A Formally Verified Automatic Verifier for Concurrent Programs

Master Thesis - Project Description
Supervised by Thibault Dardinier and Gaurav Parthasarathy

Ellen Arlt

November 16, 2022

1 Motivation

Viper [4] is a verification infrastructure for permission-based reasoning, which includes the Viper intermediate verification language (IVL) and verifiers to check the correctness of the Viper IVL programs. The assertion language in the Viper IVL is based on separation logic [7, 6], a logic used to reason about heap-manipulating programs. The Viper infrastructure has been used as a foundation to develop quite a few verifiers for real-world programming languages. This has been achieved by developing *Viper front-ends* which translate input programs and their specifications, such as pre- and postconditions, into Viper IVL programs so that correctness of the Viper program implies correctness of the input program. There already exist multiple Viper front-ends, examples of which include Gobra [8] for Go, Nagini [3] for Python, Prusti [1] for Rust, and VerCors [2] to verify Java Programs. However it has never been formally proven that the translations into Viper performed by these front-ends are correct. It could very well be that there are bugs in said translations leading to incorrect verification results.

The goal of this project is to understand better how to formally reason about the correctness of Viper front-ends. In order to do so we will formalize a Viper front-end for a simple concurrent programming language in the Isabelle theorem prover [5] and prove its correctness. Even though there has been some prior work on formally reasoning about other translational verifiers, there has been very little work on translational verifiers for programs dealing directly with an IVL as expressive as the Viper IVL.

2 The Project

As our first main contribution we will formalize and create a Viper front-end for a simple, self-designed programming language supporting concurrency. That entails formally defining a function *translate* which takes two arguments, namely the input program and a specification for the input program, and returns a Viper program. In order to define the behaviour of these input programs, we will need to define the small-step semantics of the input programs.

The second main contribution will be the proof that this translation is actually correct. That means proving the following theorem:

Theorem 2.1. *For any input program Pr and specification Sp, if translate(Pr, Sp) is correct with respect to Viper's semantics, then Pr satisfies Sp with respect to the input program's semantics.*

The set of input programs and specifications our front-end supports, as well as the small-step semantics and the function *translate* will be defined in the Isabelle theorem prover [5] which we will also use to prove the above theorem.

3 Goals

Our core goals are:

- The formalization of syntax and semantics of a simple front-end language supporting some form of concurrency to later be translated into the Viper IVL.
- The definition of an assertion language used to express the specifications of aforementioned front-end language to be able to insert pre- and post-conditions as well as loop invariants.
- The definition of the translation of the front-end language and the assertion language into the Viper IVL.
- Formally proving the correctness of said translation for a subset of the commands and capabilities.

Our extension goals are:

- Creating an executable tool which allows us to perform the translation of the front-end language and the assertion language into the Viper IVL (by extracting the translate function in Isabelle to a compilable language).
- Extending our assertion language or front-end language by, for example, adding:
  - fractional permissions
  - recursive predicates
  - heap-dependent functions in the assertion language
  - heap-dependent expressions in the front-end language

References

[1] Astrauskas, V., Müller, P., Poli, F., and Summers, A. J. Leveraging Rust types for modular specification and verification. *Proc. ACM Program. Lang. 3*, OOPSLA (oct 2019).

[2] Blom, S., Darabi, S., Huisman, M., and Oortwijn, W. The VerCors tool set: Verification of parallel and concurrent software. In *Integrated Formal Methods* (Cham, 2017), N. Polikarpova and S. Schneider, Eds., Springer International Publishing, pp. 102–110.

[3] Eilers, M., and Müller, P. Nagini: A static verifier for Python. In *Computer Aided Verification* (Cham, 2018), H. Chockler and G. Weissenbacher, Eds., Springer International Publishing, pp. 596–603.

[4] Müller, P., Schwerhoff, M., and Summers, A. J. Viper: A verification infrastructure for permission-based reasoning. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)* (2016), B. Jobstmann and K. R. M. Leino, Eds., vol. 9583 of *LNCS*, Springer-Verlag, pp. 41–62.

[5] Nipkow, T., Wenzel, M., and Paulson, L. C., Eds. *Isabelle/HOL*. Springer Berlin, Heidelberg, 2002.

[6] O'Hearn, P. W. Resources, concurrency, and local reasoning. *Theoretical Computer Science 375*, 1 (2007), 271–307. Festschrift for John C. Reynolds's 70th birthday.

[7] Reynolds, J. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science* (2002), pp. 55–74.

[8] Wolf, F. A., Arquint, L., Clochard, M., Oortwijn, W., Pereira, J. C., and Müller, P. Gobra: Modular specification and verification of Go programs. In *Computer Aided Verification* (Cham, 2021), A. Silva and K. R. M. Leino, Eds., Springer International Publishing, pp. 367–379.