Extending propositional separation logic for robustness properties

Alessio Mansutti
LSV, CNRS, ENS Paris-Saclay

Ahmedabad - December 2018
Separation logic and program verification

- **Separation Logic** (Reynolds’02) is used in Hoare proof systems for program verification of languages with pointers.

- **Hoare calculus**: axioms and rules reason about triples:

\[
\{\varphi\} \text{P} \{\varphi'\}
\]

Any (memory) model that satisfies \(\varphi\) will satisfy \(\varphi'\) after being modified by the program \(P\).
Separation Logic (Reynolds’02) is used in Hoare proof systems for program verification of languages with pointers.

Hoare calculus: axioms and rules reason about triples:

\[
\{ \varphi \} \text{P} \{ \varphi' \}
\]

Any (memory) model that satisfies \( \varphi \) will satisfy \( \varphi' \) after being modified by the program \( \text{P} \).

Tools: Infer (Facebook), SLAyer (Microsoft)...

Also, see “Why Separation Logic Works” (Pym et al. ’18)
Memory states

Separation Logic is interpreted over memory states \((s, h)\) where:

- store, \(s : \text{VAR} \rightarrow \text{LOC}\)
- heap, \(h : \text{LOC} \rightarrow_{\text{fin}} \text{LOC}\)

where \(\text{VAR} = \{x, y, z, \ldots\}\) set of (program) variables, \(\text{LOC}\) set of locations. \(\text{VAR}\) and \(\text{LOC}\) are countably infinite sets.

- Disjoint heaps: \(\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset\)
- Sum of disjoint heaps \((h_1 + h_2)\) is defined as the sum of partial functions.
Propositional Separation Logic $SL(\ast, \neg\ast)$

$\varphi := \neg\varphi \mid \varphi_1 \land \varphi_2 \mid \text{emp} \mid x = y \mid x \leftrightarrow y \mid \varphi_1 \ast \varphi_2 \mid \varphi_1 \ast\neg\varphi_2$

Semantics

- standard for $\land$ and $\neg$;
- $(s, h) \models \text{emp} \iff \text{dom}(h) = \emptyset$
- $(s, h) \models x = y \iff s(x) = s(y)$
- $(s, h) \models x \leftrightarrow y \iff h(s(x)) = s(y)$
Separating conjunction (*)

\[(s, h) \models \varphi_1 \ast \varphi_2 \text{ if and only if }\]

\[\exists h_1 \exists h_2 \Rightarrow (s, h_1) \models \varphi_1 \text{ and } (s, h_2) \models \varphi_2\]

There is a way to split the heap into two so that, together with the store, one part satisfies \(\varphi_1\) and the other satisfies \(\varphi_2\).
Separating implication (\(\rightarrow\))

\[(s, h) \models \varphi_1 \rightarrow \varphi_2 \text{ if and only if }\]

\[\forall h_1 \quad \text{dom}(h) \cap \text{dom}(h_1) = \emptyset \quad \text{and} \quad (s, h_1) \models \varphi_1 \quad \text{implies} \quad (s, h + h_1) \models \varphi_2\]

Whenever a (disjoint) heap that, together with the store, satisfies \(\varphi_1\) is added, the resulting memory state satisfies \(\varphi_2\).
Hoare proof-system requires to solve classical problems: satisfiability/validity/entailment

\[ \varphi \Rightarrow \psi \quad \{\psi\} \mathsf{P} \quad \{\psi'\} \quad \psi' \Rightarrow \varphi' \]

consequence rule

Satisfiability is PSPACE-complete for \( SL(*, \rightarrow) \)

Note: entailment and validity reduce to satisfiability for \( SL(*, \rightarrow) \).
Robustness properties

- **Acyclicity** holds for \( \varphi \) iff every model of \( \varphi \) is acyclic.
- **Garbage freedom** holds for \( \varphi \) iff in every model of \( \varphi \), each \( \ell \in \text{dom}(h) \) is reachable from a program variable of \( \varphi \).

C. Jansen et al., ESOP’17

Checking for robustness properties is EXPSPACE-complete for Symbolic Heaps with Inductive Predicates.

