Implementing Separation Logic using an SMT-backed Frame Rule

Kirill Golubev
Alcides Fonseca
alcides@ciencias.ulisboa.pt
LASIGE, Faculdade de Ciências da Universidade de Lisboa
Lisboa, Portugal

Abstract

Symbolic execution is a technique frequently used to reason about code. In symbolic execution, an analyzer keeps track of the program using a representation of the logical state, and validates that state transitions are valid. This verification is often discharged to SMT solvers as queries to the logical state.

Separation Logic is frequently used to express and verify the properties of programs with pointers or references. However, most SMT solvers (like the popular z3 [9]) do not support Separation Logic natively. Recently, the CVC5 SMT Solver has introduced partial support for separation logic, which has not yet been integrated into more high-level tools.

This work aims to address this gap, by providing a proof of concept for implementing the Frame Rule using SMT queries in the Symbolic Heap fragment of Separation Logic, as supported by CVC5. We conclude that this encoding can simplify the machinery dealing with separation logic, such as that present in Smallfoot or Verifast.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

ACM Reference Format:
Kirill Golubev and Alcides Fonseca. 2018. Implementing Separation Logic using an SMT-backed Frame Rule. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Formal verification is one of the few known ways to ensure that a computer program has the least amount of errors. The idea is to prove that a given program satisfies a specification provided in advance. The language of program specification is usually some logic that fits to describe program behavior. There are many ways to achieve this with varying resource requirements and automation degrees.

One technique employed to verify imperative programs is symbolic execution [2]. Usually, the engine for it is implemented separately for each tool [3, 6]. SMT solvers are used as oracles to verify that each step of the symbolic execution engine is correct.

Most programs in imperative languages are written in terms of heap manipulation. Separation Logic [11] is an extension of Hoare Logic [8], frequently used to reason about these types of programs. It introduces some new operations and constants to it alongside a new inference rule called *Frame Rule*. In general, Separation Logic is proved to be undecidable [5], but there are decidable subsets.

In this work, we focus on the *Symbolic Heap* fragment [3], as it enables a significant degree of automation by being supported in the CVC5 SMT Solver [1]. A simplified Symbolic Heap requires the logical context to be split into a pure boolean part ($BCtx$) and a pure spatial part, made of heaplets ($h_i = p \mapsto v$)

Todo ►*Change notation*◄

$$\mathtt{BCtx} \mid \mathtt{emp} \star \mathtt{h}_0 \star \ldots \star \mathtt{h}_n$$

where $\mathtt{h}_i = \mathtt{p} \mapsto \mathtt{v}$

This simplified version significantly restricts the expressive power of Separation Logic but still permits encoding of some interesting program properties. Thus, we chose it as a starting point, but CVC5 actually supports a larger fragment called GRASS [12]. In contrast, Z3 does not have any support for separation logic.

The goal of the present work is to provide an opportunity to shift some heavy lifting related to separation logic from a symbolic execution engine to an SMT solver. [1]

This is done by means of providing an algorithm to encode the frame rule through SMT solver queries.

2 Frame Rule

The algorithm uses the notion of SMT query that is denoted as follows.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1]But it was once prototyped: https://github.com/Z3Prover/z3/issues/811

$$\texttt{isUNSAT}(\neg\texttt{query}) = \texttt{true},$$

iff an SMT solver gives the UNSAT result on the negation of the query. This means that for all free variables negation of the query does not hold, which is equivalent to the situation when the query holds for all free variables.

$$\frac{\{\texttt{pre}\}\ \texttt{code}\ \{\texttt{post}\}}{\{\texttt{pre} \star \texttt{frame}\}\ \texttt{code}\ \{\texttt{post} \star \texttt{frame}\}}\ \text{Frame rule}$$

The algorithm itself is split into two phases.

- Check if the current context satisfies the precondition
- Apply postcondition to the larger context

Pseudocode for the first phase is quite simple. It exploits the idea, that it is possible to encode a heap containing any given one by adding $\star$ `true` to it.

