Reasoning on Dynamic Transformations of Symbolic Heaps

N. Peltier

Univ. Grenoble Alpes, CNRS, LIG, CAPP - ANR Project Narco

TIME 22 — 29th International Symposium on Temporal Representation and Reasoning
November 2022

Overview

- Starting point :
  - A fragment of separation logic (SL) with inductive definitions (*symbolic heaps*)
  - Specify pointer-based recursive data structures
  - The entailment problem is decidable if the inductive definitions satisfy some conditions (the *PCE* conditions)
- How to handle dynamic transformations of the data structures ?
- Entailment problems of the form :

$$\phi \models_{\mathcal{R}}^{\mathcal{S}} \Psi$$

  where ϕ is an SL formula and Ψ an LTL formulas built on SL formulas, interpreted modulo some inductive rules $\mathcal{R}$ and some finite transition system $\mathcal{S}$
- The problem is undecidable in general, decidable under some conditions

Separation Logic - Syntax

- Variables (no function symbols) interpreted as locations (memory addresses)
- Equational atoms : $x \approx y$ or $x \not\approx y$
- Spatial atoms (describes the shape of the heap) :
  - emp ("empty")
  - $x \mapsto (y_1, \ldots, y_k)$ ("x is the only allocated location and refers to $y_1, \ldots, y_k$")
  - $p(x_1, \ldots, x_n)$, where p is an inductively defined spatial predicate. Describes some part of the heap of unbounded size, e.g., a list segment $ls(x, y)$
- Usual connective : $\vee$ (no negation, no conjunction)
- Special connective : $*$ (separating conjunction)
- Quantifier $\exists$ (no $\forall$)

Separation Logic - Interpretations

Let Loc be an infinite (countable) set of *locations* (e.g., addresses).
Formulas are interpreted over structures $(\mathfrak{s}, \mathfrak{h})$ where :

- $\mathfrak{s}$ is a function (*store*) mapping every variable to an element of Loc
- $\mathfrak{h}$ is a partial finite function (*heap*) from Loc to Loc*
- A location is *allocated* if it occurs in $\mathrm{dom}(\mathfrak{h})$

Let ϕ be a formula without spatial predicate symbols. $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff one of the following conditions hold :

- ϕ is $x \approx y$, $\mathfrak{s}(x) = \mathfrak{s}(y)$ and $\mathfrak{h} = \emptyset$
- ϕ is $x \not\approx y$, $\mathfrak{s}(x) \neq \mathfrak{s}(y)$ and $\mathfrak{h} = \emptyset$
- ϕ is emp and $\mathfrak{h} = \emptyset$
- ϕ is $x \mapsto (y_1, \ldots, y_k)$, $\mathfrak{h}(\mathfrak{s}(x)) = (\mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k))$ and $\mathrm{dom}(\mathfrak{h}) = \{\mathfrak{s}(x)\}$
- $\phi = \phi_1 \vee \phi_2$ and there exists $i = 1, 2$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi_i$
- $\phi = \exists x\ \psi$ and there exists $\ell \in \mathrm{Loc}$ such that $(\mathfrak{s}[x \leftarrow \ell], \mathfrak{h}) \models \psi$
- $\phi = \phi_1 * \phi_2$ and there exist disjoint heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \cup \mathfrak{h}_2$ and for every $i = 1, 2$, $(\mathfrak{s}, \mathfrak{h}_i) \models \phi_i$

Separation Logic - Evaluation Of Inductively Defined Predicates

- Every spatial predicate p is associated with a set of rules $p(x_1, \ldots, x_n) \Leftarrow \phi$ (provided by the user)
- We write $\psi \to \psi'$ if ψ' is obtained from ψ by replacing an occurrence of an atom $p(y_1, \ldots, y_n)$ by $\phi[x_i \leftarrow y_i \mid i = 1, \ldots, n]$
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} p(x_1, \ldots, x_n)$ iff there exists a formula ψ not containing any predicate symbol, such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$ and $p(x_1, \ldots, x_n) \to^* \psi$

Example

Non empty list segments :

$$\begin{array}{lcll} ls(x,y) & \Leftarrow & x \mapsto (y) & \text{base case} \\ ls(x,y) & \Leftarrow & \exists z\ (x \mapsto (z) * ls(z,y)) & \text{inductive case} \end{array}$$