- Symbolic Heaps \( \Rightarrow \) no \( \ast \), no \( \land \) and \( \neg \) inside \( \ast \)
- Inductive Predicates \( \sim \) Horn clauses where \( \ast \) replaces \( \land \)

\[
P(\vec{x}) \iff \exists \vec{z} \; Q_1 \ast \ldots \ast Q_n \quad \text{fv}(Q_i) \subseteq \vec{x}, \vec{z}
\]

Our Goal
Provide similar results for **propositional** separation logic.
Desiderata

We aim to an extension of propositional separation logic where

- satisfiability is decidable in PSPACE (as $SL(\ast, \neg \ast)$)
- robustness properties reduce to one of these problems

Known extensions

\[
\begin{align*}
&\text{2SL}(\ast, \neg \ast) \quad \text{UNDECIDABLE} \\
\downarrow & \quad \downarrow \\
\text{1SL}(\ast, \neg \ast) & \quad \text{PSPACE} \\
\downarrow & \quad \downarrow \\
\text{SL}(\ast, \neg \ast) & \quad \text{SL}(\forall, \ast) \quad \text{(NON-ELEM.)} \\
\downarrow & \\
\text{SL}(\ast, \text{reach}) &
\end{align*}
\]
SL(\(*, \neg\*) + reachability and 1 quantified variable

- \((s, h) \models \text{reach}^+(x, y) \iff h^L(s(x)) = s(y)\) for some \(L \geq 1\)
- \((s, h) \models \exists u \varphi \iff\) there is \(\ell \in \text{LOC}\) s.t. \((s[u \leftarrow \ell], h) \models \varphi\)

It is only possible to quantify over the variable name \(u\).

Robustness properties reduce to entailment

- **Acyclicity**: \(\varphi \models \neg\exists u \text{reach}^+(u, u)\)
- **Garbage freedom**: \(\varphi \models \forall u (\text{alloc}(u) \Rightarrow \bigvee_{x \in \text{fv(}\varphi)} \text{reach}(x, u))\)

where \(u \notin \text{fv(}\varphi)\) and
- \(\text{alloc}(x) \overset{\text{def}}{=} (x \leftarrow x) \rightarrow \bot\)
- \(\text{reach}(x, y) \overset{\text{def}}{=} x = y \lor \text{reach}^+(x, y)\)
Restrictions

The logic $1\text{SL}(\ast, \neg\ast, \text{reach}^+)$ is undecidable. We syntactically restrict the logic so that for each occurrence of $\text{reach}^+(x, y)$:

\begin{itemize}
  \item \textbf{R1} it is not on the right side of its first $\neg\ast$ ancestor (seeing the formula as a tree)
    \begin{itemize}
      \item $\varphi \neg\ast (\psi \ast \text{reach}^+(u, u))$ violates \textbf{R1}
    \end{itemize}
  \item \textbf{R2} if $x = u$ then $y = u$ (syntactically)
    \begin{itemize}
      \item $\text{reach}^+(u, x)$ violates \textbf{R2}
    \end{itemize}
\end{itemize}

\textbf{Note}: robustness properties formulae are still expressible.
Results

1. $1SL_{R_1}(\ast, \neg\ast, \text{reach}^+)$: satisfiability is NON-ELEMENTARY (more precisely, TOWER-hard)

2. $1SL_{R_1}^R(\ast, \neg\ast, \text{reach}^+)$: satisfiability is PSPACE-complete

Proof Techniques

1. reduce *Propositional interval temporal logic under locality principle (PITL)* to a logic captured by $1SL_{R_1}(\ast, \neg\ast, \text{reach}^+)$

2. extend the *test formulae technique* used for $SL(\ast, \neg\ast)$
PITL (Moszkowski’83)

\[ \varphi := pt \mid a \mid \varphi_1 \varphi_2 \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \]

- Interpreted on finite non-empty words over a finite alphabet \( \Sigma \)

- \( w \models pt \iff |w| = 1 \)

- \( w \models a \iff \text{first letter of } w \text{ is } a \in \Sigma \) (locality principle)

- \( w \models \varphi_1 \varphi_2 \iff w[1:j] \models \varphi_1 \) and \( w[j:|w|] \models \varphi_2 \)
  
  for some \( j \in [1,|w|] \)

