Automating Magic Wands with Advanced Features

Practical Work Project Description

YIQUN LIU
Supervisor: Thibault Dardinier

1. Introduction

Viper. Viper (verification infrastructure for permission-based reasoning) [1] is a language and suite of tools, providing an architecture that supports the expressive separation logic and its automatic verification natively. This common architecture simplifies the development of new verifiers and verification techniques via translation of the source language to Viper. The current verifiers based on this include Gobra [2] for Go, Nagini [3] for Python, and Prusti [4] for Rust. Viper has two kinds of verification backends: symbolic execution and verification condition generation translating Viper programs to Boogie [5].

Separation Logic. The separation logic in Viper describing the properties of the program is separation logic [6], which is an extension of Hoare logic that permits local reasoning about low-level heap-manipulating imperative programs that use shared mutable data structures. Traditionally, the program state contains two components: a stack, mapping variables to values as in Hoare logic, and a heap, mapping addresses to values with permission amounts of mutable data structure. The permission amount controls the accessibility of addresses. 1 represents the exclusive write permission and larger than 0 a shared read permission. This can frame heap-dependent information and prove the absence of aliasing and data racing.

Quantified Permissions. The access pattern of the data structure consisting of a set of heap locations is orderly or random. For example, in the case of a linked list, the accessibility of the current node must be acquired before that of the next node. Recursive predicates specify this orderly relation between accessibility of heap locations in the data structure via information hiding. For data structures with random access like arrays, quantified permissions embody their unbounded sets of accessibility of heap locations.

Magic Wand. The magic wand (separating implication) A --∗ B expresses the "minimal sufficient" difference between all the pairs of the states that satisfy A and B respectively. In other words, an assertion C is a sufficient difference, or footprint we call it later, if all states σ which can be split into two compatible states σ_A and σ_C such that A and C hold in σ_A and σ_C respectively, B holds in σ; A--*B represents one such C which can be inferred from any of them. It is worth noting that the "the minimum sufficient" difference is not always possible in the case of disjunction. This logic connective is useful for representing part of data structures.

Package Operation. Due to the quantification over states in the semantics of the magic wand, its fully automatic verification is undecidable in the presence of variables and other logical features [7]. Thus, we need to provide user guidance to direct the verifier's proof search by ghost operations [8], which rewrite the verification state but don't change the program state. The footprint of A--*B is part of the current state and satisfies B combined with any state satisfying A; the package operation calculates a footprint, removes it from the current state, keeps it from any modification, and records the wand instance; the apply operation removes the recorded wand instance, combines the current state with the footprint, and gets the assertion B if A holds in the current state.

Current state of implementation. The current package algorithm in Viper corresponds to the package logic in [9]. The logic provides a sound and complete framework for describing the

package algorithm. Every package algorithm corresponds to a proof search strategy in the package logic. However, the implementation has not been combined with other advanced features in Viper now, including in particular quantified permissions on the left- and right-side of the magic wand.

2. Tasks

The aim of this project is to improve the support for magic wands in Viper. More precisely, it focuses on supporting the combination of the magic wand with other separation logic features and improving its implementation or documentation in Carbon. The stated goals can be achieved by the following steps:

Core Goals

1. Combine with quantified permissions [10]: Quantified permissions on the right or left side of the magic wand will require nested quantification inside the implementation of the magic wand. The fact that the footprint possibly takes the part or whole quantified permissions of one or more quantified permissions makes the algorithm harder.
2. Combine with abstract predicates [11]: The package algorithm should keep track of known-folded permissions recording the previously known heap locations even if its corresponding permission is packaged into the footprint.
3. Improve implementation and documentation of the magic wand.

Extension Goals

Improve the package algorithm with one or more of the following:

a) Fractional magic wands [12]: This requires the extension of the state to support more than full permission in the heap.
b) Combine with different strategies for packaging: We should provide different choice functions when taking permission from the current remaining state.
c) Nested package operation: To guarantee that the combination of magic wand A --∗ B with any state satisfying A satisfies B, we need to consider a set of all states satisfying A. The nested package operation requires that the algorithm considers a set of sets of states satisfying A.

References

[1] Müller, P., Schwerhoff, M. and Summers, A.J. (2015) 'Viper: A verification infrastructure for permission-based reasoning', *Lecture Notes in Computer Science*, pp. 41–62. doi:10.1007/978-3-662-49122-5_2.

[2] Wolf, F.A. *et al.* (2021) 'Gobra: Modular Specification and verification of go programs', *Computer Aided Verification*, pp. 367–379. doi:10.1007/978-3-030-81685-8_17.

[3] Eilers, M. and Müller, P. (2018) 'Nagini: A static verifier for python', *Computer Aided Verification*, pp. 596–603. doi:10.1007/978-3-319-96145-3_33.

[4] Astrauskas, V. *et al.* (2022) 'The Prusti Project: Formal verification for rust', *Lecture Notes in Computer Science*, pp. 88–108. doi:10.1007/978-3-031-06773-0_5.

[5] Leino, K.R.M. (2018) *This is Boogie 2*, *Microsoft Research*. https://www.microsoft.com/en-us/research/publication/this-is-boogie-2-2/

[6] Reynolds, J.C. (2002) 'Separation logic: A logic for shared mutable data structures', *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. doi:10.1109/lics.2002.1029817.
[7] Brochenin, R., Demri, S. and Lozes, E. (2012) 'On the almighty wand', *Information and Computation*, 211, pp. 106–137. doi:10.1016/j.ic.2011.12.003.
[8] Schwerhoff, M., Summers, A. J. (2015). 'Lightweight suppoert for magic wands in an automatic verifier', *In 29th European Conference on Object-Oriented Programming,*Volume 37, pp. 614-638. doi: 10.4230/LIPIcs.ECOOP.2015.614
[9] Dardinier, T. *et al.* (2022) 'Sound automation of magic wands', *Computer Aided Verification*, pp. 130–151. doi:10.1007/978-3-031-13188-2_7.
[10] Müller, P., Schwerhoff, M. and Summers, A.J. (2016) 'Automatic verification of iterated separating conjunctions using symbolic execution', *Computer Aided Verification*, pp. 405-425. doi:10.1007/978-3-319-41528-4_22.
[11] Heule, S. *et al.* (2013) 'Verification condition generation for permission logics with abstract predicates and abstraction functions', *ECOOP 2013 – Object-Oriented Programming*, pp. 451–476. doi:10.1007/978-3-642-39038-8_19.
[12] Dardinier, T., Müller, P. and Summers, A.J. (2022) 'Fractional Resources in unbounded separation logic', *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2), pp. 1066–1092. doi:10.1145/3563326.