Testing the Satisfiability of Formulas in Separation Logic with Permissions

N. Peltier

Univ. Grenoble Alpes, CNRS, LIG – ANR Project NARCO

TABLEAUX 23
Prague, Czech Republic, September 18-21, 2023

Separation Logic

- A logic used in program verification to reason on mutable data structures (pointers)
- Introduced in 2000 (Reynolds, O'Hearn, Ishtiaq, Yang), based on earlier work by Burstall, O'Hearn and Pym
- Now (since about 2009) used in industrial static analyzers (e.g., Facebook Infer, Microsoft SLAyer etc.)
- Facilitate *modular reasoning*
  - Express key properties in a more natural and concise way
  - Enable local reasoning
  - Separating conjunction : assert disjointness of memory blocks

Separation Logic : Ingredients

- Points-to atoms of the form $x \mapsto (y_1, \ldots, y_k)$

  "Location (i.e., memory address) x is the only allocated location *and points to the tuples of locations $y_1, \ldots, y_k$"*

- Special atom emp

  "The heap is empty (no allocated location)"

- A special connective $*$, called separating conjunction $\phi_1 * \phi_2$

  "The heap can be split into two disjoint parts, satisfying ϕ_1 and ϕ_2, respectively"

- Inductively defined predicates (fixpoint semantics), used to describe finite structures of unbounded size, e.g., list segments :

$$ls(x, y) \Leftarrow \mathrm{emp} \wedge x \approx y \qquad ls(x, y) \Leftarrow \exists z (x \mapsto z * ls(z, y))$$

- Equational atoms, usual connectives

Automation of Reasoning in SL with Inductive Definitions : Existing Results

- Focus on symbolic heaps

$$\exists x_1, \ldots, x_n \, [(A_1 * \cdots * A_n) \wedge \phi]$$

  where the A_i's are atoms, ϕ is a conjunction of equational literals
- Satisfiability is $\textsc{Exptime}$-complete [Brotherston et al. LICS 14]
- Entailment is undecidable
- Entailment is 2-$\textsc{Exptime}$-complete if the inductive definitions satisfy some conditions [Iosif et al., CADE 13, Katelaan et al. TACAS 19, Echenim et al. LPAR 20, CSL 21, CADE 22]

Automation of Reasoning in SL with Inductive Definitions : Existing Results

- Focus on symbolic heaps

$$\text{or simply : } \quad \exists x_1, \dots, x_n \, [A_1 * \dots * A_n]$$

  where the A_i's are atoms or equational literals, assuming equational literals are satisfied only in empty heaps
- Satisfiability is EXPTIME-complete [Brotherston et al. LICS 14]
- Entailment is undecidable
- Entailment is 2-EXPTIME-complete if the inductive definitions satisfy some conditions [Iosif et al., CADE 13, Katelaan et al. TACAS 19, Echenim et al. LPAR 20, CSL 21, CADE 22]

What About Reasoning With More Complex Heap Models ?

Heaps with Permissions :

- Some locations can be "shared" between threads, if the permissions are compatible [Bornat, POPL 2005]
- Points-to atoms of the form : $x \stackrel{z}{\mapsto} (y_1, \ldots, y_n)$

  "Location x is allocated with permission z and refers to $y_1, \ldots, y_n$"
- Non disjoint heaps may be combined if :
  - They agree on the shared locations
  - The permissions are compatible
- Inductive predicates have parameters denoting permissions

Permission Model

- A set of *permissions*, e.g., : `write`, `read`
- A (partial) combination operator $\oplus$ stating which permissions can be combined and what is the resulting permission, e.g. :

$$\begin{array}{cc} \texttt{write} \oplus x & \text{undefined} \\ \texttt{read} \oplus \texttt{read} & \texttt{read} \end{array}$$

- Another example of permission model
  - Rational numbers in $]0, \ldots, 1]$
  - $x \oplus y = x + y$ if $x + y \leq 1$, undefined otherwise
- (optional) Additional predicates on permissions, maximal permission
- Permission terms may be undefined : $def(p)$ true if p is defined

Example

- Basic Separation Logic
  - $x \mapsto (y) * x \mapsto (z)$ is unsatisfiable
  - x cannot be allocated in disjoint parts of the heap
- Separation Logic with Permissions
  - $x \overset{p}{\mapsto} (y) * x \overset{q}{\mapsto} (z)$ is satisfiable
  - entails that $y = z$ and that p and q are compatible (e.g. $p = q = \mathtt{read}$

Existing Results on Automated Reasoning in Separation Logic with Permissions

[Demri et al. FSTTCS 2017] :

- Focus on list segments with (a unique) permission
- Assuming we have an oracle for the permission theory :
  - Satisfiability is in $\textsc{Np}$
  - Entailment is co-$\textsc{Np}$
- What can be said about generic inductive definitions ?

