Separation logic and fragments: from expressive power to decision procedures

Stéphane Demri

CNRS – Marie Curie Fellow

Yorktown Heights, March 2015
In Memoriam: Morgan Deters
Overview

1. Separation Logic in a Nutshell
2. Expressive Power of 1SL
3. Playing with 1SL Restricted to a Single Variable (1SL1)
4. Expressiveness/Decision Procedure for 1SL1
5. A Taste of Other Decision Procedures
Separation Logic in a Nutshell
Floyd-Hoare logic

- Hoare triple: \( \{ \phi \} \ C \ \{ \psi \} \) (partial correctness).
  
  [Hoare, C. ACM 69; Floyd, 1967]

- If we start in a state where \( \phi \) holds true and the command \( C \) terminates, then it yields a state in which \( \psi \) holds.

\[\text{Separation Logic in a Nutshell}\]
Floyd-Hoare logic

- Hoare triple: \( \{ \phi \} \ C \ {\psi} \) (partial correctness).
  
  [Hoare, C. ACM 69; Floyd, 1967]

- If we start in a state where \( \phi \) holds true and the command \( C \) terminates, then it yields a state in which \( \psi \) holds.

- Commands can be composed:
  
  \[
  \begin{array}{c}
  \{ \phi \} \ C \_1 \ {\psi} \quad \{ \psi \} \ C \_2 \ {\varphi} \\
  \{ \phi \} \ C \_1 \_; \ C \_2 \ {\varphi}
  \end{array}
  \]

- Strengthening preconditions / weakening postconditions:
  
  \[
  \phi \to \phi' \quad \{ \phi' \} \ C \ {\psi} \quad \psi \to \psi'
  \]

  \[
  \{ \phi \} \ C \ {\psi'}
  \]

- Rule of constancy:
  
  \[
  \begin{array}{c}
  \{ \phi \} \ C \ {\psi} \\
  \{ \phi \land \psi' \} \ C \ {\psi \land \psi'}
  \end{array}
  \]

  where no variable free in \( \psi' \) is modified by \( C \).
When separation logic enters into the play

- Unsoundness of the rule of constancy with pointers:

\[
\frac{\{ \exists u \ (x \mapsto u) \} \ [x] := 4 \ \{ x \mapsto 4 \}}{\{(\exists u \ (x \mapsto u)) \land y \mapsto 3\} \ [x] := 4 \ \{ x \mapsto 4 \land y \mapsto 3 \}}
\]

\( x \mapsto u \): “memory has a unique memory cell \( x \mapsto u \)”
When separation logic enters into the play

- Unsoundness of the rule of constancy with pointers:

\[
\begin{align*}
\{ \exists u \ (x \mapsto u) \} \ [x] & := 4 \ \{ x \mapsto 4 \} \\
\{(\exists u \ (x \mapsto u)) \land y \mapsto 3 \} \ [x] & := 4 \ \{ x \mapsto 4 \land y \mapsto 3 \}
\end{align*}
\]

\(x \mapsto u\): “memory has a unique memory cell \(x \mapsto u\)”

- Reparation with frame rule:

\[
\begin{align*}
\{ \phi \} & \ C \ \{ \psi \} \\
\{ \phi \ast \psi' \} & \ C \ \{ \psi \ast \psi' \}
\end{align*}
\]

where no variable free in \(\psi'\) is modified by \(C\).
On separation logic

• Introduced by Ishtiaq, O’Hearn, Pym, Reynolds, Yang.

• Extension of Hoare logic with separating connectives. [Reynolds, LICS’02]

• Reasoning about the heap with a strong form of locality built-in.

• In a broad sense:

  assertion logic + programming language + specification logic
On separation logic

- Introduced by Ishtiaq, O'Hearn, Pym, Reynolds, Yang.
- Extension of Hoare logic with separating connectives. [Reynolds, LICS’02]
- Reasoning about the heap with a strong form of locality built-in.
- In a broad sense:
  assertion logic + programming language + specification logic

- A taste of separation logic:
  - Models can be finite graphs.
  - Binary modalities $\ast$ and $\ast\ast$.
  - While evaluating a formula, models can be updated.
Memory states with one record field

- Program variables $PVAR = \{x_1, x_2, x_3, \ldots\}$.