- Satisfiability is decidable, but NON-ELEMENTARY
Auxiliary Logic on Trees (ALT)

\[ \varphi := \varphi_1 \land \varphi_2 \mid \neg \varphi \mid \varphi_1 \ast \varphi_2 \mid \exists u \varphi \mid T(u) \mid G(u) \]

- interpreted on acyclic memory states (set of rooted trees)
- one special tree, rooted in \( \rho \in \text{LOC} \)
- \( \exists u \varphi \) and \( \varphi_1 \ast \varphi_2 \) as before
- \((s, h) \models_\rho T(u) \) iff \( s(u) \in \text{dom}(h) \) and it does reach \( \rho \)
- \((s, h) \models_\rho G(u) \) iff \( s(u) \in \text{dom}(h) \) and it does not reach \( \rho \)

Note: ALT is captured by \( 1\text{SL}_{R1}(*, \neg*, \text{reach}^+) \).
Reducing PITL to ALT

- Easy to encode words as acyclic memory states

\[
\text{abaa} \quad \sim \sim \sim \sim
\]

- Set of models encoding words can be characterised in ALT

- However, difficult to translate \( \varphi_1 \mid \varphi_2 \): cannot express properties about the trees not rooted in \( \rho \), apart from their size

\[
\text{After the cut, left side does not reach } \rho \text{ anymore.}
\]
PITL to ALT: alternative semantics for PITL

- **a** marked representation of \( a \in \Sigma \)

\[
\begin{array}{c}
\text{mark} \quad w_1 \ldots w_{j-1} \ w_j \ w_{j+1} \ldots \ w_{|w|} \\
\text{var} \quad w_j
\end{array}
\]

- \( \varphi \mid \psi \) on standard semantics:

\[
\begin{array}{c}
\text{mark} \quad w_1 \ldots w_{j-1} \ w_j \\
\varphi_1
\end{array} \quad \begin{array}{c}
\text{mark} \quad w_j \ w_{j+1} \ldots w_{|w|} \\
\varphi_2
\end{array}
\]

- \( \varphi \mid \psi \) on marked semantics (can be simulated in ALT)

\[
\begin{array}{c}
\text{mark} \quad w_1 \ldots w_{j-1} \ w_j \ w_{j+1} \ldots \ w_{|w|} \\
\varphi_1
\end{array} \quad \begin{array}{c}
\text{mark} \quad w_j \ w_{j+1} \ldots w_{|w|} \\
\varphi_2
\end{array}
\]

1. ALT and \( 1\text{SL}_{R1}(\ast, \ast, \text{reach}^+) \) are NON-ELEMENTARY

2. ALT is decidable in TOWER, as it is captured by \( \text{SL}(\forall, \ast) \)
$1SL_{R_1}^{R_2}(\ast, \neg, \text{reach}^+) \text{ is in PSPACE}$
$\text{ISL}_{R_1}^{R_2}(*, *, \text{reach}^+) \text{ is in PSPACE}$

Test Formulae “technique”
Design an equivalence relation on models, based on the satisfaction of atomic predicates (test formulae), e.g.

\[ x = y \quad x \hookrightarrow y \quad \text{alloc}(x) \quad \text{size} \geq \beta \]

Show that any formula of our logic is equivalent to a Boolean combination of test formulae, e.g.

\[ (x \hookrightarrow y) \ast \neg \text{emp} \iff x \hookrightarrow y \land \text{size} \geq 2 \]

Prove small-model property for the logic of test formulae.
1: Designing Test Formulae

- Fix $\alpha \in \mathbb{N}^+, \ X \subseteq_{\text{fin}} \text{VAR}$
- Let us define $\text{Test}(X, \alpha)$ as the finite set of predicates:
  $$\{ x = y, \ x \leftrightarrow y, \ \text{alloc}(x), \ \text{size} \geq \beta \ | \ \beta \in [1, \alpha], \ x, y \in X \}$$

### Indistinguishability relation $(s, h) \approx_{X, \alpha} (s', h')$

for every $\varphi \in \text{Test}(X, \alpha)$, $(s, h) \models \varphi$ iff $(s', h') \models \varphi$

