Decision Procedures for Separation Logic

Deliverable D1-1

ANR project VECOLIB

October 2015

This deliverable presents the decision procedures developed inside the VECOLIB project for different fragments of Separation Logic.

The first procedure, implemented in the SLIDE solver, decides the validity of an entailment between formulas in a fragment of Separation Logic with Inductive Definitions (SLID). This fragment is very expressive and covers interesting data structures (lists, trees, trees with parent references). It proceeds by reduction to the inclusion problem in tree automata. The procedure is implemented in the solver SLIDE.

The second procedure, implemented in the SPEN solver, soundly decides (it is not complete) the validity of the entailment between formulas in SLID extended with data. For the fragment without data, the procedure proceeds by reduction to the membership problem in tree automata. It is able to deal with inductive definitions for list-like data structures (nested lists, skip lists, etc.) For the fragment with data, the procedure proceeds by proof search using a set of lemmas automatically generated from the inductive definition. Both procedures are implemented in the solver SPEN.

Finally, we describe a DPLL-based SMT solver for the quantifier-free fragment of Separation Logic with data and without inductive definitions. This procedure is based on a translation of SL to multi-sorted first-order logic with bounded quantifiers over sets and uninterpreted functions. We deal with the quantifiers in an effective way, by a counterexample-driven instantiation algorithm. We implemented the algorithm as a branch of the CVC4 SMT solver.

We first provide an overview of the logic fragments considered and the related works on the existing decision procedures.

Contents

1 Preliminaries and Related Works

Separation Logic (SL) [19, 22] is an established approach for the specification of complex, unbounded size data structures that are used in the implementation of containers. It is used by several verification and analysis tools, e.g., SmallFoot [26], MemCAD/Xisa [8], Infer [6], Celia [2], Sleek/Hip [9], Verifast [17]. This success is due to its support for local reasoning based on the "frame rule" which leads to compact proofs and scalable analyses. Typically, SL is used in combination with *inductive definitions*, which provide a natural description of the data structures manipulated in containers, e.g., lists, trees. Moreover, it can be extended to specify low level code data structures [7].

Inside the VECOLIB project, we developed two solvers for SL: SLIDE [16] and SPEN [11, 12]. These decision procedures are or should be used in the tools developed inside the project for the verification of containers.

In this section, we first fix a logic fragment of SL which subsumes the fragments considered by both solvers. Then, we provide an overview of the existing solvers developed for SL.

1.1 Separation Logic with inductive definitions

Let $\mathtt{LVar}$ be a set of *location variables*, interpreted as heap locations, and $\mathtt{DVar}$ a set of *data variables*, interpreted as data values stored in the heap, (multi)sets of values, etc. In addition, let $\mathtt{Var} = \mathtt{LVar} \cup \mathtt{DVar}$. The domain of heap locations is denoted by $\mathbb{L}$ while the domain of data values stored in the heap is generically denoted by $\mathbb{D}$. Let $\mathcal{F}$ be a set of pointer fields, interpreted as functions $\mathbb{L} \rightharpoonup \mathbb{L}$, and $\mathcal{D}$ a set of data fields, interpreted as functions $\mathbb{L} \rightharpoonup \mathbb{D}$. The syntax of the Separation Logic fragment we considered is defined in Tab. 1.

Formulas are interpreted over pairs (s,h) formed of a *stack* s and a *heap* h. The stack s is a function giving values to a finite set of variables (location or data variables) while the heap h is a function mapping a finite set of pairs (ℓ, pf), where ℓ is a location and pf is a pointer field, to locations, and a finite set of pairs (ℓ, df), where df is a data field, to values in $\mathbb{D}$. In addition, h satisfies the condition that for each $\ell \in \mathbb{L}$, if $(\ell, df) \in \mathtt{dom}(h)$ for some $df \in \mathcal{D}$, then $(\ell, pf) \in \mathtt{dom}(h)$ for some $pf \in \mathcal{F}$. Let $\mathtt{dom}(h)$ denote the domain of h, and $\mathtt{ldom}(h)$ denote the set of $\ell \in \mathbb{L}$ such that $(\ell, pf) \in \mathtt{dom}(h)$ for some $pf \in \mathcal{F}$.

