Extending propositional separation logic for robustness properties

Alessio Mansutti

LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, Cachan, France

FSTTCS - December 2018
Separation logic and program verification

**Hoare calculus** is based on proof rules manipulating Hoare triples.

\[
\{\varphi\} \ C \ \{\varphi'\}
\]

where

- \(C\) is a program
- \(\varphi\) (precondition) and \(\varphi'\) (postcondition) are assertions in some logical language.

Any (memory) model that satisfies \(\varphi\) will satisfy \(\varphi'\) after being modified by \(C\).
Programming languages with pointers

The so-called **rule of constancy**

\[
\begin{array}{c}
\{\varphi\} \quad C \quad \{\varphi'\} \\
\{\varphi \land \psi\} \quad C \quad \{\varphi' \land \psi\}
\end{array}
\]

"C does not mess with $\psi$"

is generally not valid: it is unsound if $C$ manipulates pointers.
Programming languages with pointers

The so-called rule of constancy

\[ \{ \varphi \} \ C \ \{ \varphi' \} \]

\[ \{ \varphi \land \psi \} \ C \ \{ \varphi' \land \psi \} \]

“C does not mess with ψ”

is generally not valid: it is unsound if C manipulates pointers.

Example:

\[ \{ \exists u. [x] = u \} \ [x] \leftarrow 4 \ \{ [x] = 4 \} \]

\[ \{ [y] = 3 \land \exists u. [x] = u \} \ [x] \leftarrow 4 \ \{ [y] = 3 \land [x] = 4 \} \]

not true if x and y are in aliasing.
Separation logic (Reynolds’02)

Separation logic add the notion of \textit{separation} (\(\ast\)) of a state, so that the \textbf{frame rule}

\[
\begin{align*}
\{\varphi\} C \{\varphi'\} \quad \text{modv}(C) \cap \text{fv}(\psi) = \emptyset \\
\{\varphi * \psi\} C \{\varphi' * \psi\}
\end{align*}
\]

is valid.

Intuitively, separation means \(([x] = n \ast [y] = m) \implies x \neq y\)
Separation logic (Reynolds’02)

Separation logic add the notion of separation (*) of a state, so that the frame rule

\[
\begin{align*}
\{ \varphi \} & C \{ \varphi' \} \quad \text{modv}(C) \cap \text{fv}(\psi) = \emptyset \\
\{ \varphi * \psi \} & C \{ \varphi' * \psi \}
\end{align*}
\]

is valid.

Intuitively, separation means \([x] = n * [y] = m) \implies x \neq y

- **Automatic Verifiers**: Infer, SLAyer, Predator
- **Semi-automatic Verifiers**: Smallfoot, Verifast

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 (typically \(\text{LOC} \cong \mathbb{N} \cong \text{VAR}\)).

Disjointed heaps: \(\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset\)

Sum of disjoint heaps \((h_1 + h_2) = \text{sum of partial functions}\)
Propositional Separation Logic $\text{SL}(\ast, \neg\ast)$

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

Semantics

- standard for $\land$ and $\neg$;
- $(s, h) \models \text{emp} \iff \text{dom}(h) = \emptyset$
- $(s, h) \models x = y \iff s(x) = s(y)$
- $(s, h) \models x \hookrightarrow y \iff h(s(x)) = s(y)$, (previously $[x] = y$)
Separating conjunction (∗)

\((s, h) \models \varphi_1 \ast \varphi_2\) if and only if

\(\exists h_1\) \(\exists h_2\)

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 ($\nabla^\ast$)

$$(s, h) \models \varphi_1 \nabla^\ast \varphi_2 \text{ if and only if }$$

$$\forall h_1 \quad \text{dom}(h) \cap \text{dom}(h_1) = \emptyset$$

$$(s, h_1) \models \varphi_1$$

$$(s, h + h_1) \models \varphi_2$$

Whenever a (disjoint) heap that, together with the store, satisfies $\varphi_1$ is added, the resulting memory state satisfies $\varphi_2$. 
Decision Problems

- Hoare proof-system requires to solve classical problems:
  - satisfiability/validity/entailment
  - weakest precondition/strongest postcondition

\[
P \implies P' \quad \{P'\} \quad C \quad \{Q'\} \quad Q' \implies Q
\]

consequence rule

- satisfiability is \( \text{PSPACE} \)-complete for \( \text{SL}(\ast, \neg \ast) \)

**Note:** entailment and validity reduce to satisfiability for \( \text{SL}(\ast, \neg \ast) \).
Robustness properties

- **Acyclicitiy** holds for $\varphi$ iff every model of $\varphi$ is acyclic
- **Garbage freedom** holds for $\varphi$ iff in every model of $\varphi$, each memory cell is reachable from a program variable of $\varphi$

