Implementing Separation Logic using an SMT-backed Frame Rule

Kirill Golubev
Alcides Fonseca
gkigorevich@ciencias.ulisboa.pt
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 state using its logic representation 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) 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: • **Theory of computation** → **Separation logic**; **Separation logic**; • **General and reference** → *Verification.*

ACM Reference Format:
Kirill Golubev and Alcides Fonseca. 2023. 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

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 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 [3]. Usually, the engine for it is implemented separately for each tool [2, 7]. In the context of this work, the state of the symbolic executor can be represented as a logical statement about the program. SMT solvers are used as oracles to verify that each transition of the symbolic execution engine's state is correct.

Most programs in imperative languages are written in terms of heap manipulation. Separation Logic [12] is an extension of Hoare Logic [9], 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 [2], 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 and a pure spatial part, made of disjoint heaplets. See Section 1 for details.

This simplified version significantly restricts the expressive power of Separation Logic but still permits the encoding of some interesting program properties. Thus, we chose it as a starting point, but CVC5 actually supports a larger fragment called GRASS [13][1]. In contrast, Z3[6] does not support separation logic. [2]

The goal of this work is to highlight the opportunity to shift some heavy lifting related to separation logic from a symbolic execution engine to an SMT solver. For this, we

[1] Actually, GRASS is a fragment of first-order logic which hosts the decidable fragment of separation logic. We will further use GRASS as a name for the decidable separation logic fragment for convenience.

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

$$\begin{array}{ll}
\texttt{x, y, ...} \in \mathit{Variables} & \text{variables} \\
\alpha,\ \beta,\ \ldots \in \mathit{Values} & \text{values} \\
\texttt{S := true | x} \mapsto \alpha & \text{simple spatial formula} \\
\Sigma \texttt{ := emp | S} \star \Sigma & \text{spatial formula} \\
\Pi \wedge \Sigma & \text{simplified symbolic heap}
\end{array}$$

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

Figure 1. Simplified symbolic heap fragment of separation logic

present an algorithm to encode the Frame Rule of Separation Logic (Section 1) through SMT Solver queries.

Related work.

- Smallfoot[2] is the first tool to implement verification with symbolic heap fragment of separation logic. Going through the paper one can see that the rules for handling separation logic are complex and there is value in delegating them to a dedicated tool, such as SMT solver.
- GRASShopper[14] is a tool, that implements GRASS fragment of separation logic for analyzing C-like language. In contrast to our approach the engine for analysis is implemented from scratch instead of relying on SMT solver.
- Verifast[10] consumes and produces chunks of the heap relying on SMT proofs of pointer equivalence, which should result in a multiplicative amount of SMT queries with respect to symbolic heap size and number of heap chunks in an assertion.

2 Frame Rule

Preliminaries. To prove that a formula with universally free variables holds, we query the SMT solver whether the negation of the query, using existentially quantified free variables, is unsatisfiable (`unsat`). We denote this verification as isUNSAT(¬query).

The frame rule application is treated here as if it is in the context of function application. It is general enough to apply the same procedure in other contexts. Figure 2 is an example from Liquid Java to illustrate the context.

Algorithm. The algorithm to check the Frame Rule is split into two phases:

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

The first step focus on verifying whether the precondition is guaranteed by the context. The context is composed of the pure boolean context - BCtx, defining pointer equivalence

```
//definition
@HeapPrecondition("x ↦ ()") //pre
@HeapPostcondition("_ ↦ ()") //post
Object foo(Object x){ ... }

//use site
void main(){
    //H_in = emp
    var a = new Object();// FR app
    //H_out = a ↦ ()

    //H_in = a ↦ ()
    var b = new Object();// FR app
    //H_out = a ↦ () * b ↦ ()

    //H_in = a ↦ () * b ↦ ()
    //H_in = pre * frame
    var r = foo(a); // FR app
    //H_out = r ↦ () * b ↦ ()
    //H_out = post * frame
}
```

Figure 2. Frame rule application in Liquid Java. The pure boolean context is discarded for readability

and the heap description - H_{in}. The goal is to prove that H_{in}, contains precondition modulo pointer equivalence.

