Supporting multiple proof engines by translating between intermediate verification languages

MASTER'S THESIS PROJECT PLAN

Project period: 02.03.2015 – 01.09.2015

Student name: Michael Ameri

Email address: mameri@student.ethz.ch

Supervisor name: Carlo A. Furia

PROJECT DESCRIPTION

Overview

Program verifiers are normally built on top of back-end proof engines. In such setups, the verifier encodes the semantics of the high-level input language (instructions, memory model, specification, and annotations for verification) into an *intermediate language*, which is then used to efficiently generate verification conditions for different proof engines. The two most widely used intermediate verification languages are Boogie and Why. The goal of this project is designing and implementing a translation of Boogie programs into Why programs. The translation will make it possible to use the Why system as a back-end for verifiers that output Boogie code.

Scope of the work

Following the well-known approach of axiomatic semantics, the correctness of a program can be expressed as a collection of purely logical formulae − the so-called *verification conditions*. Logic validity of the verification conditions, which a theorem prover can try to establish, implies that the original program is correct. Generating verification conditions directly from a high-level language is impractical; most program provers perform the translation in two steps: from high-level language to an *intermediate language*, and then from the latter to actual logical verification conditions.

As a concrete example of program verifier that leverages an intermediate language, let us consider AutoProof, the auto-active verifier for Eiffel programs. AutoProof encodes the semantics of annotated Eiffel programs into Boogie programs. The Boogie programming language is a simple procedural language with a rich typed first-order logic for annotations; thus, Boogie programs encoding Eiffel programs are closer to the purely logical view of verification conditions, but still retain substantial imperative constructs. To generate verification conditions that can be fed to a theorem prover, AutoProof runs the Boogie tool on the generated Boogie program. This approach nicely decouples the problem of generating low-level verification conditions from the problem of supporting the specific complex semantics of a high-level language such as Eiffel.

Boogie is an intermediate language for verification used by a variety of verification tools. Another popular intermediate language is Why (or WhyML), part of the Why3 verification system. While the role of Why in a verification toolchain is very similar to Boogie's, the two languages differ in a variety of details, such as: Boogie is a procedural language while Why is a functional language; Boogie mainly supports SMT solvers as backend provers (and it's particularly geared towards Z3), whereas Why supports a variety of backend provers including interactive ones (such as Isabelle).

Implementing a translator from Boogie to Why would then give access to a variety of backend provers to the many verifiers that use Boogie as intermediate verification language, without need for modifications in the toolchain. At the same time, such a translation is challenging because it requires not only that Why programs have the same semantics as the translated Boogie programs, but also that they are amenable to verification using the Why3 verification systems − the translation must be semantics preserving *and* lead to reasonably good performance with Why3.

Intended results

In order to support incremental progress in the project, the first step will require to design (at least 5) *benchmark* programs. Each benchmark is a Boogie program using different features of the language, roughly in increasing complexity.

Upon completion, the project will provide the following items:

1. The benchmarks written and verified in Boogie.
2. A manually produced verified Why version of the benchmarks.
3. A general translation scheme from Boogie to Why defined rigorously in a document. The scheme should cover all features of the Boogie language that are used in the benchmarks.
4. An implementation of the translation scheme, consisting in a program that receives a Boogie program as input and produces the translated Why program as output.
5. An experimental demonstration consisting in running the translator on the benchmarks, and then running Why3 on the translated programs to verify them.
6. The final project report (including the translation scheme mentioned in point 3).

BACKGROUND MATERIAL

Reading list

K. Rustan M. Leino. **This is Boogie 2**. Manuscript KRML 178.
http://research.microsoft.com/en-us/um/people/leino/papers/krml178.pdf

J.-C. Filliatre et al.. **The Why3 manual**. Available online at http://why3.lri.fr/

Julian Tschannen et al. **Automatic Verification of Advanced Object-Oriented Features: The AutoProof Approach**. In Tools for Practical Software Verification – LASER 2011, International Summer School. Lecture Notes in Computer Science, 7682:133–155, Springer, 2012. http://bugcounting.net/pubs/laser12.pdf

PROJECT MANAGEMENT

Objectives and priorities

The primary objectives are:

- The definition of the benchmark programs.
- The definition of the Boogie-to-Why translation scheme.
- The implementation of the translation in a tool.
- The demonstration of its functionalities on the benchmarks.

Criteria for success

The overall success will be assessed as follows:

1. The definition of the translation scheme should be defined with a level of detail sufficient that a programmer could implement it without looking at the implementation (also provided as part of the project).
2. The implementation should run on the benchmark programs without errors, produce syntactically valid Why programs, consistent with the translation scheme, that can be processed by Why3 to verify it.
3. Additional elements for assessment are the usability of the translated Why code, that is how amenable it is to automated (or interactive) verification using the Why3 system.

Method of work

The implementation of the translator can use any parsing technology that is appropriate for the task, to be agreed upon with the supervisor. If this is useful, it may also reuse existing parsers or translators for Boogie and/or Why that are available as open source.

Quality management: documentation

The project documentation consists of the final report which will include:

1. A description of the benchmarks in Boogie, and of how their features are expressed in Why.
2. A rigorous description of the Boogie-to-Why translation scheme.
3. A general description of the translator's design and implementation.
4. Documentation of the translator's interface (CLI and/or API), with a brief usage tutorial.
5. A description of experiments with the benchmarks, discussing in particular how the translated Why code is amenable to verification using Why3, as well as possible limitations of the approach.

PLAN WITH MILESTONES

Project steps

1. Become familiar with the Boogie and Why languages and provers.
2. Develop the benchmarks in Boogie, and manually translate them to Why.
3. Design (and document) the translation scheme from Boogie to Why.
4. Implement the translation scheme.
5. Experiment with the benchmarks using the implemented translator.
6. Write the project report.
7. Deal with unexpected delays.

Deadline

The deadline for the project is 01.09.2015.

Tentative schedule

March	April	May	June	July	August

Step/ week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1.	🟦	🟦																								
2.			🟦	🟦	🟦																					
3.						🟦	🟦	🟦	🟦	🟦	🟦															
4.												🟦	🟦	🟦	🟦	🟦										
5.																	🟦	🟦	🟦	🟦						
6.																					🟦	🟦		🟦	🟦	
7.																									🟦	🟦

Resources

Regulations for Master's theses at ETH Computer Science (notice in particular the grading scheme):
http://www.inf.ethz.ch/content/dam/ethz/special-interest/infk/department/Education/Master-s-Tracks-and-Focus-Areas/Forms---Documents/Master-Studies-Documents/Merkblatt_MSc_Thesis_E.pdf

Microsoft Research's Boogie. http://boogie.codeplex.com/

Why3 verification system. http://why3.lri.fr/

Repository for the project's code and artifacts.

https://bitbucket.org/michael_ameri/boogie2why3