C. Jansen et al., ESOP’17

**Checking for robustness properties** is $\text{ExpSpace}$-complete for Symbolic Heaps with Inductive Predicates.

- Symbolic Heaps $\implies$ no negation, no $\neg$, no $\land$ inside $\ast$
- Inductive Predicates: akin of Horn clauses where $\ast$ replaces $\land$

$$P(\bar{x}) \leftarrow \exists \bar{z} \ Q_1 \ast \ldots \ast Q_n \quad \text{fv}(Q_i) \subseteq \bar{x}, \bar{z}$$

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

We aim to an extension of propositional separation logic where
- satisfiability, validity and entailment are decidable
- in $\text{PSPACE}$ (as propositional separation logic)
- robustness properties reduce to one of these problems

Known extensions
**SL(\(*, \not\rightarrow\)) + reachability and one 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 \(u \not\in \text{fv}(\varphi)\) and

- \(\text{alloc}(x) \overset{\text{def}}{=} x \leftarrow x \not\rightarrow \bot\)
- \(\text{reach}(x, y) \overset{\text{def}}{=} x = y \lor \text{reach}^+(x, y)\)
Restrictions

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

- **R1** it is not on the right side of its first $\ast$ ancestor (seeing the formula as a tree)
- **R2** if $x = u$ then $y = u$ (syntactically)

For example, given $\varphi, \psi$ satisfying these conditions,

- $\text{reach}^+(u, x) \ast (\varphi \ast \psi)$ only satisfies R1
- $\varphi \ast (\text{reach}^+(x, u) \ast \psi)$ satisfies both R1 and R2
- $\varphi \ast (\psi \ast \text{reach}^+(u, u))$ only satisfies R2

**Note:** robustness properties are expressible in this fragment.
Results

0 Weakening even slightly $R1$ leads to undecidability

1 $1SL_{R1}(\ast, \neg\ast, \text{reach}^+)$: satisfiability is NON-ELEMENTARY (more precisely, TOWER-hard)

2 $1SL_{R1}^R(\ast, \neg\ast, \text{reach}^+)$: satisfiability is $PSPACE$-complete

Proof Techniques

(1) reduce *Propositional interval temporal logic under locality principle* (PITL) to a logic captured by $1SL_{R1}(\ast, \neg\ast, \text{reach}^+)$

(2) extend the *test formulae technique* used for $SL(\ast, \text{reach})$
PITL (Moszkowski’83)

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

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

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

- \( w \models a \iff w \) headed by \( a \) (locality principle)

- \( w \models \varphi_1 \mid \varphi_2 \iff w[1 : j] \models \varphi_1 \) 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
- one special location: the root \( \rho \) of a tree
- \((s, h) \models T(u)\) iff \(s(u) \in \text{dom}(h)\) and it does reach \( \rho \)
- \((s, h) \models G(u)\) iff \(s(u) \in \text{dom}(h)\) and it does not reach \( \rho \)
- \(\exists u \; \varphi \) and \(\varphi_1 \ast \varphi_2\) as before

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

- Easy to encode words as acyclic memory states

\[
\text{abaa} \quad \sim \sim \sim \rightarrow \quad a \quad b \quad a \quad a \quad a
\]

- Set of models encoding words can be characterised in ALT

- However, difficult to translate \( \varphi_1 \mathbin{\mid} \varphi_2 \):
  ALT cannot express properties about the set of locations in \( \text{dom}(h) \) that do not reach \( \rho \), apart from its size

\[
\text{abaa} \quad \sim \sim \sim \rightarrow \quad a \quad b \quad a \quad a \quad a
\]

\[
\begin{array}{c}
    \varphi_1 \\
    \varphi_2
\end{array}
\]

After the cut, left side does not reach \( \rho \) anymore.
Reducing PITL to ALT: alternative semantics for PITL

- A marked representation of a

\[
\begin{array}{c}
\ldots w_{j-1} w_j w_{j+1} \ldots w_{\mid w} \\
\phi_1
\end{array}
\]

- $\phi \models \psi$ on standard semantics:

\[
\begin{array}{c}
\begin{array}{c}
\ldots w_{j-1} w_j \\
\phi_1
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
w_j w_{j+1} \ldots w_{\mid w} \\
\phi_2
\end{array}
\end{array}
\end{array}
\]

- $\phi \models \psi$ on marked semantics (can be simulated in ALT)

\[
\begin{array}{c}
\begin{array}{c}
\ldots w_{j-1} \boxed{w_j} w_{j+1} \ldots w_{\mid w} \\
\phi_1
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
w_j w_{j+1} \ldots w_{\mid w} \\
\phi_2
\end{array}
\end{array}
\end{array}
\]

1. ALT and $1\text{SL}_{R_1}(\ast, -, \text{reach}^+)$ are NON-ELEMENTARY

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

Test Formulae “technique”
Test formulae example on a Toy Logic

\[ \varphi := \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \ast \varphi_2 \mid \exists u \varphi \mid \text{alloc}(u) \mid u \xrightarrow{2} u \]

where \((s, h) \models u \xrightarrow{2} u\) iff \(h(s(u)) = \ell \neq s(u)\) and \(h(\ell) = s(u)\).