- Memory state:
  - Store $s : PVAR \rightarrow \mathbb{N}$.
  - Heap $h : \mathbb{N} \rightarrow \mathbb{N}$ with finite domain.
    (here, no distinction between locations and values)

Separation Logic in a Nutshell
Disjoint heaps

- Disjoint heaps: \(\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset\) (noted \(h_1 \perp h_2\)).

- When \(h_1 \perp h_2\), disjoint heap \(h_1 \sqcup h_2\).
Syntax and semantics for 1SL

- Quantified variables $FVAR = \{u_1, u_2, u_3, \ldots\}$.

- Expressions: $e ::= x_i \mid u_j$

- Atomic formulae: $\pi ::= e = e' \mid e \leftrightarrow e' \mid \text{emp} \mid \bot$

- Formulae: $\phi ::= \pi \mid \phi \land \psi \mid \neg \phi \mid \phi \ast \psi \mid \phi \rightarrow \psi \mid \exists u \phi$
Syntax and semantics for 1SL

- **Quantified variables** \( FVAR = \{u_1, u_2, u_3, \ldots\} \).

- **Expressions:** \( e ::= x_i \mid u_j \)

- **Atomic formulae:** \( \pi ::= e = e' \mid e \hookrightarrow e' \mid \text{emp} \mid \bot \)

- **Formulae:** \( \phi ::= \pi \mid \phi \land \psi \mid \neg \phi \mid \phi \ast \psi \mid \phi \rightarrow \psi \mid \exists u \; \phi \)

- \((s, h) \models f \text {emp} \iff \text {dom}(h) = \emptyset.\)

- \((s, h) \models f e = e' \iff [e] = [e'], \) \text {with} \([x] \stackrel{\text{def}}{=} s(x), [u] \stackrel{\text{def}}{=} f(u).\)

- \((s, h) \models f e \hookrightarrow e' \iff [e] \in \text {dom}(h) \) and \( h([e]) = [e'].\)
Binary modality: separating conjunction

\[(s, \; h) \models_f \; \phi_1 \; \ast \; \phi_2\]

\[\iff\]

for some \(h_1, \; h_2\) such that \(h = h_1 \uplus h_2\),

\[(s, \; h_1) \models_f \; \phi_1 \; \text{and} \; (s, \; h_2) \models_f \; \phi_2\]
universally quantifies over an infinite set!

\((s, h) \models_f \phi_1 \leadsto \phi_2\)

\(\overset{\text{def}}{\iff}\)

for all \(h'\),

if \(h \perp h'\) and \((s, h') \models_f \phi_1\),

then \((s, h \uplus h') \models_f \phi_2\)
Satisfiability problem

- \((s, h) \models_f \exists u \phi \overset{\text{def}}{\iff} \text{there is } l \in \mathbb{N} \text{ such that } (s, h) \models_{f[u \mapsto l]} \phi\)
  
  where \(f[u \mapsto l]\) is the assignment equal to \(f\) except that \(u\) takes the value \(l\).
Satisfiability problem

- \((s, h) \models_f \exists u \phi \overset{\text{def}}{\iff} \text{there is } l \in \mathbb{N} \text{ such that } (s, h) \models_{f[u \rightarrow l]} \phi\)
  where \(f[u \rightarrow l]\) is the assignment equal to \(f\) except that \(u\) takes the value \(l\).

- Satisfiability problem:
  
  **input:** formula \(\phi\) in 1SL
  
  **question:** are there \((s, h)\) and \(f\) such that \((s, h) \models_f \phi\)?

- Validity problem, entailment problem, ...
Standard inference rules for mutation

- \((s, h) \models f \ x \mapsto u\) iff \(\text{dom}(h) = \{s(x)\}\) and \(h(s(x)) = f(u)\).

- Local form (MUL)

\[
\{\exists u \ (x \mapsto u)\} \ [x] := y \ \{x \mapsto y\}
\]

- Global form (MUG)

\[
\{(\exists u \ (x \mapsto u)) * \phi\} \ [x] := y \ \{x \mapsto y * \phi\}
\]

- Backward-reasoning form (MUBR)

\[
\{(\exists u \ (x \mapsto u)) * ((x \mapsto y) * \phi)\} \ [x] := y \ \{\phi\}
\]
Taming the magic wand semantics

- Controversy about the use of magic wand for verification. See recent use in [Thakur & Breck & Reps, SPIN’14]

