Inference of Pointwise Specifications for Heap Manipulating Programs

Master Thesis Project Description

Severin Münger
Supervisors: Alexander J. Summers, Caterina Urban
ETH Zurich

October 19, 2016

1 Introduction

The high level goal of software verification is to prove mathematically that, given a computer program and its specification, the program is correct with respect to the specification. In general, the process of verifying a given program is not trivial and tedious to do by hand. In order to make software proofs practically usable, they have to be automated. Over the last 15 years, significant progress has been made in the field of software verification - both theoretically and practically. One especially important achievement has been the development of so-called permission logics that facilitate reasoning about programs that access and modify the heap during their execution, separation logic being probably the best-known example among them. To be able to reason about the heap in a simple way is very significant in many contexts, e.g. in verifying algorithms using data structures such as lists. Though possible, specifying pointer-based algorithms remains very cumbersome without special support.

Recently, a new intermediate verification infrastructure, called Viper [2], has been developed that natively supports permissions. Viper consists of multiple components. At the front-end, Viper provides an interface to various high-level programming languages such as Java or Scala that are translated to Viper's intermediate verification language. One of the core features of Viper is the possibility to include permission assertions in contracts. For this purpose, it provides the so-called accessibility predicate which has the general form `acc(e.f, p)` where `e` is a reference, `f` is a field of reference `e` and `p` is a permission. The accessibility predicate can also be used in quantified expressions to define permissions over a (not necessarily bounded) set of references. The general form of quantified permissions in Viper look as follows:

$$\textbf{forall}\ \mathtt{q} : \mathtt{T} :: \mathtt{c(q)} \Longrightarrow \textbf{acc}(\mathtt{e(q).f}, \mathtt{p(q)}) \tag{1}$$

While permissions significantly simplify expressing heap-modifying algorithms, they also introduce overhead. For every heap access inside a method body there have to be enough permissions for the respective location before the access (e.g. through a precondition or a so-called inhale) and the corresponding permissions also have to be properly released afterwards (e.g. through a postcondition or a so-called exhale). Optimally, we would like to reduce the need of writing such contracts by hand in order to facilitate the specification progress and ultimately to make Viper more usable. A recent master thesis [3] employed backward abstract interpretation [1] to track location accesses

(reads, writes), inhales and exhales to infer preconditions and loop invariants for quantified permissions. While the approach in the thesis looks promising, it suffers from some problems. In certain cases, the analysis can be very imprecise in the sense that it requires access to more heap locations than actually needed. A further limitation is that the analysis is only performed backwards. This means that the inference is able to determine the locations that need to be granted permissions but does not handle the release of the permissions afterwards. The high-level goal of this master thesis is to generalize and formalize the results from [3] and to improve the approach by extending it with additional analyses. The final implementation of the newly devised approach should optimally be elaborate enough to be included in the Viper infrastructure.

2 Core Goals

In this master thesis, we would like to extend the support of automated inference of (permission-related) contracts of Viper - both in theory and in practice. In particular, we would like to achieve the following:

- **Design of unified approach** The master thesis [3] describes an algorithm which is basically divided into two separate parts: one for handling arrays and one for handling general graphs. It has to be chosen before the analysis which specialized inference to use. Moreover, there currently is an unsoundness in the graph part of the algorithm. We would like to come up with a new way of relating these two separate cases and ideally to combine them to an elaborate unified and sound solution.
- **Forward analysis to infer post-conditions** Until now, only weakest preconditions have been inferred, i.e. requiring just enough permissions for a method to verify. In this thesis the aim is to also infer strongest postconditions. For this we would have to combine the current approach with an additional forward analysis using abstract interpretation. The main point of this extension is to not only be able to require enough permissions for a method in the precondition but also to return the appropriate permissions at the end of a method in the postcondition. This way we will be able to give stronger guarantees to method callers and we believe that this could also benefit the strength of loop invariants.
- **Quantifier elimination** In order to deal with changing variables inside loops, [3] introduced the `forget` operator. A numerical analysis (also by employing abstract interpretation with an interval domain for integer-typed variables) was used to infer all the possible values for a changing variable inside a loop. The result of this numerical analysis is used together with the result of the backward permission tracking analysis to generate the quantified permission assertions. Unfortunately, this forget operator in general imposes imprecision as it relies on abstract interpretation. We aim to devise a more precise approach using quantifier elimination.
- **Include heap analysis to generalize support** We would like to also support inference in case of aliasing or heap dependent receivers in arrays which is not handled in the existing algorithm. At the time of writing this description, there is work in progress to develop a heap analysis for Viper which would facilitate the inference for such cases.

3 Possible Extensions

- **Revisit and refine design of the inferred sets** For the part of the approach that infers quantified permissions over sets for graph data structures, we would like to specify the inferred sets more precisely and optimally we would also like to get rid of possibly duplicated or redundant sets. This could go so far as to also consider all the call sites of the method in question and to try to match the appropriate parameters with the sets in the contracts. An important point to consider is that the inferred contracts should still be human readable. We will probably have to balance between preciseness and readability of the inferred contracts, especially regarding the inferred sets.
- **Formal description and soundness proof** We would like to precisely describe the newly developed algorithm formally. We could take this even further and prove the soundness of our approach. Soundness in the context of inference of quantified permissions means that it is guaranteed that for every heap read access we hold at least a fractional (read) permission for the respective location and that for every write access we have write permission.
- **Generalize quantifier elimination** Unfortunately, quantifier elimination is only possible in certain restricted theories. For theories that do not fall into this category, however, it is possible to at least compute a formula without quantified variables that is implied by the original formula while it is still as strong as possible. We would therefore like to combine logical fragments that target such theories in order to be able to handle a wider range of formulas. We could for example make use of an algorithm to compute the cover of formulas in the theory of uninterpreted functions and linear arithmetic.

References

[1] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. ACM.

[2] P. Müller, M. Schwerhoff, and A. J. Summers. *Viper: A Verification Infrastructure for Permission-Based Reasoning*, pages 41–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[3] S. Walter. Automatic Inference of Quantified Permissions by Abstract Interpretation. Master thesis, ETH Zürich, 2016.