

Certificate Translation for the Verification of Concurrent Programs^{*}

César Kunz

IMDEA Software, Spain
FirstName.LastName@imdea.org

Abstract. The increasing presence of multicore execution environments is stimulating the development of concurrent software, an inherently error-prone task that affects the trust on the reliability of third-party code. There is thus a pressing need of providing verifiable evidence on a concurrent software correctness. Certificate Translation provides a means to generate verification certificates for complex functional properties. This technique, consists on progressively transferring verification results for source programs along a sequence of compilation steps. In previous work, we have shown how to transform certificates of a sequential program in the presence of compiler optimizations. In this article, we have shown that it is possible to extend certificate translation to the verification of concurrent programs, based on an Owicki/Gries-like proof system for a shared memory model.

1 Introduction

In the last years, there has been an increasing deployment of computational environments with several processing units. Following this trend, there has been a strong motivation to take advantage of the computational facilities offered by these modern architectures. There is an already widespread awareness of the risks coupled to concurrent program development: program bugs are more likely to occur due to the complexity of such systems, and the unpredictability of the scheduler hinders the reproduction of errors and thus program fixing. There is thus a pressing need to exploit verification methods during code development in order to provide trust on the reliability of the executable code.

Program certification provides a means to efficiently guarantee that a piece of executable code satisfies a required policy. Whereas certificate checking is well understood, certificate generation remains an open problem. In traditional Proof Carrying Code [8], certificates are generated during the code generation by a new module incorporated to the compiler. A negative consequence in this approach is that enforceable properties must be restricted to basic safety policies. In order to extend the set of enforceable properties to more complex specifications, we have proposed to rely on source code verification, with the potential cost of human

^{*} This work is partially funded by the EU projects Mobius and HATS, and by the Spanish project Desafios 10, and by the Community of Madrid project Prometidos.

interaction, and then transfer these verification results to certify the correctness of the compiled code.

A program compiler is commonly defined as a chain of independent stages that transforms a source program into executable code. First, the source program is transformed into a lower-level representation, preserving its overall structure. Subsequent compiler steps transform the code, inserting, moving and removing instructions, and possibly affecting the program structure. Finally, in the last steps, the executable code is generated from the intermediate representations.

In previous work [2,3,1], we have shown that verification conditions are not preserved in the presence of compiler optimizations, and that they can even turn the original program specifications invalid. We have proposed then a technique, *certificate translation*, that transforms simultaneously the specification and certificate, for each optimization step.

This article extends previous results on certificate transformation for sequential programs to a concurrent setting. In particular, we consider a concurrent program execution as the interleaved semantics of its sequential components, in a shared-memory model. We adopt a verification infrastructure similar to an Owicki-Gries logic [11]: verification is split in two independent tasks: for each component, one first verifies that it satisfies its specification in isolation and then one must verify that the other concurrent components do not invalidate this specification. Since the number of verification conditions is exponential in the size of parallel components, practical applications of Owicki-Gries logics aim to reduce the number of verification conditions. This is done in general by grouping code fragments that are known to be atomically executed or by omitting proof obligations that are trivially provable. However, in this article, we do not refer to any criteria to reduce the number of proof obligations. Instead, we deal exclusively with the problem of transferring verification evidence in the presence of optimizations applied to the parallel program components.

Paper overview. In Section 2, we formalize the program representation and provide a simplified verification framework for concurrent programs. In Section 3, we show the existence of certificate translators for the verification framework of the previous section. In Section 3.1, we extend our results on proof-producing analyzers, a main component in a certificate translation process, to a concurrent setting. In Section 3.2, we deal with the transformation of verification results in the presence of compiler optimizations. We conclude and suggest future work in Section 5.

2 Preliminaries

In this section, we formalize the representation of sequential programs and the interleaving semantics of the parallel composition. We define a weakest precondition based verification framework for sequential components and extend it later to deal with concurrent programs.