**Note:** $\alpha$ is related to the number of occurrences of $\ast$ and $\neg \ast$ in a formula of separation logic.
2: * elimination Lemma

We want to design $\text{Test}(x, \alpha)$ so that the following results hold

- For every $\varphi \in \text{Bool}(\text{Test}(x, \alpha_1))$, $\psi \in \text{Bool}(\text{Test}(x, \alpha_2))$ there is $\gamma \in \text{Bool}(\text{Test}(x, \alpha_1 + \alpha_2))$ such that

  $$\varphi \ast \psi \iff \gamma$$

- Similar elimination result for $\neg \ast$.

Lemmata holds for

$$\text{Test}(x, \alpha) = \begin{cases} x = y, \ x \hookrightarrow y \\ \text{alloc}(x), \ \text{size} \geq \beta \end{cases} \quad \mid \beta \in [1, \alpha]$$

$x, y \in X$
3: Test formulae, after \(*\) and \(\rightarrow\) elimination

**Hypothesis:** A family of test formulae, such that
- captures the atomic predicates of \(\text{SL}(\ast, \rightarrow)\)
- satisfies the \(\ast\) and \(\rightarrow\) elimination Lemmata

**Thesis:** for every formulae \(\varphi\) of \(\text{SL}(\ast, \rightarrow)\),
- \(\varphi\) is equivalent to a Boolean combination of test formulae.
- If \(\alpha \geq |\varphi|\), \(X \supseteq v(\varphi)\) and \((s, h) \approx^X_\alpha (s', h')\) then

\[(s, h) \models \varphi \iff (s', h') \models \varphi.\]

**Small-model property derived from** \(\approx^X_\alpha\)
- Small-model property for Boolean combination of test formulae carries over to \(\text{SL}(\ast, \rightarrow)\).
- test formulae in \(\text{PSPACE} \implies \text{SL}(\ast, \rightarrow)\) is in \(\text{PSPACE}\).
1SL\textsuperscript{R1}(\ast, \neg\ast, \text{reach}^+) is in PSPACE

\[ \pi := x = y \mid x \leftrightarrow y \mid \text{emp} \mid A \rightarrow C \quad (R1) \]
\[ C := \pi \mid C \wedge C \mid \neg C \mid \exists u C \mid C \ast C \]
\[ A := \pi \mid \text{reach}^+(v_1, v_2) \mid A \wedge A \mid \neg A \mid \exists u A \mid A \ast A \]

where (R2) if \( v_1 = u \) then \( v_2 = u \)

Not so easy...

- Asymmetric \( A \rightarrow C \).
  - two sets of test formulae: two \( \ast/\exists \) elimination Lemmata
  - \( \neg\ast \) elimination Lemma that glues the two set of test formulae

- instead of “\text{size} \geq \beta \text{ s.t. } \beta \in [1, \alpha]”, the \( \beta \)s of new test formulae are bounded by functions on \( \alpha \), e.g.

\[ \#\text{loop}_x(\beta) \geq \gamma \quad \gamma \in [1, \frac{1}{2} \alpha(\alpha + 3) - 1] \]

bounds are found by solving a set of recurrence equations!
Recap

\[ SL(\ast, \ast, reach) \quad \text{undec.} \]

\[ 1SL_{R1}(\ast, \ast, reach^+) \quad \text{unk. non-elem.} \]

\[ 1SL_{R2}(\ast, \ast, reach^+) \quad \text{PSPACE-complete} \]

\[ SL(\ast, reach) \quad \text{PSPACE-complete} \]

\[ ALT \quad \text{dec. non-elem.} \]

\[ PITL \quad \text{dec. non-elem.} \]

- \( 1SL_{R2}(\ast, \ast, reach^+) \) strictly generalise other PSPACE-compl. extensions of propositional separation logic
- Can be used to check for robustness properties.
ALT seems to be an interesting tool for reductions, as it is a fragment or it is easily captured by many logics in TOWER e.g. $\text{QCTL}(U)$, $\text{MSL}(\Diamond, \langle U \rangle, *)$, $2\text{SL}(\cdot)$.