Our Contribution

- On the negative side : satisfiability is undecidable in general
- On the positive side : EXPTIME-complete for a syntactic fragment : $\exists$-restricted $\mathfrak{h}$-regular inductive definitions
- The fragment is sufficiently expressive to denote many usual data structures such as lists or trees, but not, e.g., doubly linked lists

Two Separating Conjunctions

- $*$ Weak separating conjunction :
  - Based on the combination of permissions
  - Used in input formulas
- $\circ$ Strong separating conjunction : disjoint union of heaps
  - Usual separating conjunction in SL without permissions
  - Useful in the paper to define the satisfiability testing algorithm
  - Also useful to define inductive predicates
  - $x \overset{p}{\mapsto} (y) \circ x \overset{q}{\mapsto} (z)$ is unsatisfiable

Weak Separating Conjunction

Heaps are partial functions mapping locations to pairs $(\mathbf{l}, p)$ where $\mathbf{l}$ is a tuple of locations and p is a permission

Definition

If $\mathfrak{h}_1, \mathfrak{h}_2$ are heaps, then $\mathfrak{h}_1 \sqcup \mathfrak{h}_2$ is defined iff for every $\ell \in \mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2)$, if $\mathfrak{h}_i(\ell) = (\ell_1^i, \ldots, \ell_{k_i}^i, \pi_i)$ (for $i = 1, 2$) then :

- $k_1 = k_2$, $\ell_j^1 = \ell_j^2$ holds for all $j \in \{1, \ldots, k_1\}$
- and $\pi_1 \oplus \pi_2$ is defined

Then $\mathfrak{h}_1 \sqcup \mathfrak{h}_2$ is defined as follows :

- If $\ell \in \mathrm{dom}(\mathfrak{h}_i) \setminus \mathrm{dom}(\mathfrak{h}_j)$ with $(i, j) \in \{(1, 2), (2, 1)\}$ then $(\mathfrak{h}_1 \sqcup \mathfrak{h}_2)(\ell) \stackrel{\text{def}}{=} \mathfrak{h}_i(\ell)$
- If $\ell \in \mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2)$ then $(\mathfrak{h}_1 \sqcup \mathfrak{h}_2)(\ell) \stackrel{\text{def}}{=} (\ell_1^1, \ldots, \ell_{k_1}^1, \pi_1 \oplus \pi_2)$

Why Do We Need Strong Separating Conjunction ?

Can we define list segment with weak separating conjunction ?

$$
\begin{array}{rcl}
ls(x,y,z) & \Leftarrow & x \approx y \\
ls(x,y,z) & \Leftarrow & \exists u\,(x \overset{z}{\mapsto} (u) * ls(u,y,z))
\end{array}
$$

Two issues :

- Does not fit in with the usual definition of lists [Demri et al., 2017] :
  - $ls(x,x,z)$ true on the heap : $\{x \mapsto (x,z)\}$ (good)
  - but also on any heap of the form

$$\{x \mapsto (x, z \oplus \ldots \oplus z)\}$$

  (if $z \oplus \ldots \oplus z$) defined)

- Using weak conjunction inside inductive definitions makes the satisfiability problem undecidable, even for some very simple structures

Why Do We Need Strong Separating Conjunction ? (2)

ls should be defined as follows (using strong conjunction) :

$$\begin{array}{lcl} ls(x,y,z) & \Leftarrow & x \approx y \\ ls(x,y,z) & \Leftarrow & \exists u\,(x \overset{z}{\mapsto} (u) \circ ls(u,y,z)) \end{array}$$

Weak separating conjunction is useful only in input formulas

A Restriction on Inductive Rules

Definition

A rule is $\mathfrak{h}$-*regular* if it is of the following form : $P(x, \mathbf{y}, \mathbf{z}) \Leftarrow \exists u_1, \ldots, u_n (x \overset{p}{\mapsto} (v_1, \ldots, v_k) \circ Q_1(u_1, \mathbf{y}_1, \mathbf{z}_1) \ldots \circ Q_n(u_n, \mathbf{y}_n, \mathbf{z}_n) \circ \phi)$ where

- $x, \mathbf{y}$ are location variables, $\mathbf{z}_i$, $\mathbf{z}$ are tuples of permission variables, p is a permission term
- $\{u_1, \ldots, u_n\} \subseteq \{v_1, \ldots, v_k\}$ and ϕ is purely equational
- All the variables in $\mathbf{z}_i, z$ occur in $\mathbf{z}$

