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, the analyser keeps track of a logical representation of state, and correctness verifications are SMT queries. Separation logic is frequently used to express and verify properties of programs with pointers or references. However, most SMT solvers (like the popular z3) do not support Separation Logic natively. CVC5 has introduced partial support for separation logic, which has not yet been integrated into a 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, supported by CVC5. We conclude that this encoding can simplify the machinery dealing with separation logic, such as that present in Viper, Smallfoot, and others.

Todo ►*Viper is a debatable example, as it does not use separation logic internally. Instead it relies on more powerful mechanism of Implicit Dynamic Frames.*◄

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, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Todo ►*Introduce the topic of Program Verification.*◄

Todo ►*Introduce the topic of Separation Logic*◄

Todo ►*Mention tools that use separation logic, and describe how they are encoded.*◄

Todo ►*Describe the partial support of SL in CVC5, and identify that z3 has no such support.*◄

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.

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.

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

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.

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

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.

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


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

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 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 branches.

3 Conclusions

The primary target of this algorithm is Liquid Java, 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 this algorithm for Liquid Java, preliminary results are encouraging feature- and performance-wise.

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

References

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