Generalized Verification Support for Magic Wands

Bachelor's Thesis Description

Nils Becker, nbecker@student.ethz.ch

Supervisors:
Alexander J. Summers, alexander.summers@inf.ethz.ch
Malte Schwerhoff, malte.schwerhoff@inf.ethz.ch

ETH Zürich, March 2017

1 Introduction

The Viper project [3], consisting of an intermediate language and verification tools, is being developed by the Chair of Programming Methodology at ETH Zurich. It uses two verifiers: Silicon, which is based on symbolic execution, and Carbon, which is based on verification condition generation.

These tools are based on separation logic, within which one operation is the magic wand, which, among other things, can be used to encode information about partial data structures. Since verification in the presence of magic wands has been shown to be undecidable, one approach to automating their verification is for the verifier to require guidance, in the form of ghost code, from the user. This ghost code is statements that are inserted into the program code, but are not part of the program execution. Rather, they instruct the verifier to transform the current verification state.

In their paper [4], Schwerhoff and Summers describe a lightweight and largely automated approach for verifying properties expressed using magic wands. Their approach is restricted to only a few ghost operations, that can be nested. In their approach the *package* ghost operation is used to create a magic wand instance and add it to the current verification state.

For example, packaging `acc(x.f) --* Cell(x)`, where `Cell()` is defined as `predicate Cell(x: Ref) {acc(x.f)}` and `acc(x.f)` means full access to `x.f`, requires the verifier to prove that `Cell(x)` follows from `acc(x.f)`. This, too, requires guidance, resulting in the package statement `package acc(x.f) --* folding Cell(x) in Cell(x)`, which can be verified using Viper. In this case two ghost operations were used: `package` instructs

the verifier to add the magic wand to the current verification state and `folding` replaces the definition of `Cell()` with the predicate when proving the right-hand-side.

Blom and Huisman have developed [1] a different technique that allows arbitrary blocks of statements to guide the verifier when proving that the right-hand-side of the magic wand follows from its left-hand-side. Their approach can therefore be used to verify more general magic wands. Note especially, that since arbitrary code blocks can be used for the proof of the right-hand-side, it is possible to branch during the proof by using regular if-then-else statements.

Another problem with the current approach is Viper's error reporting for nested ghost operations and functions, which makes it difficult to localize errors. For example, consider `package Cell(x)` $-\!\!*$ $\boxed{1}$ `unfolding Cell(id(x)) in` $\boxed{2}$ `id(x) == x`, where `Cell()` is defined as above and `id()` is the identity function with the precondition `Cell(x)`. Note also that $\boxed{1}$ and $\boxed{2}$ are not part of the Viper syntax, but serve to denote the two distinct states, at the marked points, in the following discussion. This package operation will always fail, because after unfolding `Cell()` in state $\boxed{1}$ it is no longer available in state $\boxed{2}$ and thus the precondition of the second `id()` cannot be established. For this example the current mechanism for error reporting only includes information about which *package* operation failed and that the precondition of `id(x)` could not be verified. This information is, however, not sufficient to figure out which of the two `id(x)` calls caused the issue. Note that, while the problem is not specific to magic wands (`assert unfolding Cell(id(x)) in id(x) == x` would lead to the same issue), it is especially relevant with respect to magic wands, since the current implementation requires all ghost operations to be nested. In addition to improving error reporting the goal of this thesis is to generalize Viper's support for magic wands.

2 Core Goals

- Design a language extension for Viper that enables expressing more general proofs and proof structures for magic wands. This extension should support blocks of statements as ghost operations, branching within a proof and it should work well with nested *package* operations.
- Implement this language extension for at least one of Viper's verifiers.
- Develop test cases and real world examples using magic wands that can be verified using the new approach, but not using the old one.
- Evaluate the annotation overhead of the new approach, by comparing it to the old approach based on a set of examples, and assess the value of keeping the old syntax as syntactic sugar. If necessary, reimplement the old syntax for the new approach.
- Improve error reporting to display all necessary information to localize errors in more complex proof structures (including proofs that do not use magic wands).

3 Extended Goals

- Evaluate implications of the new syntax on referencing intermediate states and, if necessary, add syntax to facilitate such references: since the current syntax requires the proof for the right-hand-side of a magic wand to be nested inside the package operation it is, for example, not currently possible to reference the states $\boxed{1}$ and $\boxed{2}$ from the example above, since this would require a nested version of the *label* operation.

- Integrate magic wands with quantified permissions. Quantified permissions in Viper in their simplest form look like `forall x:Ref :: c(x) ==> acc(x.f)`, meaning that for any x that satisfies the condition `c(x)` access to a field `f` of x can be obtained. Quantified magic wands could be implemented as two independent parts: the first one can be understood as sets of magic wands. In this case `acc(x.f)` from the example above is simply replaced by a magic wand. The second step would be to allow quantified permissions inside the left-hand-sides and right-hand-sides of magic wands. Both Müller in her master's thesis [2] and Schwerhoff in his PhD thesis [5] outlined basic support for the former, but these approaches have neither been fully developed nor implemented.

- Improve and extend Silicon's heuristics for inferring magic wand related annotations to work well with the new approach.

- Investigate to which extent similar heuristics can be implemented in Carbon.

- Develop support for using magic wands in functions. This will most likely require an *applying* expression, similar to *unfolding* for predicates, that can be used to temporarily apply a magic wand.

References

[1] Stefan Blom and Marieke Huisman. Witnessing the elimination of magic wands. *International Journal on Software Tools for Technology Transfer*, 17(6):757–781, 2015.

[2] Nadja Müller. Generalised Verification for Quantified Permissions. Master's thesis, ETH Zürich, Switzerland, 2016.

[3] 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.

[4] M. Schwerhoff and A. J. Summers. Lightweight Support for Magic Wands in an Automatic Verifier. In J. T. Boyland, editor, *European Conference on Object-Oriented Programming (ECOOP)*, volume 37 of *LIPIcs*, pages 614–638. Schloss Dagstuhl, 2015.

[5] Malte Schwerhoff. *Advancing Automated, Permission-Based Program Verification Using Symbolic Execution.* PhD thesis, ETH Zürich, 2016.