Decision Procedures for Separation Logic Modulo Theories of Data

Radu IOSIF (VERIMAG, Grenoble)
mailto:Radu.Iosif@univ-grenoble-alpes.fr

1 Background

Separation Logic (SL) [12] is a logical framework for describing dynamically allocated mutable data structures generated by programs that use pointers and low-level memory allocation primitives. The logics in this framework are used by a number of academic (SPACE INVADER [2], SLEEK [9]), and industrial (INFER [3]) tools for program verification. The main reason for choosing to work within the SL framework is that it allows to write compositional proofs of programs, based on the principle of *local reasoning*:

> *To understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged [10].*

The main ingredients of SL are (i) the *separating conjunction* $\varphi * \psi$, which asserts that φ and ψ hold for separate portions of the heap, and (ii) its logical adjoint[1], the *magic wand* $\varphi \mathrel{-\!\!*} \psi$, asserting that each disjoint extension of the heap, with a model of φ, results in a model of ψ. Consider for instance, the heap consisting of two memory cells, pointed to by the program variables x and y, such that x cell has an outgoing selector field to the y cell, and viceversa. The heap can be split into two disjoint parts, each containing exactly one cell, and described by an atomic proposition $\mathsf{x} \mapsto \mathsf{y}$ and $\mathsf{y} \mapsto \mathsf{x}$, respectively. This heap is described by the formula $\mathsf{x} \mapsto \mathsf{y} * \mathsf{y} \mapsto \mathsf{x}$, which reads x *points to* y *and separately* y *points to* x.

When reasoning about programs that manipulate data structures, it is crucial to have the ability of describing infinite sets of heaps, that are instances of recursively defined data structures, such as singly- or doubly-linked lists, trees and such. This is achieved in SL by introducing *inductive definitions*. For instance, the inductive definition below defines singly-linked list segments from head to tail:

$$\begin{array}{lcll} \mathsf{ls}(\mathsf{head},\mathsf{tail}) & \Leftarrow & \mathsf{emp} \wedge \mathsf{head} = \mathsf{tail} & (\mathsf{r}_1) \\ \mathsf{ls}(\mathsf{head},\mathsf{tail}) & \Leftarrow & \exists \mathsf{x} \,.\, \mathsf{head} \mapsto \mathsf{x} * \mathsf{ls}(\mathsf{x},\mathsf{tail}) & (\mathsf{r}_2) \end{array}$$

The first case (r_1) of the definition of the predicate ls is the *base rule*, i.e. the heap is empty (emp) and head equals tail, while the second case (r_2) is the *inductive rule* which corresponds to one unfolding of the definition. The separating conjunction here states that the cell pointed to by head is disjoint from the rest of the heap, which is a list segment from x to tail.

An important decision problem in SL concerns the validity of *entailments* $\phi \models \psi$: given formulæ ϕ and ψ does every heap that satisfies ϕ also satisfy ψ ? For instance, the entailment problem $\mathsf{ls}(\mathsf{x},\mathsf{y}) * \mathsf{ls}(\mathsf{y},\mathsf{z}) \models \mathsf{ls}(\mathsf{x},\mathsf{z})$ asks whether the concatenation of two list segments is again a list segment.

The expressive power of SL comes with the inherent difficulty of automatically reasoning about the satisfiability of its formulae, as required by push-button program analysis tools. Indeed, SL becomes undecidable in the presence of first-order quantification, even when the fragment uses only points-to predicates, without the separating conjunction or the magic wand [4]. Undecidability occurs also when inductive definitions are used on top of the existential fragment of SL [8].

2 Combining Separation Logic with Data Theories

The purpose of this internship is to study the combined theory of SL with inductive predicates and data, needed to reason about the contents of data structures in the memory. For instance, we can consider a variant of the ls predicate that specifies sorted list segments:

$$\begin{array}{lcll} \mathsf{sls}(\mathsf{head},\mathsf{d},\mathsf{tail}) & \Leftarrow & \mathsf{emp} \wedge \mathsf{head} = \mathsf{tail} & (\mathsf{r}_1) \\ \mathsf{sls}(\mathsf{head},\mathsf{d},\mathsf{tail}) & \Leftarrow & \exists \mathsf{x} \exists e \,.\, \mathsf{head} \mapsto (d,\mathsf{x}) * \mathsf{ls}(\mathsf{x},\mathsf{e},\mathsf{tail}) * d < e & (\mathsf{r}_2) \end{array}$$

[1] We have $\varphi * \psi \Rightarrow \phi$ if and only if $\varphi \Rightarrow \psi \mathrel{-\!\!*} \phi$.

Here the atomic proposition $\mathsf{head} \mapsto (d, \mathsf{x})$ means that head is a pointer to a record whose first field is the data value d and second field is a pointer to the tail x of the list segment. Note also that the data elements are recursively required to increase strictly, by the atomic proposition $d < e$, belonging to the theory of data.

Reasoning about combined theories of separation and data is difficult mainly because the effect of adding a data theory to existing decidable fragments of SL [7, 6] is not yet fully understood. Roughly speaking, combining inductive reasoning with data requires invariant synthesis techniques that resemble the fixed point computations used in program analysis [5].

During this internship you are expected to contribute to the study of this problem. Another line of work is more practical and consists in integrating state-of-the-art decision procedures for SL (see e.g. [11, 6]) with Satisfiability Modulo Theories (SMT) solvers, such as CVC4 [1].

3 How to Apply

Send your CV and a transcript of your university grades to:

`mailto:Radu.Iosif@univ-grenoble-alpes.fr`

References

[1] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Computer Aided Verification (CAV)*. Springer, 2011.

[2] J. Berdine, C. Calcagno, B. Cook, D. Distefano, P. O'Hearn, T. Wies, and H. Yang. Shape analysis for composite data structures. In *Proc. CAV'07*, volume 4590 of *LNCS*. Springer, 2007.

[3] C. Calcagno and D. Distefano. Infer: An automatic program verifier for memory safety of c programs. In *Proc. of NASA Formal Methods'11*, volume 6617 of *LNCS*. Springer, 2011.

[4] C. Calcagno, H. Yang, and P. W. OHearn. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS 2001*, pages 108–119. Springer, 2001.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[6] M. Echenim, R. Iosif, and N. Peltier. Decidable entailments in separation logic with inductive definitions: Beyond established systems, 2020.

[7] R. Iosif, A. Rogalewicz, and J. Simacek. The tree width of separation logic with recursive definitions. In *Proc. of CADE-24*, volume 7898 of *LNCS*, 2013.

[8] R. Iosif, A. Rogalewicz, and T. Vojnar. Deciding entailments in inductive separation logic with tree automata. In F. Cassez and J. Raskin, editors, *ATVA 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2014.

[9] H. H. Nguyen and W.-N. Chin. Enhancing program verification with lemmas. In *Proc of CAV'08*, volume 5123 of *LNCS*. Springer, 2008.

[10] P. W. O'Hearn, J. C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, CSL '01, pages 1–19, 2001.

[11] J. Pagel and F. Zuleger. Beyond symbolic heaps: Deciding separation logic with inductive definitions. In *LPAR-23*, volume 73 of *EPiC Series in Computing*, pages 390–408. EasyChair, 2020.

[12] J. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.