Formulas are conjunctions between a pure formula Π and a spatial formula Σ. Pure formulas characterize the stack s using (dis)equalities between location variables, e.g., a stack models $x = y$ iff $s(x) = s(y)$, and constraints Δ over data variables. We let Δ unspecified, though we assume that they belong to decidable theories, e.g., linear arithmetic or quantifier-free first order theories over multisets of values. The atom *emp* of spatial formulas holds iff the domain of the heap is empty. The *points-to atom* $E \mapsto \{(f_i, x_i)\}_{i \in I}$ specifies that the heap contains exactly one location E, and for all $i \in I$, the field f_i of E equals x_i, i.e., $h(s(E), f_i) = s(x_i)$. The *predicate atom* $P(E, \vec{F})$ specifies a heap segment rooted at E and shaped by the predicate P; the fragment is parameterized by a set $\mathcal{P}$ of *inductively defined predicates*, formally defined hereafter.

Let $P \in \mathcal{P}$. An *inductive definition* of P is a finite set of rules of the form $P(E, \vec{F}) ::= \exists \vec{Z}. \Pi \wedge \Sigma$, where $\vec{Z} \in \mathtt{Var}^*$ is a tuple of variables. A rule R is called a *base rule* if Σ contains no predicate atoms. Otherwise, it is called an *inductive rule*.

1.2 Existing solvers for Separation Logic

Several teams are developing SL solvers for different fragments included in the one presented above. Some of these solvers, including SLIDE and SPEN, have participated to competition of SL-COMP'14

Table 1: The syntax of the Separation Logic fragment

$X, Y, E \in \mathtt{LVar}$ location variables	$\rho \subseteq (\mathcal{F} \times \mathtt{LVar}) \cup (\mathcal{D} \times \mathtt{DVar})$
$\vec{F} \in \mathtt{Var}^*$ vector of variables	$P \in \mathcal{P}$ predicates
$x \in \mathtt{Var}$ variable	Δ formula over data variables
$\Pi ::= X = Y \mid X \neq Y \mid \Delta \mid \Pi \wedge \Pi$	pure formulas
$\Sigma ::= emp \mid E \mapsto \rho \mid P(E, \vec{F}) \mid \Sigma * \Sigma$	spatial formulas
$\varphi ::= \Pi \wedge \Sigma \mid \varphi \vee \varphi \mid \exists x.\ \varphi$	formulas

[23], organized after the model and with the help of the SMT-COMP, the competition of solvers on SAT modulo theory.

ASTERIX [20]: is developed by Juan Navarro Perez (UCL, UK) and Andrey Rybalchenko (Microsoft Research Cambridge, UK). The solver deals with the satisfiability and entailment checking in the SL fragment using only `ls` predicate. For this, it implements a model-based approach. The procedure relies on SMT solving technology (Z3 solver is used) to untangle potential aliasing between program variables. It has at its core a matching function that checks whether a concrete valuation is a model of the input formula and, if so, generalizes it to a larger class of models where the formula is also valid.

SeLoger [14]: is developed by Christophe Hasse (LSV, France). The solver deals with the satisfiability and entailment checking in the SL fragment using only `ls` predicate. Is uses a special procedure based on searching paths in graphs to untangle potential aliasing between program variables.

CYCLIST-SL [5, 10]: is developed by James Brotherston and Nikos Gorogiannis (Middlesex University London, UK). The solver deals with the entailment checking for the SL fragment without data. It is an instantiation of the theorem prover CYCLIST for the case of Separation Logic with inductive definitions. The solver builds derivation trees and uses induction to cut infinite paths in these trees that satisfy some soundness condition. For the Separation Logic, CYCLIST-SL replaces the rule of weakening used in first-order theorem provers with the frame rule of SL.

SLEEK [9, 24]: is developed in the team of Wei Ngan Chin (NUS, Singapore). The solver deals with the satisfiability and entailment checking for SL formulas in the fragment we presented. It is an (incomplete but) automatic prover, that builds a proof tree for the input problem by using the classical inference rules and the frame rule of SL. It also uses a database of lemmas for the inductive definitions in order to discharge the proof obligations on the spatial formulas. The proof obligations on pure formulas are discharged by external provers like CVC4, Mona, or Z3.

SLSAT [4]: is developed by James Brotherston, Nikos Gorogiannis (Middlesex University London, UK), and Juan Navarro Perez (UCL, UK). The solver deals with the satisfiability problem for the SL fragment without data but with general inductively defined predicates. The decision procedure is based on a fixed point computation of a constraint, called the "base" of an inductive predicate definition. This constraint is a conjunction of equalities and dis-equalities between a set of free variables built also by the fixed point computation from the set of inductive definitions.

GRASShopper [21] is developed by Ruzica Piskac, Thomas Wies, and Damien Zufferey (NYU, USA). The solver deals with the satisfiability and entailment problem for the SL fragment with data but mainly list and trees predicates. The decision procedure is based on the translation of the problem into a equi-satisfiable problem in the theory of sets and the the application of existing solvers for the set theory.

