Reflecting on the Design of the Whiley Programming Language

David J. Pearce and Lindsay Groves

Victoria University of Wellington, Wellington, New Zealand
{djp,lindsay}@ecs.vuw.ac.nz

1 Introduction

The idea of verifying that a program meets a given specification for all possible inputs has been studied for a long time. Hoare's Verifying Compiler Grand Challenge was an attempt to spur new efforts in this area to develop practical tools [1]. A verifying compiler "*uses automated mathematical and logical reasoning to check the correctness of the programs that it compiles*" [1]. Hoare's intention was that verifying compilers should fit into the existing development tool chain, "*to achieve any desired degree of confidence in the structural soundness of the system and the total correctness of its more critical components*". For example, commonly occurring errors could be automatically eliminated, such as: *division-by-zero*, *integer overflow*, *buffer overruns* and *null dereferences*.

In this vein, we have been developing a verifying compiler for the Whiley programming language [2, 3, 4, 5]. Whiley is an imperative language designed primarily to simplify verification. Two important goals of the project are:

1. **Usability.** An important goal is to develop a system which is as accessible as possible, and which one could imagine being used in a day-to-day setting. To that end, the language was designed to superficially resemble modern imperative languages (e.g. Python). Furthermore, unbounded integer and rational arithmetic is used in place of e.g. IEEE 754 floating point (which is notoriously difficult to reason about [6]). Likewise, pure (i.e. mathematical) functions are distinguished from those which may have side-effects.

2. **Expressivity.** Another important goal of the project is to enable programs to be automatically verified as: *correct with respect to their declared specifications*; and, *free from runtime error* (e.g. divide-by-zero, array index-out-of-bounds, etc). However, more complex properties such as *termination*, *worst-case execution time*, *worst-case stack depth*, etc are not considered (although would be interesting future work).

These goals have informed our decisions along the way. For example, to improve usability we want to reduce programmer burden by inferring simple loop invariants (where possible). Likewise, we do not provide syntax for expressing loop variants, since establishing termination is not a consideration. Ultimately, we hope to demonstrate that Whiley is suitable for developing safety-critical systems, although significant work remains to be done. Indeed, many of our choices (e.g. the use of unbound arithmetic) would appear at odds with this goal; however, in many cases, we can exploit specifications to ensure the required properties are met (e.g. that all integers used within a function are, in fact, bounded). Finally, the Whiley verifying compiler is released under an open source license (BSD), can be downloaded from `http://whiley.org` and forked at `http://github.com/DavePearce/Whiley/`.

The Talk. The presentation will overview new developments in Whiley over the last year since SAPLING'13. In that time, we have: used Whiley for teaching 200-level students about verification (approx 130 students); and, completed an exploratory project investigating the use of Whiley on a quadcopter. Both of these provided lots of opportunity to reflect on the design of Whiley, and to guide future design decision. The purpose of the talk is to highlight some of the interesting findings we have made.

References

[1] Tony Hoare. The verifying compiler: A grand challenge for computing research. *Journal of the ACM*, 50(1):63–69, 2003.

[2] The whiley programming language, http://whiley.org.

[3] D. J. Pearce and L. Groves. Whiley: a platform for research in software verification. In *Proc. SLE*, pages 238–248, 2013.

[4] D. Pearce and J. Noble. Implementing a language with flow-sensitive and structural typing on the JVM. *Electronic Notes in Computer Science*, 279(1):47–59, 2011.

[5] D. J. Pearce. Sound and complete flow typing with unions, intersections and negations. In *Proc. VMCAI*, pages 335–354, 2013.

[6] R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. A. Brady. Deciding bit-vector arithmetic with abjc10straction. In *Proc. TACAS*, pages 358–372, 2007.