- A strictly more restrictive version of the PCE conditions of [Iosif et al., CADE 2013]
  - Each existential variable must be allocated at *the next recursive call*
- Encode regular languages + additional pointers to previously allocated nodes (or free variables)
- No compound permission term in recursive predicate calls

Necessary But Not Sufficient

- As we shall see, these restrictions are insufficient for the decidability of the satisfiability problem
- Additional restrictions are needed on the use of existential variables

We first describe the last step of the algorithm :

- Consider a formula of the form $\phi_1 \circ \ldots \circ \phi_n$, where ϕ_i are atoms
- Close to separation logic with no permission
- It suffices to construct abstractions of models $(\sim, A, \rho)$, where : $\sim$ is an equivalence relation on free variables denoting locations (equality relation), A denotes a the set of allocated free variables, and ρ is a permission formula
- Easy to construct by induction on the formulas (with fixpoint computation)

Constructing Abstractions

$\mathcal{A}(\phi)$ is a set of heap abstraction defined as follows : as follows (for all equivalence relations $\sim$) :

- If $\phi = x \overset{p}{\mapsto} (y_1, \dots, y_n)$ then $(\sim, \{y \mid y \sim x\}, \mathit{def}(p)) \in \mathcal{A}(\phi)$
- If $\phi = x \approx y$ with $x \sim y$ then $(\sim, \emptyset, \top) \in \mathcal{A}(\phi)$.
- If $\phi = x \not\approx y$ with $x \not\sim y$ then $(\sim, \emptyset, \top) \in \mathcal{A}(\phi)$.
- If ϕ is a permission formula then $(\sim, \emptyset, \phi) \in \mathcal{A}(\phi)$.
- If $\phi = \exists x.\psi$, $(\sim, A, \rho) \in \mathcal{A}(\psi)$ then $(\{u, v \mid u \sim v, u, v \neq x\}, A \setminus \{x\}, \rho) \in \mathcal{A}(\phi)$.
- If $\phi = \phi_1 \circ \phi_2$, $(\sim, A_i, \rho_i) \in \mathcal{A}(\phi_i)$ (for all $i = 1, 2$) with $A_1 \cap A_2 = \emptyset$, then $(\sim, A_1 \cup A_2, \rho_1 \circ \rho_2) \in \mathcal{A}(\phi)$.
- If $\phi = P(\mathbf{x}, p)$ and $\phi \Leftarrow_{\mathfrak{R}} \xi$ then $\mathcal{A}(\xi) \subseteq \mathcal{A}(\phi)$.

Lemma

$\mathcal{A}(\phi)$ is finite (trivial as only a bounded number of free variables need to be considered, but key point : no existential variable of type permission)

Lemma

ϕ is satisfiable iff $\mathcal{A}(\phi)$ contains a tuple $(\sim, A, \rho)$ such that ρ is satisfiable

What About Weak Separating Conjunctions ?

- Goal : transform formulas of the form $\phi_1 * \cdots * \phi_n$ into $\psi_1 \circ \ldots \circ \psi_m$
- Three steps :
  1. Decompose every spatial atom ϕ_i into a $\circ$-conjunction $\psi_1^i \circ \ldots \circ \psi_{m_i}^i$ such that every ψ_j^i allocates *exactly* one free variable x_j
  2. Using the latter property, we may push $*$ below $\circ$
     We get a $\circ$-conjunction of $*$-conjunctions of atoms each allocating the same variable x_j
  3. Merge each $*$-conjunctions of atoms into a single atom (with new rules)

Context Predicates $\multimap$

- To decompose atoms, we need to synthesize new predicate symbols and rules
- $q(y) \multimap r(x)$: true in a structure that satisfies $r(x)$ after a disjoint structure satisfying $q(y)$ is added
- Similar to the separating implication $-\!\!*$, but not exactly equivalent, because the definition is not purely semantic : it depends on the unfolding tree
- No need to extend the logic : $q(y) \multimap r(x)$ may be denoted as an atom, with rules automatically generated from those of $r(x)$

Context Predicates : Example

Lists :

$$p(x,y) \Leftarrow x \overset{y}{\mapsto} () \quad p(x,y) \Leftarrow \exists z\,(x \overset{y}{\mapsto} (z) \circ p(z,y))$$