With this definition :

$$x \mapsto (y) * y \mapsto (z) \models_{\mathcal{R}} ls(x,z)$$

$$ls(x,y) * ls(y,z) \models_{\mathcal{R}} ls(x,z)$$

$$ls(x,y) * ls(y,x) \models_{\mathcal{R}} \exists u\, ls(u,u)$$

$$ls(x,y) * ls(x,y') \text{ is unsatisfiable}$$

$$x \mapsto (y) * y \mapsto (z) * x \approx y \text{ is unsatisfiable}$$

$$x \mapsto (y) * y \mapsto (z) \not\models_{\mathcal{R}} x \not\approx y$$

Existing Results

- Satisfiability is decidable (Brotherston et al., LICS 14)
- Entailment is undecidable in general : an easy reduction from the inclusion problem for context-free grammars
- Decidable for a specific class of inductive definitions (Iosif, Rogalewicz, Simácek, CADE 2013)
- A 2-EXPTIME algorithm (Katelaan and Zuleger, LPAR 20)
- The 2-EXPTIME bound is tight (Echenim, Iosif and Peltier, LPAR 2020)
- A 2-EXPTIME algorithm handling existential variables (Echenim, Iosif, Peltier CSL 2020)
- Other complexity results for specific fragments

A Class Of Inductive Definitions For Which Entailment Is Decidable

3 conditions :

1. Progress (P) : Every rule allocates exactly one memory location, i.e., is of the form $p(x_1, \dots, x_n) \Leftarrow \exists z_1, \dots, z_m \,.\, x_1 \mapsto (y_1, \dots, y_k) * \phi$, where ϕ contains no $\mapsto$
   The variable x_1 is called the **root** of $p(x_1, \dots, x_n)$
2. Connectivity (C) : If an atom $q(x'_1, \dots, x'_l)$ occurs in ϕ, then necessarily $x'_1 = y_i$, for some $i = 1, \dots, k$
3. Establishment (E) : For every $i = 1, \dots, m$, z_i is allocated in all models of ϕ

PCE problems : **P**rogress, **C**onnectivity and **E**stablishment

Heap Constraints

Definition

A *heap constraint* is a triple (S^+, S^-, X), where S^+ and S^- are sets of symbolic heaps, $S^+ \neq \emptyset$ and X is a finite set of variables

Definition

A heap constraint is *satisfiable* iff there exists a structure $(\mathfrak{s}, \mathfrak{h})$ satisfying all formulas in S^+, satisfying no formula in S^- and allocating no variables in X

Theorem

The satisfiability problem is decidable for heap constraints (with PCE rules)

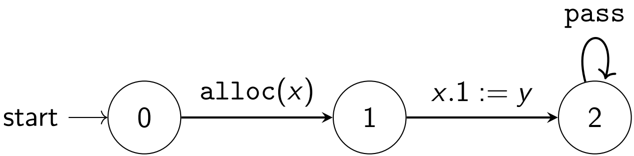
Proof : an easy extension of existing results

Dynamic Transformation Of Heaps : Actions

- Terms : x or $x.i$, where x is a variable, $i \in \mathbb{N}$ (non nested)
- Basic actions :
  - affectations : $x := s$, where x is a variable and s is a term
  - redirections : $x.i := s$, where s is a term
  - allocations : $\texttt{alloc}(x)$ (x refers to $(x, \ldots, x)$)
  - desallocations : $\texttt{free}(x)$
  - null actions : $\texttt{pass}$
  - tests : $\texttt{test}(\gamma)$, where γ is a condition, i.e., a boolean combination of equations $t \approx s$ between terms
- $(\mathfrak{s}, \mathfrak{h})[a]$: structure $(\mathfrak{s}', \mathfrak{h}')$ obtained by applying a on $(\mathfrak{s}, \mathfrak{h})$
- $(\mathfrak{s}, \mathfrak{h})[a]$ is a *partial function* (a may "fail")

- Transition systems are finite state automata, where edges are labeled by actions
- A run from an initial structure $(\mathfrak{s}, \mathfrak{h})$ is an infinite path $a_1, \ldots, a_n, \ldots$ in the automaton such that there exists a sequence $(\mathfrak{s}_i, \mathfrak{h}_i)$ with :
  - $(\mathfrak{s}_0, \mathfrak{h}_0) = (\mathfrak{s}, \mathfrak{h})$
  - For all $i \geq 0$, $(\mathfrak{s}_{i+1}, \mathfrak{h}_{i+1}) = (\mathfrak{s}_i, \mathfrak{h}_i)[a_i]$ (must be defined)
- Only infinite runs are considered

Transition Systems - Example



LTL Formulas

Syntax :

- LTL atoms : SL formulas , atomic conditions, actions and states
- Usual LTL connectives : $\neg\Phi$, $\Phi \vee \Psi$, $\boldsymbol{X}\,\Phi$, $\Phi\,\boldsymbol{U}\,\Psi$ etc.