Some formulae:

- \#loops(2) \geq \beta \overset{\text{def}}{=} \exists u u \xrightarrow{2} u \ast \ldots \ast \exists u u \xrightarrow{2} u \]

- \(H_1 \overset{\text{def}}{=} \exists u \text{alloc}(u) \land \neg (\exists u \text{alloc}(u) \ast \exists u \text{alloc}(u)) \]

- \(\text{rem} \geq 0 \overset{\text{def}}{=} \top \]

- \(\text{rem} \geq \beta + 1 \overset{\text{def}}{=} \\
\exists u : \text{alloc}(u) \land \neg u \xrightarrow{2} u \land ((\text{alloc}(u) \land H_1) \ast \text{rem} \geq \beta)) \)
Test Formulae

1 Design an equivalence relation on models, based on the satisfaction of atomic predicates (test formulae), e.g.

\[ \#\text{loops}(2) \geq \beta \quad \text{rem} \geq \beta \]

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

\[ \#\text{loops}(2) \geq 3 \times \#\text{loops}(2) \geq 5 \iff \#\text{loops}(2) \geq 8 \]

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

- Fix $\alpha \in \mathbb{N}^+$

- Let $\text{Test}(\alpha)$ be the finite set of predicates:
  \[
  \{ \#\text{loops}(2) \geq \beta, \text{rem} \geq \gamma \mid \beta \in [1, \mathcal{L}(\alpha)], \gamma \in [1, \mathcal{G}(\alpha)] \}
  \]
  for some functions $\mathcal{L}$ and $\mathcal{G}$ in $[\mathbb{N} \rightarrow \mathbb{N}]$

Indistinguishability relation $(s, h) \approx_\alpha (s', h')$

for every $T \in \text{Test}(\alpha)$, $(s, h) \models T$ iff $(s', h') \models T$

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

We want to design $\text{Test}(\alpha)$ so that the following result holds

**Hypothesis:**
- $(s, h) \approx_\alpha (s', h')$
- $\alpha_1, \alpha_2 \in \mathbb{N}^+$ s.t. $\alpha_1 + \alpha_2 = \alpha$
- $h_1 + h_2 = h$

**Thesis:** there are $h'_1, h'_2$ s.t.
- $h'_1 + h'_2 = h'$
- $(s, h_1) \approx_{\alpha_1} (s', h'_1)$
- $(s, h_2) \approx_{\alpha_2} (s', h'_2)$

**Note:** it can be restated as an EF-style game. Spoiler splits $\alpha$ and $h$, Duplicator has to mimic the split on $h'$ so that $\approx$ still holds.
We want to design Test(\(\alpha\)) so that the following result holds

Hypothesis:
- \((s, h) \approx_\alpha (s', h')\)
- \(\alpha_1, \alpha_2 \in \mathbb{N}_+\) s.t. \(\alpha_1 + \alpha_2 = \alpha\)
- \(h_1 + h_2 = h\)

Thesis:
- There are \(h'_1, h'_2\) s.t.
  \[ (s, h'_1) \approx_{\alpha_1} (s', h'_1) \]
  \[ (s, h'_2) \approx_{\alpha_2} (s', h'_2) \]

Note: it can be restated as an EF-style game. Spoiler splits \(\alpha\) and \(h\), Duplicator has to mimic the split on \(h'\) so that \(\approx\) still holds.
Finding $G$ for $\text{rem} \geq \gamma$ formulae

Given $h = h_1 + h_2$, every location not in a loop of size 2 of $h$ cannot be in a loop of size 2 of $h_1$ or $h_2$. Then $G$ must satisfy

$$G(\alpha) \geq \max_{\alpha_1,\alpha_2 \in \mathbb{N}^+} (G(\alpha_1) + G(\alpha_2))$$

Finding $L$ for $\#\text{loops}(2) \geq \beta$ formulae

Take $h = h_1 + h_2$. Given a loop of size 2 of $h$, two cases:

- both locations of the loop are in the same heap ($h_1$ or $h_2$);
- one location of the loop is in $h_1$ and the other is in $h_2$.

