Automatic Test-Generation for Spec#

Master Thesis Proposal

Florian Egli
Supervisor: Prof. Dr. Peter Müller, Valentin Wüstholz

November 18, 2011

1 Introduction

Spec# [12, 1] is a superset of C# 2.0 that provides language extensions for contracts. These contracts can be checked statically using a theorem prover to verify the correctness of the code with respect to its specification, or they can be checked at run-time. Microsoft's CodeContracts specification language [4] has similar goals, but is supported by a growing number of tools.

To allow Spec# users to benefit from these tools (e.g., Clousot [7], Pex [11]), we will translate Spec# programs to CodeContracts programs.

For the translation we will use an extended variant of CodeContracts that provides additional specification constructs (e.g., for ghost state or framing). The main challenge will be the encoding of the Spec# methodology (e.g., ownership, invariant semantics).

2 Project Proposal

2.1 Core

The goal of this project is to transform Spec# code into CodeContracts code on the intermediate language (IL) level, as can be seen in Figure 1, so that the transformed Spec# code can be further processes with CodeContracts tools.

One benefit of this translation will be to have automatic test generation for Spec# programs, using Pex, which allows programmers to easily discover bugs in their code via generated counterexamples that violate specified contracts. Another benefit will be to test the Spec# code with the Clousot analyzer.

For the translation, the Spec# IL code has to be rewritten to CodeContracts compatible IL code. The main parts of this projects are, defining the translation rules to translate Spec# code into CodeContracts code, and to implement these rules in the Spec# compiler.

Some Spec# features (e.g., ownership methodology) are only checked statically but not at runtime [10], and make use of implicit background predicates integrated in the

verification logic. In order to provide runtime support, Boogie assertions and verification conditions have to be reintegrated into the IL code as well as additional ghost state. For this, the Spec# compiler and verifier have to be extended to emit CodeContracts++ (CC++) code, instead of Boogie code [3, 2]. CC++ is an extension of CodeContacts, which provides extensions for modifies clauses, ghost state and explicit assumptions, features that otherwise are not supported by CodeContracts.

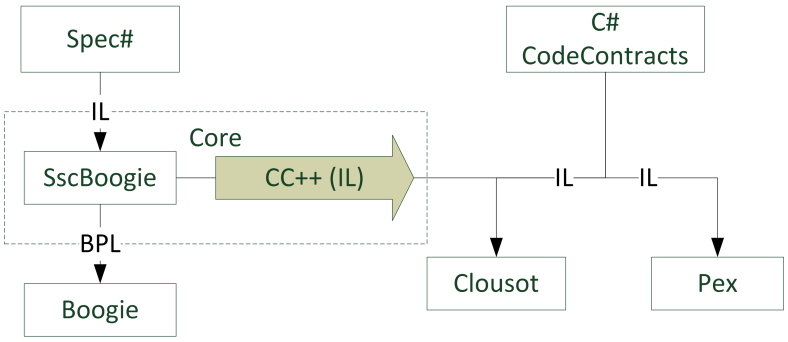

Figure 1: Tool Chain

In the first iteration only a subset of all Spec# language features are translated to CC++.

Required core features:

- All Spec# programming language instructions are supported. All Spec# programs with valid syntax can be translated (even if not all language features are supported).
- Method contracts (pre-/postconditions)
- Object invariants (ownership-based and visibility-based), local expose
- The ownership model [9], including peers, is represented in the CC++ code to work with aggregated objects
- Frame conditions (modifies clauses)
- Contract inheritance (virtual methods)
- Arrays (with covariance tests)
- Loop invariants

The generated CC++ code can be tested with Pex and analyzed with Clousot.

CC++ was partially inspired by Dafny [5] and provides similar specification constructs. In order to have higher confidence in the defined transation rules, the Spec# features

will first be translated into verifiable Dafny code. This can be done only for a subset of the translation rules, because not all features supported by Spec# and CC++ are supported by Dafny (e.g., inheritance).

2.2 Extension Points

Additional features are implemented after the core features of this thesis have been completed and enough time is left.