Semantics :

- Given $\mathcal{R}$ and $\mathcal{S}$, an LTL formula is interpreted w.r.t. some initial (time 0) structure $(\mathfrak{s}, \mathfrak{h})$ and run $(\mathfrak{s}_i, \mathfrak{h}_i)$ $(i \in \mathbb{N})$ (corresponding to a given path in the transition system)
- SL atoms and conditions are interpreted on $(\mathfrak{s}_i, \mathfrak{h}_i)$ at time i
- Actions and states refer to the considered run : state and transition applied at time i
- LTL connectives are handled as usual

Entailment Problem

Entailment problem :

$$\phi \models^{\mathcal{S}}_{\mathcal{R}} \Psi$$

where

- ϕ is an SL formula, Ψ is an LTL formula
- $\mathcal{R}$ is a set of inductive definitions (PCE), $\mathcal{S}$ is a transition system

e.g., $ls(y,z) \models^{\mathcal{S}}_{\mathcal{R}} \boldsymbol{F}\, ls(x,z)$ or $ls(y,z) \models^{\mathcal{S}}_{\mathcal{R}} \boldsymbol{G}(2 \Rightarrow ls(x,z))$

Theorem

The entailment problem is undecidable

Proof : $\mathcal{S}$ encodes a Turing machine, ϕ allocates a tape of unbounded size, Ψ states that the machine does *not* terminate

A Restriction On Transition Systems

Definition

A system is *oriented* if affectations do not occur inside a cycle (i.e., no action $x := s$ where x is a variable occurs inside a path from some state q to q)

Our goal :

- Define an algorithm to test entailments, that will terminate on oriented systems
- Idea : reduce the entailment problem to an LTL satisfiability problem
- Expresses transitions and SL properties as LTL formulas

LTL Encoding

- Encoding of states and transitions is trivial
- Encoding of “static” SL properties
  - Dismiss unsatisfiable sets of SL literals (SL formulas or negations of SL formulas)
    e.g., the valid SL entailment

$$ls(x,y) * ls(y,z) \models_{\mathcal{R}} ls(x,z)$$

    should yield the LTL axiom :

$$\neg(ls(x,y) * ls(y,z)) \vee ls(x,z)$$

- Encode the semantics of actions, i.e. :
  - state preconditions of actions
    e.g. $x.1 := y$ possible only if x is allocated
  - relate $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h})[a]$
    $\rightarrow$ use a weakest precondition calculus

Weakest Precondition Calculus

- Weakest precondition : given an SL formula ϕ and an action a, $wpc(\phi, a)$ asserts conditions ensuring that ϕ is satisfied after the action is performed
- Can $wpc(\phi, a)$ be computed and expressed in SL ?
- In some cases, yes, for instance :
  - $wpc(\phi, \mathtt{free}(x)) \stackrel{\text{def}}{=} \exists y_1 \dots \exists y_k.(\phi * x \mapsto (y_1, \dots, y_k))$.
  - $wpc(\phi, x := y) \stackrel{\text{def}}{=} \phi\{x \leftarrow y\}$ (if x, y are variables)
- For actions depending on $x.i$, this is feasible only if x is explicitly allocated in the formula ϕ, i.e., if ϕ contains an atom $x \mapsto (x_1, \dots, x_k)$
- For instance : $wpc(\exists \mathbf{x}.(\phi * x \mapsto (x_1, \dots, x_k)), x.i := y)$ is $\exists \mathbf{x} \exists x'.(\phi * x \mapsto (x_1, \dots, x_{i-1}, x', x_{i+1}, \dots, x_k) * x_i \approx y)$ but $wpc(ls(x, z), x.i := y)$ cannot be defined

How To Enforce The "Explicit" Allocation Of Variables ?

Given an SL formula ϕ and a variable x, can we compute an SL formula ψ such that :

- ψ and ϕ are equivalent in all structures $(\mathfrak{s}, \mathfrak{h})$ in which $\mathfrak{s}(x)$ is allocated
- ψ contains an atom of the form $x \mapsto (x_1, \dots, x_k)$

Example :

- $\phi = ls(y, z)$
- Solution :

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

Can ψ be computed automatically in all cases ?

