Adding Generalized Magic Wand Support to a Verification-Condition-Generation-Based Verifier

Bachelor Thesis Description

Ahmed Gamal

ahmed.ahmed@inf.ethz.ch

Supervised by:

Alexander J. Summers

Malte Schwerhoff

ETH Zurich, Switzerland

April 2018

1 Introduction

Separation logic is a permission-based verification logic that extends Hoare logic. Separation logic is the basis of many automatic verifiers, and one of its useful connectives that is hard to implement in automatic verifiers is the magic wand. Magic wands are useful for specifying partial data structures or loop invariants for iterative traversals of data structures.

In [1] Malte Schwerhoff and Alexander J. Summers provided an approach to support magic wands in automatic verifiers along with an implementation of their approach in Silicon. Silicon is one of two verifiers that the Viper project [2] uses. Silicon is based on symbolic execution. Currently Silicon supports a newer version of magic wands [4] than the one discussed in [1]. The new approach has made it possible to write more arbitrary code inside package statements like branching statements for example. One of the main problems with the older approach was the difficulty of localizing errors due to the nested structure of the ghost operations in package statements. It is much easier to localize errors in the new approach as it allows adding assert statements which help in debugging and localizing errors.

The other verifier that Viper uses is Carbon, which is based on verification condition generation. There were some attempts [3] to use the same algorithm discussed in [1] to support magic wands in Carbon but ,with respect to the current support for magic wands in Silicon, the current encoding is incomplete and shows inconsistent behavior in some cases. The goal of this thesis is to provide a suitable encoding in Carbon for the generalized version of magic wands in [4].

2 Core goal

The core goal of this thesis is to provide support for magic wands in Carbon that is compatible with the current version in Silicon. To achieve this goal we need to complete specific tasks. These tasks are:

- Finding a suitable encoding for magic wands in Boogie [5]. As earlier work in [3] has shown, finding such encoding is not trivial. We have two directions here. One is to extend the current encoding in Carbon of the older version of magic wand that is mentioned in [3]. The other direction is to design a new encoding for supporting the new version. The direction we intend to take for our core goal is extending the current encoding in [3]. We have included the other direction in the extensions. The main reason we included extending the current encoding in the main goals instead of designing a new one is that the current encoding was difficult to come up with, and as mentioned in [3] it is not even clear whether such an encoding that precisely models the package operation exists.
- Implementation of generalized magic wand support in Carbon. This step will come after deciding on the encoding to be used. We then can start implementing this encoding in the current version in Carbon.
- Reconsidering the current architecture of the code. The current implementation treats statements differently inside and outside package statements. Finding a more general implementation for operations that can be used anywhere in the code will make it easier to include more arbitrary code inside package statements. It will also make it easier to maintain the code and to change it to support newer versions.

3 Extensions

Possible extensions for this bachelor project are:

- Redesigning the encoding in Boogie. The encoding in [3] is incomplete and results unpredicted behavior in some cases as discussed in [3]. A possible extension is trying to redesign the encoding to solve this problem of inconsistent behavior. It is not clear whether such an encoding exists or not. It will also be an interesting result if we find out that such an encoding does not exist.

- Adding combinations between various features that are currently supported by Viper. One interesting combination for example is the combination of magic wands and quantified permissions. Such combination could be implemented by supporting quantified permissions on both right-hand-sides and left-hand-sides of magic wands. Another part of supporting this combination is to be allowed to write something like ***forall x:Ref :: p(x) ==> A --* B*** ,which means for any x for which condition p(x) holds, we can obtain the magic wand ***A --* B***.
- Encoding and verifying more serious examples using magic wands in Viper.

4 References

[1] 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.

[2] 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.

[3] Gaurav Parthasarathy. Adding Magic Wand Support to Carbon (Bachelor thesis), ETH Zurich, 2015

[4] Nils Becker. Generalized Verification Support for Magic Wands (Bachelor thesis), ETH Zurich, 2017

[5] K. Rustan M. Leino. This is boogie 2, 2008.