Sound Automation of Magic Wands in a Symbolic-Execution Verifier

Bachelor Thesis Description

Nicola Widmer

Supervised by Thibault Dardinier, Dr. Malte Schwerhoff and Prof. Peter Müller

March 29, 2022

1 Introduction

Separation logic [5] is an extension of Hoare logic [3] widely used to reason about heap-manipulating programs. It is also the basis of many verification tools such as Viper [4]. The most important connective in separation logic is the separating conjunction, $*$. Intuitively, the separation-logic assertion $A * B$ expresses that A and B hold in two disjoint portions of the program heap. Another important connective is the magic wand $-\!\!*$, which is similar to the implication but for separation logic. If $A * (A -\!\!* B)$ holds in a state, then so does B. The magic wand is very useful, for example to express invariants while traversing data structures such as lists or trees. This is the case, because $A -\!\!* B$ intuitively refers to the data structure B "minus" the data structure A. For that reason magic wands are already supported in Viper, and this approach is based on a package algorithm [6] that automates the computation of a footprint. Unfortunately the proposal is unsound [1]. But there is a proposal for a frame-work to characterise possible sound package algorithms [1]. There is already an implementation based on that framework for Carbon, a verification condition generation based backend for Viper. This thesis is concerned with implementing a sound package algorithm for Silicon [7], which is the backend for Viper and is based on symbolic execution.

2 Background

2.1 Package/Apply

To use magic wands in Viper there are two statements **package** and **apply**. If we **package** a wand then Viper tries to find a footprint for the wand and then remove the footprint from the current state and add the wand to the current state. The footprint of a wand is a state which, combined with any compatible state in which A holds, yields a state in which B holds. The footprint has to be removed from the state because otherwise there could be changes to the footprint that change the state in a way such that the magic wand no longer holds. If we **apply** the wand then we remove the wand and resources that satisfy the left-hand side of the wand from the state and add resources that satisfy the right-hand side of the wand to the state.

2.2 Current Package Algorithm in Silicon

To show the key idea of the current package algorithm [6] we look at the execution for the general wand $A \mathrel{-\!\!*} B$. First, the algorithm constructs an arbitrary state σ_A in which A is satisfied by inhaling A in an empty state. Then the algorithm tries to construct a state σ_B in which B holds, by trying to take the permissions it needs from σ_A and, if σ_A doesn't contain them, from the current state.

2.3 Unsoundness of the Current Algorithm

It has been shown that the current algorithm is unsound [1]. One problem is that the current algorithm sometimes performs a case split on the content of σ_A. The algorithm then computes a footprint for all cases individually. However, sometimes none of these footprints are strong enough to satisfy B together with any state σ_A in which A holds. We illustrate the issue below, with an example that uses fractional permissions. Fractional permission [2] is an extension of separation logic in which permission to a heap location is not binary but a fraction between 0 and 1. A full (1) permission is then required to write, and positive permission is required to read a heap location.

Listing 1: Example of a Viper program that shows how to prove false using package and apply statements. This program is verified by both Carbon and Silicon.

```
field f: Bool

method main(x: Ref, a: Ref, b: Ref)
 requires acc(x.f) && acc(a.f) && acc(b.f)
 ensures false
{
 package acc(x.f) && (x.f ? acc(a.f, 2/4) : acc(b.f, 2/4)) --*
    acc(a.f, 3/4) && acc(b.f, 3/4)
 assert (perm(a.f) == 3/4 && perm(b.f) == 1/4) || (perm(a.f) ==
     1/4 && perm(b.f) == 3/4)
 x.f := perm(a.f) != 1/1
 apply acc(x.f) && (x.f ? acc(a.f, 2/4) : acc(b.f, 2/4)) --*
    acc(a.f, 3/4) && acc(b.f, 3/4)
```

```
11 }
```

