs

First Lecture Object-oriented semantics. Memory model. Correctness.

Second Lecture Connection to Spec#. Pre- and postconditions. Non-null type system. Data-flow static analysis.

5.3 Object Invariants

First Lecture Object invariants: single and composite objects. Ownership systems. Abstraction.

Second Lecture Verification condition generation.

5.4 SMT Solvers

First Lecture Nelson-Oppen.

Second Lecture Theories: congruence closure, SAT.

5.5 SMT Solvers

First Lecture Theories: arithmetic, quantification with triggers.

Second Lecture Summary, discussion, questions.

6 Evaluation

Students will be given a take home exam that will then be sent to the instructors after the course has finished. The exam will include problems to be solved using the Spec# programming system.

7 Instructors

Mike Barnett has been with the Microsoft Corporation since July 1995. He first was a member of the Natural Language Processing Group in Microsoft Research, moving to the Foundations of Software Engineering group in the fall of 1999. He is now a member of the Programming Languages and Methods group. Before coming to Microsoft, Mike had been an assistant professor of Computer Science at the University of Idaho for three years. He received his PhD in Computer Science from the University of Texas at Austin in 1992. He is currently working on the Spec# Programming System.

Shaz Qadeer is a member of the Software Reliability Research group at Microsoft Research. His work aims to improve software reliability by providing programmers with automated tools to analyze their programs. He is interested in a variety of program analysis techniques, such as model checking, automated theorem proving, type systems, and run-time verification. Most of his work has focused on applying these techniques to analysis of concurrent software.

References

0. Mike Barnett, Robert DeLine, Manuel Fähndrich, K. Rustan M. Leino, and Wolfram Schulte. Verification of object-oriented programs with invariants. *Journal of Object Technology*, 3(6):27–56, 2004.
1. Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In Gilles Barthe, Lilian Burdy, Marieke Huisman, Jean-Louis Lanet, and Traian Muntean, editors, *CASSIS 2004, Construction and Analysis of Safe, Secure and Interoperable Smart devices*, volume 3362 of *Lecture Notes in Computer Science*, pages 49–69. Springer, 2005.
2. Mike Barnett and David A. Naumann. Friends need a bit more: Maintaining invariants over shared state. In *Seventh International Conference on Mathematics of Program Construction (MPC 2004)*, Lecture Notes in Computer Science, pages 54–84. Springer-Verlag, July 2004.
3. Bart Jacobs, K. Rustan M. Leino, and Wolfram Schulte. Verification of multithreaded object-oriented programs with invariants. In *Specification and Verification of Component-Based Systems*. Computer Science Department, Iowa State University, 2004. TR #04-09.
4. K. Rustan M. Leino and Peter Müller. Modular verification of global module invariants in object-oriented programs. Technical Report 459, ETH Zürich, 2004.
5. K. Rustan M. Leino and Peter Müller. Object invariants in dynamic contexts. In Martin Odersky, editor, *European Conference on Object-Oriented Programming (ECOOP)*, volume 3086 of *Lecture Notes in Computer Science*, pages 491–516. Springer-Verlag, June 2004.
6. K. Rustan M. Leino and Wolfram Schulte. Exception safety for C#. In Jorge R. Cuellar and Zhiming Liu, editors, *SEFM 2004—Second International Conference on Software Engineering and Formal Methods*, pages 218–227. IEEE, September 2004.
7. David A. Naumann and Mike Barnett. Towards imperative modules: Reasoning about invariants and sharing of mutable state. In *Logic in Computer Science (LICS)*, pages 313–323. IEEE, 2004.