$$L(\alpha) \geq \max_{\alpha_1,\alpha_2 \in \mathbb{N}^+} (L(\alpha_1) + L(\alpha_2) + G(\alpha_1) + G(\alpha_2))$$
Finding $\mathcal{L}$ and $\mathcal{G}$

We have the inequalities

$$
\mathcal{G}(1) \geq 1 \quad \mathcal{G}(\alpha) \geq \max_{\alpha_1, \alpha_2 \in \mathbb{N}^+} (\mathcal{G}(\alpha_1) + \mathcal{G}(\alpha_2))
$$

$$
\mathcal{L}(1) \geq 1 \quad \mathcal{L}(\alpha) \geq \max_{\alpha_1, \alpha_2 \in \mathbb{N}^+} (\mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{G}(\alpha_1) + \mathcal{G}(\alpha_2))
$$

Which admit $\mathcal{G}(\alpha) = \alpha$ and $\mathcal{L}(\alpha) = \frac{1}{2}\alpha(\alpha + 3) - 1$ as a solution.

An indistinguishability relation built on the set

$$
\begin{align*}
\left\{
\begin{array}{l}
\#\text{loops}(2) \geq \beta, \\
\text{rem} \geq \gamma
\end{array}
\right\}
\quad \beta \in \left[1, \frac{1}{2}\alpha(n + 3) - 1\right]\quad \gamma \in [1, \alpha]
\end{align*}
$$

satisfy the * elimination Lemma.
(3) Test formulae, after $\ast$ elimination

**Hypothesis:** Two family of test formulae, such that
- captures the atomic predicates of the Toy Logic
- satisfies the $\ast$ elimination Lemma (and $\exists$ elimination Lemma)

**Thesis:** for every formulae $\varphi$ of Toy Logic, by taking $\alpha \geq |\varphi|$ we have
- If $(s, h) \approx_{\alpha} (s, h')$ then we have $(s, h) \models \varphi$ iff $(s, h') \models \varphi$.
- $\varphi$ is equivalent to a Boolean combination of test formulae.

**Small-model property**

1. Small-model property for Boolean combination of test formulae carries over to Toy Logic.
2. All bounds are polynomial $\implies$ test formulae in $\text{PSPACE}$
3. Toy Logic is in $\text{PSPACE}$
1\text{SL}_{R_1}^{R_2}(\ast, \neg\ast, \text{reach}^+) is in PSPACE

\begin{align*}
\pi & := x = y \mid x \leftrightarrow y \mid \text{emp} \mid A \neg\ast C \, (R_1) \\
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
\end{align*}

where \((R_2)\) if \(v_1 = u\) then \(v_2 = u\)

Not so easy...

- Find the right set of test formulae that capture the logic
- Asymmetric \(A \ast C\).
  - two indistinguishability relation, two sets of test formulae
  - two \(\ast\) and two \(\exists\) elimination Lemmata
  - \(\ast\) elimination Lemma that glues the two relations
If you like bounds: Test($X, \alpha$) for the $A$ fragment

\[
\begin{align*}
\begin{cases}
    v_1 = v_2, \text{ sees}_X(v_1, v_2) \geq \beta^\downarrow \\
    \#\text{loop}_X(\beta) \geq \beta^\circ, \ #\text{loop}_X^\uparrow \geq \beta^\circ \\
    \#\text{pred}_X^A(x) \geq \beta, \ \text{size}_X^A \geq \beta \\
    u \in \text{sees}_X(v_1, v_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta}) \\
    u = v_1, u \in \text{loop}_X(\beta), u \in \text{loop}_X^\uparrow \\
    u \in \text{pred}_X^A(x), u \in \text{size}_X^A \\
\end{cases}
\end{align*}
\]

\[
\begin{align*}
    \beta^\downarrow & \in [1, \frac{1}{6}(\alpha + 1)(\alpha + 2)(\alpha + 3)] \\
    \beta^\circ & \in [1, \frac{1}{2}\alpha(\alpha + 3) - 1], \ \beta \in [1, \alpha] \\
    \overleftarrow{\beta} & \in [1, \frac{1}{6}\alpha(\alpha + 1)(\alpha + 2) + 1] \\
    \overrightarrow{\beta} & \in [1, \frac{1}{2}\alpha(\alpha + 3)] \\
    x \in X, \ v_1, v_2 \in \text{ATERM}_X
\end{align*}
\]
Recap

- $1SL_{R_1}(*, \neg, \text{reach}^+)$ strictly generalise other
  PSPACE-complete extensions of propositional separation logic
- Can be used to check for robustness properties
Recap

- 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. QCTL(∪), MSL(◇, ⟨∪⟩, *), 2SL(*)