- Program variable $x$ is allocated:

  $$ (x \leftrightarrow x) \star \perp $$

- Equality between expressions $e$ and $e'$ ($u$ not in $e, e'$):

  $$ \forall u \left( u \leftrightarrow e \star u \leftrightarrow e' \right) $$
Simple properties stated in 1SL

• The value of $u$ is in the domain of the heap:
\[
\text{alloc}(u) \overset{\text{def}}{=} \exists u \ u \leftrightarrow u
\]  
(variant of $(u \leftrightarrow u) * \perp$)

• The heap has a unique cell $u_1 \leftrightarrow u_2$:
\[
u_1 \leftrightarrow u_2 \overset{\text{def}}{=} u_1 \leftrightarrow u_2 \land \neg \exists u' (u' \neq u_1 \land \text{alloc}(u'))
\]

• The domain of the heap is empty: \(\text{emp} \overset{\text{def}}{=} \neg \exists u \ \text{alloc}(u)\)

• $u$ has at least $k$ predecessors (2 options):
\[
\exists u_1, \ldots, u_k \bigwedge_{i \neq j} u_i \neq u_j \land \bigwedge_{i=1}^{k} u_i \leftrightarrow u
\]

\(k\) times
\[
(\exists u (u \leftrightarrow u)) \ast \cdots \ast (\exists u (u \leftrightarrow u))
\]
Expressive power / Decidability / Complexity

1SL ≡ DSOL ≡ WSOL ≡ 1SL(¬∗), undec.

1SL1, PSPACE-C

1SL2, undec.

1SL2(¬∗) ≡ DSOL, undec.

1SL2(∗), non-elem.

1SL0, PSPACE-C

1SL0, PSPACE-C

[Calcagno & Yang & O’Hearn, APLAS’01] 1SL0

[Brochenin & Demri & Lozes, IC 12] 1SL(¬∗)

[Demri & Galmiche & Larchey-Wendling & Mery, CSR’14] 1SL1

[Demri & Deters, CSL-LICS’14] 1SL2(¬∗)
A smooth extension: 2SL

- Heap $h : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ with finite domain.

- When $h_1 \perp h_2$, disjoint heap $h_1 \uplus h_2$.

- Atomic formulae: $\pi ::= e = e' \mid e \leftrightarrow e', e'' \mid \text{emp} \mid \bot$

- $(s, h) \models e \leftrightarrow e', e'' \iff \llbracket e \rrbracket \in \text{dom}(h)$ and $h(\llbracket e \rrbracket) = (\llbracket e' \rrbracket, \llbracket e'' \rrbracket)$

- 2SL satisfiability problem is undecidable by reduction from finitary satisfiability for classical predicate logic.
  
  [Trakhtenbrot, 50; Calcagno & Yang & O’Hearn, APLAS 01]

- Validity in 2SL is not recursively enumerable.
Expressive Power of 1SL
Weak second-order logic WSOL

- Formulae:

\[
\phi ::= u_i = u_j \mid u_i \leftrightarrow u_j \mid \phi \land \phi \mid \neg \phi \mid \\
\exists u_i \phi \mid \exists P \phi \mid P(u_1, \ldots, u_n)
\]

- \( h \models_{\mathcal{W}} \exists P \phi \) iff there is a finite \( R \subseteq \mathbb{N}^n \) such that 
  \( h \models_{\mathcal{W}[P \rightarrow R]} \phi \).

- \( h \models_{\mathcal{W}} P(u_1, \ldots, u_n) \) iff \( (\mathcal{W}(u_1), \ldots, \mathcal{W}(u_n)) \in \mathcal{W}(P) \).

- DSOL: Dyadic fragment of WSOL.

- Known reduction from WSOL to DSOL.
From 1SL to DSOL
(internalization of 1SL semantics)

$$\text{hp}(P) \overset{\text{def}}{=} \forall u, u', u'' \ (P(u, u') \land P(u, u'')) \Rightarrow u' = u''$$

$$P = Q*R \overset{\text{def}}{=} \forall u, u' \ (P(u, u') \Leftrightarrow (Q(u, u') \lor R(u, u')) \land \neg (Q(u, u') \land R(u, u'))$$

- Translation $$\exists P \ (\forall u, u' \ P(u, u') \Leftrightarrow u \rightarrow u') \land t_P(\phi):$$

  $$t_P(u \rightarrow u') \overset{\text{def}}{=} P(u, u')$$

  $$t_P(\psi * \varphi) \overset{\text{def}}{=} \exists Q, Q' \ P = Q * Q' \land t_Q(\psi) \land t_{Q'}(\varphi)$$

  $$t_P(\psi \rightarrow \varphi) \overset{\text{def}}{=} \forall Q \ ((\exists Q' \ \text{hp}(Q') \land Q' = Q * P) \land \text{hp}(Q) \land t_Q(\psi)) \Rightarrow (\exists Q' \ \text{hp}(Q') \land Q' = Q * P \land t_{Q'}(\varphi))$$

Expressive Power of 1SL
Principles to reduce DSOL into 1SL2(−*)

- Valuation heap encodes first-order and second-order valuations.