Table 2: Examples of inductive definitions without data

singly linked lists

$$\texttt{ls}(E,F) \;::=\; (E = F \wedge emp) \tag{1}$$

$$::= \; (E \neq F \wedge \exists X. E \mapsto \{(\mathtt{nxt},X)\} \,*\, \texttt{ls}(X,F)) \tag{2}$$

nested linked lists

$$\texttt{nll}(E,F,B) \;::=\; (E = F \wedge emp) \tag{3}$$

$$::= \; (E \neq F \wedge E \neq B \wedge \exists X,Z.\; E \mapsto \{(\mathtt{nxtup},X),(\mathtt{inner},Z)\} * \texttt{ls}(Z,B) \,*\, \texttt{nll}(X,F,B)) \tag{4}$$

doubly linked lists

$$\texttt{dll}(E,L,P,F) \;::=\; (E = F \wedge L = P \wedge emp) \tag{5}$$

$$::= \; \big(E \neq F \wedge L \neq P \wedge \exists X. E \mapsto \{(\mathtt{nxt},X),(\mathtt{prv},P)\} \,*\, \texttt{dll}(X,L,E,F)\,\big) \tag{6}$$

binary tree

$$\texttt{btree}(E) \;::=\; (E = \texttt{nil} \wedge emp) \tag{7}$$

$$::= \; \big(E \neq \texttt{nil} \wedge \exists X,Y.\; E \mapsto \{(\mathtt{lson},X),(\mathtt{rson},Y)\} * \texttt{btree}(X) \,*\, \texttt{btree}(Y)\big) \tag{8}$$

tree with linked leaves

$$\texttt{tll}(R,P,E,F) \;::=\; (R = E \wedge R \mapsto \{(\mathtt{lson},\texttt{nil}),(\mathtt{rson},\texttt{nil}),(\mathtt{parent},P),(\mathtt{nxt},F)\}) \tag{9}$$

$$::= \; \big(R \neq E \wedge \exists X,Y,Z.\; R \mapsto \{(\mathtt{lson},X),(\mathtt{rson},Y),(\mathtt{parent},P),(\mathtt{nxt},Z)\} * \texttt{tll}(X,R,E,Z) \,*\, \texttt{tll}(Y,R,Z,F)\big) \tag{10}$$

Table 3: Examples of inductive definitions with data

binary search tree

$$\texttt{bst}(E,M) ::= E = \texttt{nil} \wedge M = \emptyset \wedge emp \tag{11}$$

$$\texttt{bst}(E,M) ::= \exists X,Y,M_1,M_2,v. E \mapsto \{(\mathtt{lson},X),(\mathtt{rson},Y),(\mathtt{data},v)\} * \texttt{bst}(X,M_1) * \texttt{bst}(Y,M_2) \wedge M = \{v\} \cup M_1 \cup M_2 \wedge M_1 < v < M_2 \tag{12}$$

$$\tag{13}$$

binary search tree with a hole

$$\texttt{bsthole}(E,M_1,F,M_2) ::= E = F \wedge M_1 = M_2 \wedge emp \tag{14}$$

$$\texttt{bsthole}(E,M_1,F,M_2) ::= \exists X,Y,M_3,M_4,v. E \mapsto \{(\mathtt{lson},X),(\mathtt{rson},Y),(\mathtt{data},v)\} * \texttt{bst}(X,M_3) * \texttt{bsthole}(Y,M_4,F,M_2) \wedge M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4 \tag{15}$$

$$\texttt{bsthole}(E,M_1,F,M_2) ::= \exists X,Y,M_3,M_4,v. E \mapsto \{(\mathtt{lson},X),(\mathtt{rson},Y),(\mathtt{data},v)\} * \texttt{bsthole}(X,M_3,F,M_2) * \texttt{bst}(Y,M_4) \wedge M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4 \tag{16}$$

