An empirical study into how modularisation can be applied to program verification

Andrew Healy ahealy@cs.nuim.ie
Principles of Programming Research Group
Department of Computer Science
Maynooth University

Supervisors: Dr. Rosemary Monaghan and Dr. James Power

Abstract:

The benefits of modularisation techniques for software development are clear and well understood. The ability to consider systems at multiple levels of abstraction helps to reduce their inherent complexity (and complexity is invariantly a characteristic of large software projects). When the correctness of such projects needs to be verified by formal means, it stands to reason that this task should benefit from modularisation. Verification would benefit from the ability to prove parts of the program in question independently from each other. This would encourage a style of verification that is closer to how programs are naturally developed – piece by piece, making extensive use of previously developed components; verification time would be significantly reduced and the correctness of components (such as libraries) could be trusted without knowledge of their implementation details.

Unfortunately the modularisation constructs that are used in software development (classes, modules, packages, methods etc.) do not have corresponding concepts in the verification process. Instead, source code must undergo a series of transformations before we have proved that it matches its specifications. A typical verification scenario can be thought of as involving three steps. Each of these steps can be seen as a modular part of the overall procedure: (1) source code is annotated with pre- and post-conditions, invariants, etc., to ensure correct behaviour according to specifications; (2) the program is translated to verification conditions by software that interprets the programs and annotations applied in step 1; (3) the verification conditions are written as a number of boolean formulas (goals) that the SMT (Satisfiability Modulo Theories) solver can understand, and these conditions are proved hold or not (often through the use of constraint propagation). As the name would suggest, each SMT solver can only return answers to queries posed in particular logical theories.

This year my research has taken the form of a survey of the uses of modularisation in verification systems – mainly focussing on steps 2 and 3 as outlined above. I have taken the verification condition generation performed by the Why3 platform as a case study with a particular emphasis on how it stores information about the goals generated and the results of applying a particular SMT solver to those goals. This means that when Why3 identifies that a program has changed between verification sessions, only the parts that have changed need to be re-verified. I have also looked at the theories (and their associated decision procedures) that make up SMT solvers as modular constructs. I have compared the operation of two popular and efficient SMT solvers (CVC4 and Z3) by using profiling and code coverage tools on two sets of benchmarks: (1) those from the program verification domain (using verification competition problems as input); and (2) benchmarks written specifically to test SMT solvers from the SMT-LIB project repository. This talk will present an outline of the problem domain and questions I would like to answer with my research.