Context Predicates

Answer : yes (for PCE rules), but this requires to create new predicates and rules

- For every pair of predicates p, q with arities n and m, define a predicate $(q \multimap p)$ of arity $n + m$
- $(q \multimap p)(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is satisfied by all (non empty) structures that will satisfy $p(x_1, \ldots, x_n)$ after a disjoint heap satisfying $q(y_1, \ldots, y_m)$ is added to the current heap
- The rules of $(q \multimap p)$ are defined exactly as those of p, except that exactly one call to $q(y_1, \ldots, y_m)$ is removed

More formally, for each rule

$$p(u_1, \ldots, u_n) \Leftarrow \exists \mathbf{w}.(u_1 \mapsto (\mathbf{y}) * p'(\mathbf{z}) * \psi)$$

we add :

$$(q \multimap p)(u_1, \ldots, u_n, v_1, \ldots, v_m) \Leftarrow \\ \exists \mathbf{w}.(u_1 \mapsto (\mathbf{y}) * (q \multimap p')(\mathbf{z}, v_1, \ldots, v_m) * \psi)$$

$$(q \multimap p)(u_1, \ldots, u_n, v_1, \ldots, v_m) \Leftarrow \\ \exists \mathbf{w}.(u_1 \mapsto (\mathbf{y}) * \mathbf{z} \approx (v_1, \ldots, v_m) * \psi) \quad \text{if } q = p'$$

Context Predicates

Given a formula ϕ and a variable x, ψ is the disjunction of formulas obtained as follows :

- Choose an atom $p(y, \mathbf{z})$ in ϕ, and either :
  - add the condition $x \approx y$ and replace $p(y, \mathbf{z})$ by $p(x, \mathbf{z})$
  - or replace $p(y, \mathbf{z})$ by $\exists \mathbf{u} \, ((q \multimap p)(y, \mathbf{z}, x, \mathbf{u}) * q(x, \mathbf{u}))$
- In both cases, we get an atom with first argument x
- By the progress condition, it suffices to unfold this atom once to get an atom of the form $x \mapsto (\dots)$

LTL Encoding (Continued)

- Using context predicates, weakest preconditions can be automatically computed in all cases
- Allow one to encode all the properties of the transition systems in LTL (see paper for the definition of the set of axioms)
- If $\mathcal{S}$ is oriented then the obtained set of axioms is *finite*
- Intuition : the set of "visible" locations is finite, hence the set of symbolic heaps that need to be considered is finite
- The entailment problem $\phi \models^{\mathcal{S}}_{\mathcal{R}} \Psi$ can be reduced to an LTL satisfiability test (if $\mathcal{R}$ is PCE and $\mathcal{S}$ is oriented)
- Generating all axioms at once is not practical : use a incremental model-refinement algorithm instead

Entailment Checking Algorithm

$\mathcal{A} \leftarrow \{\phi, q_I, \neg\Psi\}$
while $\mathcal{A}$ admits an LTL interpretation $\mathcal{I}$ **do**
  $S^+ \leftarrow \{\phi \mid \mathcal{I}(\phi, 0) = \mathit{true},\ \phi$ is a symbolic heap$\}$
  $S^- \leftarrow \{\phi \mid \mathcal{I}(\phi, 0) = \mathit{false},\ \phi$ is a symbolic heap$\}$
  $X \leftarrow \{x \in \mathcal{V}^\star \mid \mathcal{I}(\phi, 0) \not\models \mathtt{alloc}(x)$ (i.e. $\mathcal{I}(\phi, 0) \not\models x.1 \approx x.1)$ $\}$
  if Heap constraint (S^+, S^-, X) is unsatisfiable **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\chi\}$, where χ is an LTL-encoding of $\neg(S^+, S^-, X)$
  else
    Let $(\mathfrak{s}, \mathfrak{h})$ be an $\mathcal{R}$-model of (S^+, S^-, X)
    if $\mathcal{I}$ corresponds to a run r in $\mathcal{S}$ from $(\mathfrak{s}, \mathfrak{h})$ **then**
      Return $(\mathfrak{s}, \mathfrak{h})$
    else
      Let ψ be an axiom s.t. $(\mathfrak{s}, \mathfrak{h}) \not\models_{\mathcal{R}}^{\mathcal{S}} \psi$
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{\psi\}$
    end if
  end if
end while
Return $\top$

Properties Of The Entailment Checking Algorithm

- If the algorithm returns $(\mathfrak{s}, \mathfrak{h})$ then $(\mathfrak{s}, \mathfrak{h})$ is a counter-example of the considered entailment problem
- If the algorithm returns $\top$ then the considered entailment problem is valid
- The algorithm always terminates if $\mathcal{S}$ is oriented

LTL Encoding (Continued)

Why do we need both pre- and post-conditions ?

- Weakest preconditions allow one to move all constraints backward in the path, so that we get constraints on the initial structure (at $t = 0$)
- Strongest postconditions ensure that at every time at least one symbolic heap is satisfied
  - $\rightarrow$ allows one to encode all elementary conditions into the considered fragment of SL

Future Work

- Is the algorithm complete (for counter-examples) on non oriented problems ?
- Complexity ? (2- or 3-EXPTIME ?)
- How to handle non deterministic actions ? (e.g., allocate a new, arbitrary chosen, location)
- Implementation