2 Slide Solver

Slide [16, 25]: developed by Adam Rogalewicz and Tomas Vojnar (VeriFIT, Czech Rep) and Radu Iosif (Verimag, France). The solver deals with the entailment problem in the decidable sub-fragment of SL without data constraints defined in [15]. The decision method implemented in Slide belongs to the class of *automata-theoretic* decision techniques. We translate an entailment problem $\varphi \models \psi$ into a language inclusion problem $\mathcal{L}(A_\varphi) \subseteq \mathcal{L}(A_\psi)$ for tree automata (TA) A_φ and A_ψ that (roughly speaking) encode the sets of models of φ and ψ, respectively. Yet, a naïve translation of the inductive definitions of SL into TA encounters a *polymorphic representation* problem: the same set of structures can be defined in several different ways, and TA simply mirroring the definition will not report the entailment. For example, DLLs with selectors `next` and `prev` for the next and previous nodes, respectively, can be described by a forward unfolding of the inductive definition: $\mathtt{DLL}(head, prev, tail, next) \equiv \exists x.\ head \mapsto (x, prev) * \mathtt{DLL}(x, head, tail, next) \mid emp \wedge head = tail \wedge prev = next$, as well as by a backward unfolding of the definition: $\mathtt{DLL}_{rev}(head, prev, tail, next) \equiv \exists x.\ tail \mapsto (next, x) * \mathtt{DLL}_{rev}(head, prev, x, tail) \mid emp \wedge head = tail \wedge prev = next$. Also, one can define a DLL starting with a node in the middle and unfolding backward to the left of this node and forward to the right: $\mathtt{DLL}_{mid}(head, prev, tail, next) \equiv \exists x, y, z\ .\ \mathtt{DLL}(y, x, tail, next) * \mathtt{DLL}_{rev}(head, prev, z, x)$. The circular entailment: $\mathtt{DLL}(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}) \models \mathtt{DLL}_{rev}(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}) \models \mathtt{DLL}_{mid}(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}) \models \mathtt{DLL}(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d})$ holds, but a naïve structural translation to TA might not detect this fact. To bridge this gap, we define a closure operation on TA, called *canonical rotation*, which adds all possible representations of a given inductive definition, encoded as a tree automaton.

The translation from SL to TA provides also tight complexity bounds, showing that entailment in the local fragment of SL with inductive definitions is EXPTIME-complete. Moreover, we implemented our method using the Vata [18] tree automata library, which leverages from recent advances in non-deterministic language inclusion for TA [3], and obtained quite encouraging experimental results.

3 Spen Solver

Spen [11, 27]: developed by Constantin Enea and Mihaela Sighireanu (UPD & CNRS, France), Ondra Lengal and Tomas Vojnar (VeriFIT, Czech Rep). The solver deals with satisfiability and entailment problems for the fragment SL with special form of ID and data constraints. The decision procedures calls the MiniSAT solver on a boolean abstraction of the SL formulas to check their satisfiability and to "normalize" the formulas by inferring its implicit (dis)equalities. The core of the algorithm checking if $\varphi \Rightarrow \psi$ is valid searches a mapping from the atoms of ψ to sub-formulas of φ. This search uses the membership test in tree automata to recognize in sub-formulas of φ some unfolding of the inductive definitions used in ψ.

In 2015, this solver has been extended to allow more general inductive definitions (including trees) combined with data constraints [12]. The inductive definitions should belong to a class formally defined in [12] that (i) supports simple lemmas and (ii) allows to **automatically synthesize** these lemmas using efficiently checkable, almost syntactic, criteria. The solver is based on a proof strategy using the lemmas generated from the inductive definitions. The proof strategy is based on simple syntactic matchings of spatial atoms (points-to atoms or predicate atoms like `bsthole`) and reductions to SMT solvers for dealing with the data constraints. We shown experimentally that this proof strategy is powerful enough to deal with sophisticated benchmarks, e.g., the verification conditions generated from the iterative procedures for searching, inserting, or deleting elements in binary search trees, red-black trees, and AVL trees, in a very efficient way.

4 Cvc4-SL Solver

We present a decision procedure for quantifier-free SL which is entirely parameterized by a base theory T of heap locations and data, i.e. the sorts of memory addresses and their contents can be chosen from a large variety of available theories handled by Satisfiability Modulo Theories (SMT) solvers, such as linear integer (real) arithmetic, strings, sets, uninterpreted functions, etc. Given a base theory T, we call $SL(T)$ the set of separation logic formulae built on top of T, by considering points-to predicates and the separation logic connectives. Applications of our procedure include:

- Integration with more sophisticated theorem provers for separation logic with inductive predicates. Currently, such solvers concentrate on aspects related to applying induction efficiently and apply heavy restrictions on the ground fragment of the logic considered (typically only separating conjunctions combined with very restricted data theories).
- Use as back-end of a bounded model checker for programs with pointer and data manipulations, based on a complete weakest pre-condition calculus that involves the magic wand connective.