- $p(y,z) \multimap p(x,z)$ denotes a structure obtained from a list satisfying $p(x,z)$ by deleting the part corresponding to the call $p(y,z)$
- $p(y,z) \multimap p(x,z)$ denotes a list segment from x to y : $p(y,z) \multimap p(x,z) \equiv ls(x,y,z)$
- $p(y,z) \multimap p(x,z)$ is defined by the following rules :

$$\begin{array}{lcc} p(y,z) \multimap p(x,z) & \Leftarrow & x \approx y \\ p(y,z) \multimap p(x,z) & \Leftarrow & \exists u\,(x \overset{z}{\mapsto} (u) \circ p(y,z) \multimap p(x,z)) \end{array}$$

These rules can be computed automatically

Using Context Predicates to Decompose Atoms

For every atom $p(y, \mathbf{z})$ and free variable x, replace $p(y, \mathbf{z})$ by :

- Either $p'(y, x, \mathbf{z})$ where the rules of p' are obtained from those of p by adding the constraint $u \not\approx x$ for each points-to atom $u \mapsto (\dots)$

  Cover the case where x is not allocated
- Or $\exists \mathbf{u} \, [q(x, \mathbf{u}) \circ (q(x, \mathbf{u}) \multimap p(y, \mathbf{z}))]$ (for some predicate q)
  - Cover the case where x is allocated
  - If x is allocated then (due to the restriction on the rules) there must be a call to some atom of the form $q(x, \mathbf{u})$)
  - $q(x, \mathbf{u}) \multimap p(y, \mathbf{z})$ cannot allocate x (more generally each $\circ$-conjunction contains at most on atom allocating x)

Decomposition of Atoms : Termination Issue

- The previous transformation ensures that every predicate atom allocates exactly one free variable. . .
- . . . but it does not terminate in general :
  - The transformation must be applied on each variable
  - New variables $\mathbf{u}$ are introduced during the process

Add additional restrictions on the rules :

- Easy solution : forbid existential parameters (except at first position in the atom) : for all $p(x, y_1, \ldots, y_n)$, $y_1, \ldots, y_n$ must be free
  Rather restrictive
- More general condition : assume that for all y_i that is not free, there is an atom $q(y_i, z_1, \ldots, z_n)$ where $z_1, \ldots, z_n$ are free
- Require to compute, for each predicate p, the set of arguments $\gamma(p)$ of p that may be instantiated by existential variables

For instance :

$$\begin{array}{lcll} p(x,y,z) & \Leftarrow & \exists u,v\, x \stackrel{z}{\mapsto} (u,v,y) * p(u,v,z) * q(v,z) & ok \\ q(v,z) & \Leftarrow & v \stackrel{z}{\mapsto} () & \\ p(x,y,z) & \Leftarrow & \exists u,v\, x \stackrel{z}{\mapsto} (u,v,y) * p(u,v,z) * p(v,u,z) & ko \end{array}$$

Decomposition of Atoms : Conditions for Termination (3)

- Rules satisfying the above condition are called $\exists$-restricted
- The condition ensures termination of the previous decomposition process
- Intuitively :
  - New variables are introduced only when applying the decomposition on a variable originally occurring in the formula
  - No new variables are introduced when applying the decomposition on a variable introduced during the decomposition process

Representable Structures

- Regular tree languages + additional pointers (as in PCE)
- A set of distinguished nodes (free variables)
- The additional pointers may refer :
  - either to distinguished nodes
  - or to a structure with no additional pointers other than distinguished nodes

Second Step : Pushing $*$ below $\circ$

Lemma

If ψ_i and ψ'_i allocates exactly *the free variables V_i, and $V_1 \cap V_2 = \emptyset$, then : $(\psi_1 \circ \psi_2) * (\psi'_1 \circ \psi'_2)$ is satisfiable iff $(\psi_1 * \psi'_1) \circ (\psi_2 * \psi'_2)$ is satisfiable*

Proof

Idea :

- $(\psi_1 * \psi'_1) \circ (\psi_2 * \psi'_2) \models (\psi_1 \circ \psi_2) * (\psi'_1 \circ \psi'_2)$ holds if every case
- For the converse :
    - Rename all locations not associated to free variables in the model of $\psi_2 * \psi'_2$ so that they do not occur in the model of $\psi_1 * \psi'_1$
    - The renaming does not affect the model of $\psi_1 * \psi'_1$
    - As ψ_1 and ψ'_2 allocate distinct free variables, this ensures that the heaps corresponding to ψ_1 and ψ'_2 are disjoint (and similarly for ψ_2 and ψ'_1)

Pushing $*$ below $\circ$ (2)

By applying repeatedly the previous result we may transform $*_{i=1}^{n}(\psi_1^i \circ \ldots \circ \psi_{m_i}^i)$ into :

$$\circ_{j=1}^{m}(\psi_j^1 * \cdots * \psi_j^i)$$

where every formula ψ_j^i allocates *exactly* the same variable x_j (or is emp)