- Pair \((l, l')\) belongs to \(P_i\) whenever \(l\) and \(l'\) can be identified thanks to some special patterns with arithmetical constraints on the number of predecessors.

- To be able to distinguish the original heap from the valuation heap.

- To be able to have distinct patterns for different variables.
\{2, 5, 7, 9\}-well-formed heap
A translation $T$

- For every sentence $\phi$ in DSOL, for every heap $h$, we have $h \models \phi$ iff $h \models T(\phi)$.

- WSOL and 1SL2(→∗) have the same expressive power.  
  [Demri & Deters, CSL-LICS’14]

- Satisfiability problem for 1SL2(→∗) is undecidable.

- Validity in 1SL is not recursively enumerable.
  . . . but 1SL(∗) is decidable by translation into weak monadic 2nd order theory of $(D, f, =)$.
  [Rabin, Trans. of AMS 69]
A translation \( T \)

- For every sentence \( \phi \) in DSOL, for every heap \( h \), we have \( h \models \phi \) iff \( h \models T(\phi) \).

- WSOL and 1SL2(\( \rightarrow \ast \)) have the same expressive power.
  
  [Demri & Deters, CSL-LICS’14]

- Satisfiability problem for 1SL2(\( \rightarrow \ast \)) is undecidable.

- Validity in 1SL is not recursively enumerable.

- \( \ldots \) but 1SL(\( \ast \)) is decidable by translation into weak monadic 2nd order theory of \((D, f, =)\).

  [Rabin, Trans. of AMS 69]
Playing with 1SL Restricted to a Single Variable

(1SL0: PSPACE-complete — 1SL2: undecidable)
Simple properties stated in 1SL1

- Unique cell $x_1 \mapsto x_2$:

  \[ x_1 \mapsto x_2 \overset{\text{def}}{=} x_1 \mapsto x_2 \land \neg\exists u \left( u \neq x_1 \land ((u \leftrightarrow u) \not\rightarrow \bot) \right) \]

- The variable $x_i$ points to a location that is a loop:

  \[ \text{toloop}(x_i) \overset{\text{def}}{=} \exists u \left( x_i \leftrightarrow u \land u \leftrightarrow u \right). \]

- The variable $x_i$ points to a location that is allocated:

  \[ \text{toalloc}(x_i) \overset{\text{def}}{=} \exists u \left( x_i \leftrightarrow u \land \text{alloc}(u) \right). \]
More properties

- Variables $x_i$ and $x_j$ point to a shared location:
  \[ \text{conv}(x_i, x_j) \overset{\text{def}}{=} \exists u (x_i \rightarrow u \land x_j \rightarrow u). \]

- There is a location between $x_i$ and $x_j$:
  \[ \text{btwn}(x_i, x_j) \overset{\text{def}}{=} \exists u (x_i \rightarrow u \land u \rightarrow x_j). \]
More properties

• Variables $x_i$ and $x_j$ point to a shared location:

$$\text{conv}(x_i, x_j) \overset{\text{def}}{=} \exists u \ (x_i \xleftarrow{} u \land x_j \xrightarrow{} u).$$

![Diagram](image1)

$\textbf{x}_i$ $\textbf{x}_j$

• there is a location between $x_i$ and $x_j$:

$$\text{btwn}(x_i, x_j) \overset{\text{def}}{=} \exists u \ (x_i \xleftarrow{} u \land u \xrightarrow{} x_j).$$

![Diagram](image2)

$\textbf{x}_i$ $\textbf{x}_j$

What else?
Expressive completeness

- We characterize the expressive power of 1SL1:
  any 1SL1 formula is logically equivalent to a Boolean combination of atomic properties (yet to be defined).