The outline is that during the first step, it checks if the boolean context, defining pointer equivalence, and the heap imply precondition.

```
BCtx | H ⊩ pre
iff isUNSAT(¬(BCtx ∧ H  ⟹  pre ⋆ true))
```

The second phase exploits the same idea. The frame is inferred by checking each pointer for belonging in a frame, by precondition invalidation.

```
//BCtx | H - callsite context
//H = h_0 ⋆ ... ⋆ h_{n-1}
frame = H.filter(λ h.
            let c = pre ⋆ h ⋆ true
            in BCtx | H ⊩ c)
        .fold(emp, ⋆)
//BCtx | pre ⋆ frame - result
```

The unfortunate consequence of this approach is a performance hit. Usually, the systems that are using SMT solvers need only `O(1)` SMT queries to make the symbolic execution step, but this algorithm does it in `O(size(H))` queries.

If examined more closely, this algorithm is not exactly doing frame rule application, but rather heap reconstruction. It will discard every heaplet that invalidates precondition and the remainder will be the result. This makes it possible to use it for other purposes with slight variations. For example, for merging heaps after branching.

3 Conclusions

One big advantage of this approach is that the SMT solver algorithm for separation logic is decidable. In contrast to Viper [10] which is a main alternative to writing a symbolic execution engine from scratch.

The simplicity and decidability come with the cost of features that are possible to support. Viper is a much more mature and rich backend for the language, while the presented approach is capped by the capabilities of separation logic support in SMT solver. Said capabilities are defined by GRASS fragment of separation logic which looks like "propositional" separation logic. The main features that are kept unreachable by this limitation are recursive predicates and fractional permissions [4].

The primary target of this algorithm was Liquid Java [7], but it is general enough to be used in other projects relying on SMT solvers to verify symbolic execution steps. The primary benefit of this algorithm is simplicity and delegation of responsibility for separation logic handling to the SMT solver instead of a symbolic execution engine which is usually implemented separately for each tool.

We implemented and tested [2] this algorithm for Liquid Java, preliminary results are encouraging feature- and performance-wise. The prototype supports function calls, conditional branching, and assignments.

The performance degradation for synthetic benchmarks is around 30% relative to the pure boolean version of Liquid Java.

References

[1] Barbosa, H., Barrett, C. W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., and Zohar, Y. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I* (2022), D. Fisman and G. Rosu, Eds., vol. 13243 of *Lecture Notes in Computer Science*, Springer, pp. 415–442.

[2] Berdine, J., Calcagno, C., and O'Hearn, P. W. Symbolic execution with separation logic. In *Programming Languages and Systems: Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005. Proceedings 3* (2005), Springer, pp. 52–68.

[3] Berdine, J., Calcagno, C., and O'Hearn, P. W. Smallfoot: Modular automatic assertion checking with separation logic. In *Formal Methods for Components and Objects: 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures 4* (2006), Springer, pp. 115–137.

[4] Boyland, J. Checking interference with fractional permissions. In *Static Analysis: 10th International Symposium, SAS 2003 San Diego, CA, USA, June 11–13, 2003 Proceedings* (2003), Springer, pp. 55–72.

[5] Brotherston, J., and Kanovich, M. Undecidability of propositional separation logic and its neighbours. In *2010 25th Annual IEEE Symposium on Logic in Computer Science* (2010), IEEE, pp. 130–139.

[6] Distefano, D., and Parkinson J, M. J. jstar: Towards practical verification for java. *ACM Sigplan Notices 43*, 10 (2008), 213–226.

[7] Gamboa, C., Santos, P. A., Timperley, C. S., and Fonseca, A. User-driven design and evaluation of liquid types in java. *arXiv preprint arXiv:2110.05444* (2021).

[8] Hoare, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM 12*, 10 (1969), 576–580.

[9] Moura, L. d., and Bjørner, N. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems* (2008), Springer, pp. 337–340.

[2] https://github.com/CatarinaGamboa/liquidjava/pull/20

[10] Müller, P., Schwerhoff, M., and Summers, A. J. Viper: A verification infrastructure for permission-based reasoning. In *Verification, Model Checking, and Abstract Interpretation: 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings 17* (2016), Springer, pp. 41–62.

[11] O'Hearn, P. Separation logic. *Communications of the ACM 62*, 2 (2019), 86–95.

[12] Piskac, R., Wies, T., and Zufferey, D. Automating separation logic using smt. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25* (2013), Springer, pp. 773–789.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009