Decision Procedures for Inductive Separation Logic Modulo Data Theories

Nicolas PELTIER (LIG, Grenoble) and Radu IOSIF (VERIMAG, Grenoble)
{Nicolas.Peltier,Radu.Iosif}@univ-grenoble-alpes.fr

1 Context and Motivation

Separation Logic (SL) [10] 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 [1], SLEEK [8], VERIFAST [7]), and industrial (INFER [2]) tools capable of finding, as well as proving the absence of, incorrect low-level memory manipulations, such as: null/non-allocated pointer dereferences, memory leaks, unaligned and out-of-bounds accesses, etc.

The advantage of SL logics over related approaches, such as concolic testing [6] or classical shape analysis [11] is the ability of scaling up to industrial-size codebases leveraging from *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 [9].*

Reasoning about programs that manipulate pointer-linked data structures requires the ability to describe common patterns of the structures used by programmers, such as: various types of lists (singly-linked, doubly-linked, skip-lists, etc.), trees (red-black trees, splay trees, etc.) arrays and hash tables. It is standard practice among programmers to use *inductive definitions* to define recursive datatypes. For instance, the inductive definition below defines singly-linked list segments:

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

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 logical connective $*$ 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. By changing (r_2) into $\mathsf{ls}(\mathsf{head},\mathsf{tail}) \Leftarrow \exists \mathsf{x} \;.\; \mathsf{head} \mapsto \mathsf{x} * \mathsf{ls}(\mathsf{x},\mathsf{tail}) \wedge \mathsf{head} \neq \mathsf{tail}$, we specify acyclic lists, whose tail pointer does not refer to the address of a cell inside the list.

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 disjoint list (acyclic) segments is again a (acyclic) 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, the problem of entailment between formulæ using inductively defined predicates is undecidable, in general. Although decidability, within elementary recursive complexity bounds, can be recovered for standard SL, by considering natural syntactic restrictions on the form of the inductive definitions [3, 4, 5], several problems remain unanswered:

- What is the impact (in terms of computational complexity) of reasoning about the data stored in a structure, by combining SL with theories of data (integers, reals, strings, sets, etc.)? Are there interesting data theories that can be embedded within inductive definitions of data structures, while preserving decidability (with reasonable complexity)?
- Do there exist proof calculi, based on induction (noetherian, infinite descent, etc.), that are complete for a given decidable entailment problem? In other words, can one generate a (human or machine) checkable explanation for every valid entailment?
- For which theories and logical fragments can those calculi be turned into efficient (e.g., polynomial-time) decision procedures?

Answering such questions is an important step for improving the state of the art of existing and building a new generation of inductive solvers for SL.

2 Tasks and Goal

The PhD candidate will take active part in the following tasks: 1. the integration of SL with theory reasoning, 2. the development of formal proof calculi for fragments of SL with inductive definitions, and 3. the implementation of the procedure and experimentation. The exact ratio between theoretical study and implementation will depend on the profile and background of the candidate, and will be determined at the beginning of the project.

The goal of this PhD project is to devise a procedure that is optimal from a theoretical point of view, practically efficient, and able to handle a class of definitions that is as large as possible, combining spatial reasoning (to reason on the shape of the considered data structures) with theory reasoning based on external tools (to take into account the properties of the data stored inside the structure). The procedure should be able to output a proof certificate when the considered entailment holds (to be checked by some external software such as Coq or Isabelle) and a counter-model when it is not valid. To ensure efficiency and maintainability, the implementation will be developed using some modern efficient programming language such as C++ or OCaml. Experimentations will be conducted, using in particular the benchmarks developed for the SL Competition `https://sl-comp.github.io/`, to which homegrown test cases will be added.

3 Hosting Laboratories

The work of this PhD project will be carried out jointly within the following academic laboratories, located in the same building on the Campus of Grenoble:

1. **LIG** (*Laboratoire d'Informatique de Grenoble*, UMR 5217) `https://www.liglab.fr/` is one of the largest public laboratories in Computer Science in France. Its mission is to contribute to the development of fundamental aspects of Computer Science (models, languages, methodologies, algorithms) and address conceptual, technological, and societal challenges. The applicant will work in the CAPP team: `http://capp.imag.fr/`.
2. **VERIMAG** (UMR 5104) `https://www-verimag.imag.fr/` focuses on theoretical and practical aspects of formal methods for embedded system development. Since its creation, in 1993, VERIMAG has a proven record in basic theoretical research and in development of tools for system verification.

The PhD title will be conferred by the Université de Grenoble Alpes (UGA). The sucessful applicant will be officially registered as an UGA graduate student during the period of the thesis.

4 How to Apply

The applicant should have a master's degree in theoretical computer science. Send your CV and a transcript of your university grades to:

{`Nicolas.Peltier,Radu.Iosif`}`@univ-grenoble-alpes.fr`

References

[1] 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.

[2] 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.

[3] M. Echenim, R. Iosif, and N. Peltier. Checking entailment between separation logic symbolic heaps: Beyond connected and established systems. *CoRR*, abs/2012.14361, 2020.

[4] M. Echenim, R. Iosif, and N. Peltier. Entailment checking in separation logic with inductive definitions is 2-exptime hard. In *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 191–211. EasyChair, 2020.

[5] M. Echenim, R. Iosif, and N. Peltier. Decidable entailments in separation logic with inductive definitions: Beyond establishment. In *CSL 2021: 29th International Conference on Computer Science Logic*, EPiC Series in Computing. EasyChair, 2021.

[6] P. Godefroid, N. Klarlund, and K. Sen. Dart: Directed automated random testing. In *Proceedings of PLDI'05*, page 213–223, New York, NY, USA, 2005. Association for Computing Machinery.

[7] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. Verifast: A powerful, sound, predictable, fast verifier for c and java. In *NASA Formal Methods*, pages 41–55, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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

[9] 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.

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

[11] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *Proceedings of POPL'99*, page 105–118, New York, NY, USA, 1999. Association for Computing Machinery.