Here we package the wand `acc(x.f)*(x.f ? acc(a.f,1/2) : acc(b.f,1/2) ) --* acc(a.f,1/2) * acc(b.f,1/2)`. The algorithm then computes the footprint for two cases, one where `x.f` is true and one where `x.f` is false. For the true case the footprint is `acc(b.f,1/2)`, because the left-hand side of the wand already provides `acc(a.f,1/2)`, only `acc(b.f,1/2)` is needed from the current state, and for the false case it is `acc(a.f,1/2)` but, none of them are valid footprints. A valid footprint would be the union of both, namely `acc(a.f,1/2)*acc(b.f,1/2)`. After the package, we are in a state where we have the wand and either full permission to `a.f` and half to `b.f`, or the other way around, as shown by the assertion on line 8. Then we set x.f in such a way that we don't have to give up part of the full permission to satisfy the right-hand side of the wand. We can then use the wand to get half permission to a field for which we already have full permission, which leads to an inconsistent state, hence we can prove the postcondition false on line 5. Since we started from a valid state, this example shows that the current algorithm is unsound.

2.4 Sound Package Algorithm

There is already a recipe for a sound package algorithm [1]. Let us again look at the general wand $A \mathrel{-\!\!*} B$. The key idea is that we start with the *set of all states* σ_A that satisfy A, instead of looking at each state individually. We then extract from the current state as much as we need and add it to all σ_A's in the set such that they all now satisfy B. This approach is already implemented in the Carbon back-end for Viper. However, it is not straightforward to implement this approach in Silicon, since there is not an obvious way to represent the set of states σ_A which statisfy A as it is potentially infinitely large.

3 Goals

The goal of this thesis is to develop a sound automated package algorithm for Silicon.

3.1 Core Goals

- Develop a categorisation of wands. Since the current algorithm is sound for many practical wands, a syntactic and a semantic criterion for wands which can be soundly packaged by the current algorithm would help a lot.

- Explore how to overcome the hurdle of representing the set of possible infinite states.
- Develop and implement a sound package algorithm for Silicon based on the findings of the previous points.
- Evaluate the implemented algorithm with respect to completeness and performance. In particular, compare it to the current unsound algorithm.

3.2 Extension Goals

- Implement alternative packaging strategies as part of the algorithm.
- Add support for advanced features of the package algorithm such as nested wands, quantified permissions and proof scripts.

References

[1] Anonymous authors. "Sound Automation of Magic Wands". Submitted.

[2] John Tang Boyland. "Checking Interference with Fractional Permissions". In: *SAS*. 2003.

[3] C. A. R. Hoare. "An Axiomatic Basis for Computer Programming". In: *Commun. ACM* 12.10 (Oct. 1969), pp. 576–580. ISSN: 0001-0782. DOI: 10.1145/363235.363259. URL: https://doi.org/10.1145/363235.363259.

[4] Peter Müller, Malte Schwerhoff, and Alexander J. Summers. "Viper: A verification infrastructure for permission-based reasoning". en. In: *Dependable Software Systems Engineering*. Ed. by Alexander Pretschner, Doron Peled, and Thomas Hutzelmann. Vol. 50. Amsterdam: IOS Press BV, 2017, pp. 104–125. ISBN: 978-1-61499-809-9. DOI: 10.3233/978-1-61499-810-5-104.

[5] J.C. Reynolds. "Separation logic: a logic for shared mutable data structures". In: *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. 2002, pp. 55–74. DOI: 10.1109/LICS.2002.1029817.

[6] Malte Schwerhoff and Alexander J. Summers. "Lightweight Support for Magic Wands in an Automatic Verifier". In: *29th European Conference on Object-Oriented Programming (ECOOP 2015)*. Ed. by John Tang Boyland. Vol. 37. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 614–638. ISBN: 978-3-939897-86-6. DOI: 10.4230/LIPIcs.ECOOP.2015.614. URL: http://drops.dagstuhl.de/opus/volltexte/2015/5240.

[7] Malte H. Schwerhoff. "Advancing Automated, Permission-Based Program Verification Using Symbolic Execution". en. PhD thesis. Zürich: ETH Zurich, 2016. DOI: 10.3929/ethz-a-010835519.