As main contribution, we show that quantifier-free $SL(T)$ is decidable, provided that the quantifier-free fragment of the base theory T is decidable. Our method is based on a semantics-preserving translation of $SL(T)$ into first-order T formulae with quantifiers over a domain of sets, whose cardinality is bound by the size of the input formula. For the fragment of T formulae produced by the translation from $SL(T)$, we developped a lazy quantifier instantiation method, based on counterexample-driven refinement. We show that the quantifier instantiation algorithm is sound complete and terminates on the fragment under consideration. We present our algorithm for the satisfiability of quantifier-free $SL(T)$ logics as a component of a DPLL(T) architecture [13], which is widely used by modern SMT solvers. We have implemented a prototype solver as a branch of the CVC4 SMT solver [1] and carried out experiments that handle non-trivial examples quite effectively.

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] A. Bouajjani, C. Dragoi, C. Enea, and M. Sighireanu. On inter-procedural analysis of programs with lists and data. In *PLDI*, pages 578–589. ACM, 2011.

[3] A. Bouajjani, P. Habermehl, L. Holik, T. Touili, and T. Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In *Proc. of CIAA*, volume 5148 of *LNCS*. Springer, 2008.

[4] J. Brotherston, C. Fuhs, J. A. N. Pérez, and N. Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates. In *CSL-LICS*, pages 25:1–25:10. ACM, 2014.

[5] J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In *APLAS*, volume 7705 of *LNCS*, pages 350–367. Springer, 2012.

[6] C. Calcagno and D. Distefano. Infer: An automatic program verifier for memory safety of c programs. In *NASA FM*, volume 6617 of *LNCS*, pages 459–465. Springer, 2011.

[7] C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang. Beyond reachability: Shape abstraction in the presence of pointer arithmetic. In *SAS*, volume 4134 of *LNCS*, pages 182–203. Springer, 2006.

[8] B.-Y. Chang and X. Rival. Relational inductive shape analysis. In *POPL'08*, pages 247–260. ACM, 2008.

[9] W.-N. Chin, C. David, H. H. Nguyen, and S. Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Sci. Comput. Program.*, 77(9):1006–1036, 2012.

[10] CYCLIST. `https://github.com/ngorogiannis/cyclist`.

[11] C. Enea, O. Lengál, M. Sighireanu, and T. Vojnar. Compositional entailment checking for a fragment of separation logic. In *APLAS*, volume 8858 of *LNCS*, pages 314–333. Springer, 2014.

[12] C. Enea, M. Sighireanu, and Z. Wu. *Automated Technology for Verification and Analysis: 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, chapter On Automated Lemma Generation for Separation Logic with Inductive Definitions, pages 80–96. Springer International Publishing, Cham, 2015.

[13] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Dpll (t): Fast decision procedures. In *Computer aided verification*, pages 175–188. Springer, 2004.

[14] C. Haase, S. Ishtiaq, J. Ouaknine, and M. J. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *CAV*, volume 8044 of *LNCS*, pages 790–795. Springer, 2013.

[15] R. Iosif, A. Rogalewicz, and J. Simácek. The tree width of separation logic with recursive definitions. In *CADE*, volume 7898 of *LNCS*, pages 21–38. Springer, 2013.

[16] R. Iosif, A. Rogalewicz, and T. Vojnar. Deciding entailments in inductive separation logic with tree automata. In *ATVA*, volume 8837 of *LNCS*, pages 201–218. Springer, 2014.

[17] B. Jacobs and F. Piessens. The verifast program verifier. Technical Report CW-520, Department of Computer Science, Katholieke Universiteit Leuven, 2008.

[18] O. Lengal, J. Simacek, and T. Vojnar. Vata: a tree automata library.
URL: `http://www.fit.vutbr.cz/research/groups/verifit/tools/libvata/`.

[19] P. W. O'Hearn, J. C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.

[20] J. A. N. Pérez and A. Rybalchenko. Separation logic modulo theories. In *APLAS*, volume 8301 of *LNCS*, pages 90–106. Springer, 2013.

[21] R. Piskac, T. Wies, and D. Zufferey. Automating separation logic using SMT. In *CAV*, volume 8044 of *LNCS*, pages 773–789. Springer, 2013.

[22] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

[23] M. Sighireanu and D. Cok. Report on SL-COMP'14. *JSAT, Journal of Satisfiability*, 1, 2014.

[24] SLEEK. http://loris-7.ddns.comp.nus.edu.sg/~project/s2/beta/.

[25] SLIDE. http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/.

[26] SmallFoot. http://www0.cs.ucl.ac.uk/staff/p.ohearn/smallfoot/.

[27] SPEN. https://www.github.com/mihasighi/spen.