2.2.1 Extended Language Vocabulary / Language Features

Support for the following language features will be added to the implementation according to their suitability for finding good test cases:

- Reads clauses
- Pure methods
- Model fields [8]
- Non-null types
- Delayed types [6]
- Interfaces
- Exception handling (Exceptional postconditions, checked exceptions)
- Parameter capturing
- Structs
- Additive fields
- Additive expose

2.2.2 Background Predicates

The Spec# source language guarantees additonal properties that are not explicitly declared in the source code [2]. These background predicates, which include axioms, function definitions, and invariant semantics, that are implicitly assumed in Spec#, must also be encoded in the CC++ code. For example, the ownership structure is always a tree and can not be cyclic.

2.2.3 Pex

Pex results are analyzed and interesting examples with good test results are evaluated. The generated CC++ code is optimized for Pex, to achieve better results and to find better test cases.

2.2.4 Clousot

Clousot results are analyzed. The generate CC++ code is optimized for Clousot, to achieve better results and more meaningful output messages.

2.2.5 SscBoogie Verification Result Integration

SscBoogies verification results are reintegrated into the CC++ code, so that other tools, such as Pex, can use this additional information. Methods that are proven incorrect by SscBoogie can be extended with additional assert statements that help finding suitable test cases to reproduce this error.

2.2.6 Visual Studio Integration

All tools are integrated into Visual Studio 2010 and can be used within this environment. This provides a better user experience than working in the command prompt.

2.3 Excluded Features

The following Spec# features are not considered in this thesis, because they are not supported by the current Spec# implementation itself, CodeContracts or are out of scope for this thesis:

- Delegates
- Generics

3 Schedule

Total time available: 6 months (24.10.2011 - 23.04.2012)

- **Milestone:** Initial presentation
- **1 month:** Analysis & Design

  Acquire knowledge in Spec#, CodeContracts, CC++, Boogie, Clousot, Pex, IL, Dafny

  Translate selected Spec# examples manually to CC++ and Dafny

  Test tool chain manually

  Specify translation rules

  Create system design

- **Milestone:** Concept presentation

- **3 months:** Core implementation & Testing

  Iterative implementation and improvement of the system

  System tests

- **2 months:** Extension implementation & Documentation

  Prioritize extensions and implement according to their value for the system

  Final documentation write-up

- **Milestone:** Final presentation

References

[1] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. "The Spec# Programming System: An Overview". In: *Construction and Analysis of Safe, Secure and Interoperable Smart devices*. Vol. 3362. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 49–69.

[2] Mike Barnett et al. "Boogie: A modular reusable verifier for object-oriented programs". In: *Formal Methods for Components and Objects: 4th International Symposium, FMCO 2005, volume 4111 of Lecture Notes in Computer Science*. Springer-Verlag, 2006, pp. 364–387.

[3] *Boogie*. URL: `http://boogie.codeplex.com/`.

[4] *CodeContracts*. URL: `http://msdn.microsoft.com/en-us/devlabs/dd491992.aspx`.

[5] *Dafny*. URL: `http://research.microsoft.com/en-us/projects/dafny/`.

[6] Manuel Fahndrich and Songtao Xia. "Establishing object invariants with delayed types". In: *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. OOPSLA '07. New York, NY, USA: ACM, 2007, pp. 337–350.

[7] Manuel Fähndrich and Francesco Logozzo. "Static contract checking with abstract interpretation". In: *Proceedings of the 2010 international conference on Formal verification of object-oriented software*. FoVeOOS'10. Paris, France: Springer-Verlag, 2011, pp. 10–30.

[8] K. Rustan M. Leino and Peter Müller. "A verification methodology for model fields". In: *European Symposium on Programming (ESOP)*. Ed. by P. Sestoft. Vol. 3924. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 115–130.

[9] K. Rustan M. Leino and Peter Müller. "Object Invariants in Dynamic Contexts". In: *European Conference on Object-Oriented Programming (ECOOP)*. Ed. by M. Odersky. Vol. 3086. Lecture Notes in Computer Science. Springer-Verlag, 2004, pp. 491–516.

[10] K. Rustan M. Leino and Peter Müller. "Using the Spec# Language, Methodology, and Tools to Write Bug-Free Programs". In: *Advanced Lectures on Software Engineering—LASER Summer School 2007/2008*. Ed. by Peter Müller. Vol. 6029. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 91–139.

[11] *Pex*. URL: `http://research.microsoft.com/en-us/projects/pex/`.

[12] *Spec#*. URL: `http://specsharp.codeplex.com/`.