Optimisation of a Deductive Program Verifier

Bachelor's thesis description

Philippe Voinov

Supervised by Alexander J. Summers and Malte Schwerhoff

1 Introduction

Viper [5] is a set of tools for software verification developed at ETH Zürich. It supports permission-based reasoning, which is useful for verifying concurrent and heap-manipulating programs. Input is given using Viper's own intermediate language.

Users of Viper occasionally create inputs which take unusually long to verify. The reasons for bad performance are often not immediately clear to the user. Though performance issues definitely exist with certain kinds of inputs, there hasn't been a centralized survey and classification of these inputs yet.

Viper currently has two backends to perform verification: Silicon - a verifier based on symbolic execution, and Carbon - based on verification condition generation. This project will focus on investigating and improving verification time in Carbon.

2 Core goals

The first few core goals are about preparing problematic examples and creating a performance testing setup. Since there may soon be a bachelor's thesis project about optimizing Viper's other backend (Silicon), some of the work for these goals may be shared between the two projects. Using the same problematic examples to test both backends could also provide a valuable overview of the strengths of each backend.

The remaining core goals are about improving performance. Since no examples have been collected yet, it is not clear which Viper features are problematic for performance. However some users and developers of Viper have suggested possible areas to focus on to improve performance - for example changing the heap encoding as described in a section below. The actual Viper features investigated will depend on the collected examples. Since Carbon works by encoding into another intermediate verification language (Boogie [4]), it is expected that most performance gains will come from improving the encoding.

Collect and categorize problematic examples

There is currently no centralized collection of examples which cause unusually slow verification in Viper. We plan to collect problematic examples by talking to users of Viper. There are various possible ways to categorize these examples. For example, one could compare the relative verification time in Silicon and Carbon, or identify the Viper features which the examples makes extensive use of.

Create a semi-automatic testing setup

Since this project is focused on improving verification time, it's important to have a consistent way to measure the verification time of many examples with minimal manual effort. We plan to create a tool for which allows measuring the verification time of examples in Silicon and Carbon. We will deploy the tool on a server and allow comparing timings against a baseline version of Viper for consistent results. It will provide raw timings and summarized reports for the executed examples. We plan to create a mechanism to easily submit a version of Viper from source control for performance testing.

Improve heap and permission encoding

Methods in Viper are translated into Boogie as procedures which manipulate a heap and a mask. The heap stores the values of object fields, whereas the mask stores the permission amount which the method holds for that field. There has been some work comparing different heap encodings in Boogie [1], which showed significant performance differences between different encodings. Carbon currently encodes the heap as a Boogie map which is indexed by a reference paired with a field (H[p, f]). However other approaches, like splitting the heap into separate maps per field (H[f][p]), may be faster. We plan to try encoding the heap and mask using the forms of maps presented in [1]. This will be an experiment and possibly done by hand. We then plan to implement the most promising encoding as an alternative heap encoding in Carbon.

Improve encoding of predicates

Viper uses predicates to abstract over assertions. Recursive predicates allow specifying data structures such as lists. Replacing a predicate with its body (an assertion) is called unfolding. Carbon currently creates a fresh heap and mask after each fold and unfold operation in a method. We plan to modify Carbon so that information about predicates is stored separately from the main heap and mask.

Profile time spent at every layer

Carbon primarily encodes an input for Boogie and then interprets the results, so it's possible that only a small portion of the verification time is spent in Carbon itself. However, some early investigation has shown that Boogie runs for less than half of the total verification time of some examples, even when ignoring the startup time of the JVM which Carbon runs in. We plan to measure the time spent on various stages of execution (for example starting Carbon or parsing input) to understand which of them take unnecessarily long. We will attempt to optimize these stages so that most of the total verification time is spent in Boogie.

3 Extension goals

Improve wildcard permission encoding

Wildcard permissions are an implementation of abstract read permissions in Viper [6] [3]. Abstract read permissions extend the traditional model of fractional permissions, without forcing the user to explicitly specify the fraction of a permission held by a method. The current encoding of wildcard permissions in Carbon makes the SMT solver find concrete permission amounts, even though this may not always be necessary. We plan to find examples which are slow due to wildcard permissions, and then investigate whether encoding wildcard permissions without searching for concrete permission amounts is possible for those examples.

Adjust configuration of underlying tools

Carbon encodes to Boogie, which uses the Z3 SMT solver [2] by default. Boogie and Z3 have various configuration options. Most options currently used in Carbon have not been chosen systematically. We plan to investigate how various options affect verification time of the problematic examples in Viper.

Try another SMT solver

Currently Boogie uses the Z3 SMT solver. It should be possible to use a different SMT solver with Boogie. We plan to use the CVC4 SMT solver and investigate how verification time changes.

References

[1] Sascha Böhme and Michał Moskal. Heaps and data structures: A challenge for automated provers. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 177–191, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[3] Stefan Heule, Rustan Leino, Peter Müller, and Alexander J. Summers. Abstract read permissions: Fractional permissions without the fractions. volume 7737, pages 315–334. Springer Berlin Heidelberg, January 2013.

[4] Rustan Leino. This is boogie 2. Microsoft Research, June 2008.

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

[6] Benjamin Schmid under supervision of Vytautas Astrauskas, Marco Eilers and Peter Müller. Abstract read permission support for an automatic python verifier. Bachelor's thesis, ETH Zürich, 2018.