Merging $*$-Conjunctions

- Merge $p(x, \mathbf{y}, \mathbf{z_1}) * p'(x, \mathbf{y}', \mathbf{z_2})$ into $pp'(x, \mathbf{y}, \mathbf{y}', \mathbf{z_1}, \mathbf{z_2})$
- The rules of pp' are computed by "merging" rules of p and p', as for tree automata

Merging $*$-Conjunctions (2)

Examples :

- $p(x, y, z) * p'(x, z')$ with

$$\begin{array}{rcl} p(x, y, z) & \Leftarrow & x \overset{z}{\mapsto} (u, y) \circ q(u, z) \\ p'(x, y', z') & \Leftarrow & x \overset{z'}{\mapsto} (u, y') \circ r(u, y', z') \end{array}$$

- We get : $pp'(x, y, y', z, z')$ with

$$pp'(x, y, y', z, z') \Leftarrow x \overset{z \oplus z'}{\mapsto} (u, y) \circ qr(u, y', z) \circ y \approx y'$$

- The fact that every predicate atom allocates exactly one free variable ensures that the rules can always be combined

Summary

- The steps described above yield an algorithm for testing the satisfiability of $*$ conjunctions of atoms defined over regular, $\exists$-restricted rules
- The algorithm has exponential complexity (modulo satisfiability testing for permission formulas)
- The problem is $\textsc{Exptime}$-hard (by reduction from the halting problem for alternating Turing machines running in polynomial space), even with no permissions

Undecidability Results

The problem is **undecidable** in both the following cases :

1. If the rules are $\mathfrak{h}$-regular, but not $\exists$-restricted and there are (not necessarily distinct) permissions π_1, π_2 such that $\pi_1 \oplus \pi_2$ is defined
2. Or, if $*$ is used instead of $\circ$ in the definition of the rules and for all $n \geq 0$ there is a permission π such that $\underbrace{\pi \oplus \ldots \oplus \pi}_{n \text{ time}}$ is defined

Undecidability Results (1)

If the rules are not $\exists$-restricted, one may encode the PCP as follows

- Let $u_1, \ldots, u_N, v_1, \ldots, v_N$ be words
- Construct a heap $\{y_i \mapsto (y_{i+1}, c_i, \ell_i, \ell'_i, \pi) \mid i = 1, \ldots, k\}$ encoding a potential witness $w = w_1. \ldots .w_k$ with $w = u_{i_1}. \ldots .u_{i_n} = v_{j_1}. \ldots .v_{j_m}$ (with $n, m > 0$)
- Add links ℓ_i, ℓ'_i to elements of two lists $\lambda_{i_1}, \ldots, \lambda_{i_n}$ and $\lambda'_{j_1}, \ldots, \lambda'_{j_m}$ denoting the sequences $i_1 \ldots, i_n$ and $j_1 \ldots, j_m$
- The lists must be constructed in reverse order to ensure $\mathfrak{h}$-regularity
- Add a predicate checking that the two lists denote identical sequences (constructing a list of tuples $(\lambda_{i_k}, \lambda'_{j_k})$, again in reverse order)

Undecidability Results (2)

If $*$ is used instead of $\circ$ then one may encode the PCP as follows :

- Construct a *circular* list representing the potential witness
- Since the list is circular one may go through it an arbitrary number of times (assuming that for all n, there exists a permission π such that $n.\pi$ is defined)
- Use two parameters x, y denoting the positions of the start of words u_{i_j} and v_{i_j} (initially $x = y = 1$, then $x = |u_{i_1}| + 1$ and $y = |v_{i_1}| + 1$ etc.)
- At each step : check that the words starting at position x and y are identical and compute the start of the next words x', y'
- Then repeat the process with x', y'
- End with a special word $\#$
- To fulfill the $\exists$-restrictedness condition, x, x' cannot refer to elements of the list
- Add instead dummy "marks" associated with every element in the list and refer to these "marks"

Future Work

- Relax the $\exists$-restrictedness conditions ?
  - Identify formal parameters that may only be instantiated by a bounded number of variables during unfolding
  - Should be possible, but does not really extend expressive power (encodings are possible)
- Extend to non $\mathfrak{h}$-regular case ? (use full PCE conditions of [Iosif et al. CADE 2013] instead)
  - $\exists$-restrictedness seems more important for decidability than $\mathfrak{h}$-regularity
- Entailment Problem ?
  - Use the same ideas : decomposition + commutation of $*$ and $\circ$ + merge ?
  - Could be sufficient for quantifier-free entailments ?
  - Entailments with existential variables may be more difficult