$$\text{BCtx} \wedge H_{in} \Vdash_{SMT} \text{pre} \iff \text{isUNSAT}(\neg(\text{BCtx} \wedge H \implies \text{pre} * \text{true}))$$

The second step isolates the unchanged part of the heap, called frame, to be kept in the outgoing context, by discarding all heaplets that invalidate the precondition. In particular, for the incoming context $S_{in} = \text{BCtx} \wedge H_{in} = \text{BCtx} \wedge (\text{pre} * \text{frame})$, we will generate the outgoing context $S_{out} = \text{BCtx} \wedge H_{out} = \text{BCtx} \wedge (\text{post} * \text{frame})$.

$$\begin{aligned}
&H_{in} = h_1 * \ldots * h_n \\
&frame = \{h_i \mid i \in 1 \ldots n,\ \text{BCtx} \wedge H_{in} \Vdash_{SMT} \text{pre} * h_i * \text{true}\} \\
&H_{out} = \text{post} * \prod_{h_j \in frame} h_j = \text{post} * \text{frame}
\end{aligned}$$

Where $\prod$ is used with respect to separating conjunction. From this heap reconstruction, we can build the complete outgoing context S_{out}. The heap reconstruction process can be used in other contexts, such as merging heaps after branching.

When compared to other SMT-based verification procedures, our approach brings an overhead of `O(size(H))` SMT queries.

Evaluation. We validated the feasibility of this approach by implementing it in the Liquid Java compiler, supporting function calls, conditional branching, and assignments. The performance degradation for synthetic benchmarks is around 30% relative to the pure boolean version of Liquid Java.

Todo ►*maybe mention benchmarking details?*◄

3 Conclusions

The big advantage of this approach is that the SMT solver algorithm for separation logic is decidable, in contrast to Viper[11], which is a popular backend for implementing reasoning in some ownership logic. In contrast to GRASShopper[14], a verification project with a decidable inference algorithm, our approach provides a much more lightweight path to incorporating it into some existing tools. Another contribution of the present work is that it explores poorly utilized capabilities of native separation logic support in SMT solvers.

Todo ►*elaborate on "some existing tool"*◄

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 feature that is particularly challenging to support is fractional permissions [4].

While our validation was in the specific context of Liquid Java [8][3], 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.

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. 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* (2005), F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, Eds., vol. 4111 of *Lecture Notes in Computer Science*, Springer, pp. 115–137.

[3] 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.

[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), R. Cousot, Ed., vol. 2694 of *Lecture Notes in Computer Science*, Springer, pp. 55–72.

[5] Brotherston, J., and Kanovich, M. I. Undecidability of propositional separation logic and its neighbours. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom* (2010), IEEE Computer Society, pp. 130–139.

[6] de Moura, L. M., and Bjørner, N. S. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings* (2008), C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963 of *Lecture Notes in Computer Science*, Springer, pp. 337–340.

[7] Distefano, D., and Parkinson, M. J. jstar: towards practical verification for java. In *Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, October 19-23, 2008, Nashville, TN, USA* (2008), G. E. Harris, Ed., ACM, pp. 213–226.

[8] 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).

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

[10] Jacobs, B., Smans, J., Philippaerts, P., Vogels, F., Penninckx, W., and Piessens, F. Verifast: A powerful, sound, predictable, fast verifier for c and java. In *NASA formal methods symposium* (2011), Springer, pp. 41–55.

[11] 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* (2016), B. Jobstmann and K. R. M. Leino, Eds., vol. 9583 of *Lecture Notes in Computer Science*, Springer, pp. 41–62.

[12] O'Hearn, P. W. Separation logic. *Commun. ACM 62*, 2 (2019), 86–95.

[13] 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* (2013), N. Sharygina and H. Veith, Eds., vol. 8044 of *Lecture Notes in Computer Science*, Springer, pp. 773–789.

[14] Piskac, R., Wies, T., and Zufferey, D. Grasshopper: complete heap verification with mixed specifications. In *Tools and Algorithms for the Construction and Analysis of Systems: 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings 20* (2014), Springer, pp. 124–139.

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

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