A Certified Multi-Language Multi-prover Verification Condition Generator

Paolo Herms

1 CEA, LIST, Lab. de Sûreté du Logiciel, Gif-sur-Yvette F-91191
2 INRIA Saclay - Île-de-France, 4 rue Jacques Monod, Orsay, F-91893
3 Lab. de Recherche en Informatique, Univ Paris-Sud, CNRS, Orsay, F-91405

Abstract. Among the various classes of approaches to static verification of programs, the so-called deductive verification approach amounts to verify that a program satisfies a given behavioral specification by means of theorem proving. Typically, given a program and a formal specification, a verification condition generator produces a set of logical formulas, that must be shown to be valid by some theorem prover. Deductive verification tools have nowadays reached a maturity level allowing them to be used in industrial context where a very high level of assurance is required [12]. This raises the question of the level of confidence we can grant to the tools themselves. This is the question we address in this work.

One can distinguish two main kinds of deductive verification approaches. The first kind is characterized by the use of a deep embedding of the input programming language, in a general purpose proof assistant. One of the earlier work of this kind is done in the SunRise system in 1995 [10] where a simple imperative language is defined in HOL, with a formal operational semantics. A set of Hoare-style deduction rules are then shown valid. A SunRise program can then be specified using HOL assertions, and proved in the HOL environment.

The second kind of approaches do not consider a deep embedding of programs: verification condition generators are standalone tools automatically producing verification conditions, usually by means of variants of Dijkstra’s weakest precondition calculus. This is the case of ESC/Java [6], B [1]; the Why platform [9] and its Java [11] and C [8] front-ends; and Spec# [8] and VCC [7] which are front-ends to Boogie [2]. Being independent of any underlying proof assistant, these tools analyze programs where formal specifications are given in ad-hoc annotations language such as JML [5] and ACSL [4]. However, up to now these stand-alone tools have never been formally proved to be sound.

Our goal is to combine the best of both approaches by implementing a certified sound VC generator that can be used independently of any particular proof environment. We present a certified implementation of a verification condition generator inspired by the former
Why tool. In order to make it usable with arbitrary front-ends and arbitrary input specification language, and with arbitrary theorem provers as back-ends, we make it generic with respect to a logical context. This context contains arbitrary abstract data types and axiomatizations defined by a user. On one hand, the input programs of the VC generator are imperative programs written in a core language which operates on mutable variables whose values are some of these logic data types. On the other hand, the output logic formulas are built upon the same logical context. This certified implementation is performed within the Coq proof assistant, but is crafted such that it can be extracted into executable OCaml code. This allows us to realise a plug-in for the Why3 platform replacing its built-in WP calculus by our certified version.

References