- This extends results for 1SL0. [Lozes, SPACE’04]
• \( \text{pred}(s, h) \overset{\text{def}}{=} \bigcup_i \text{pred}(s, h, i) \) with
  \( \text{pred}(s, h, i) \overset{\text{def}}{=} \{ l' : h(l') = s(x_i) \} \) for every \( i \in [1, q] \).

• \( \text{loop}(s, h) \overset{\text{def}}{=} \{ l \in \text{dom}(h) : h(l) = l \} \).

• \( \text{rem}(s, h) \overset{\text{def}}{=} \text{dom}(h) \setminus (\text{pred}(s, h) \cup \text{loop}(s, h)) \).

• \( \text{dom}(h) = \text{rem}(s, h) \uplus (\text{pred}(s, h) \cup \text{loop}(s, h)) \).
Partition two: introducing the core

- \( \text{ref}(s, h) \overset{\text{def}}{=} \text{dom}(h) \cap s(\mathcal{V}); \text{acc}(s, h) \overset{\text{def}}{=} \text{dom}(h) \cap h(s(\mathcal{V})). \)

- \( \diamondsuit(s, h) \overset{\text{def}}{=} \text{ref}(s, h) \cup \text{acc}(s, h); \overline{\diamondsuit}(s, h) \overset{\text{def}}{=} \text{dom}(h) \setminus \diamondsuit(s, h). \)

Playing with 1SL Restricted to a Single Variable (1SL1)
Locations outside of the core

• Locations in the core are easy to identify thanks to program variables.

• $\text{pred}_\heartsuit(s, h, i) \overset{\text{def}}{=} \text{pred}(s, h, i) \setminus \heartsuit(s, h)$.

• $\text{loop}_\heartsuit(s, h) \overset{\text{def}}{=} \text{loop}(s, h) \setminus \heartsuit(s, h)$.

• $\text{rem}_\heartsuit(s, h) \overset{\text{def}}{=} \text{rem}(s, h) \setminus \heartsuit(s, h)$.

$$\text{dom}(h) = \heartsuit(s, h) \uplus \text{pred}_\heartsuit(s, h) \uplus \text{loop}_\heartsuit(s, h) \uplus \text{rem}_\heartsuit(s, h)$$
Test formulae

- **Equality** $\overset{\text{def}}{=} \{ x_i = x_j \mid i, j \in [1, q] \}$.

- **Pattern** $\overset{\text{def}}{=} \{ x_i \mapsto x_j, \text{conv}(x_i, x_j), \text{btwn}(x_i, x_j) \mid i, j \in [1, q] \} \cup \{ \text{toalloc}(x_i), \text{toloop}(x_i), \text{alloc}(x_i) \mid i \in [1, q] \}$.

- **Extra** $u \overset{\text{def}}{=} \{ u = u, u \mapsto u, \text{alloc}(u) \} \cup \{ x_i = u, x_i \mapsto u, u \mapsto x_i \mid i \in [1, q] \}$.
Test formulae

- **Equality** \(\overset{\text{def}}{=} \{ x_i = x_j \mid i, j \in [1, q] \} \).

- **Pattern** \(\overset{\text{def}}{=} \{ x_i \leftarrow x_j, \text{conv}(x_i, x_j), \text{btwn}(x_i, x_j) \mid i, j \in [1, q] \} \cup \{ \text{toalloc}(x_i), \text{toloop}(x_i), \text{alloc}(x_i) \mid i \in [1, q] \} \).

- **Extra** \(\overset{\text{def}}{=} \{ u = u, u \leftarrow u, \text{alloc}(u) \} \cup \{ x_i = u, x_i \leftarrow u, u \leftarrow x_i \mid i \in [1, q] \} \).

- **Size** \(\overset{\text{def}}{=} \{ \# \text{pred} \overset{i}{\downarrow} \geq k \mid i \in [1, q], k \in [1, \alpha] \} \cup \{ \# \text{loop} \overset{\downarrow}{\downarrow} \geq k, \# \text{rem} \overset{\downarrow}{\downarrow} \geq k \mid k \in [1, \alpha] \} \).

- **Test** \(\overset{\text{def}}{=} \text{Equality} \cup \text{Pattern} \cup \text{Size}_\alpha \cup \text{Extra}^u \cup \{ \bot \} \).
Deciding satisfiability for test formulae

- Satisfiability of conjunctions of test formulae or their negation can be checked in polynomial time.
- Satisfiability problem for Boolean combinations of test formulae is $\mathbf{NP}$-complete.
Deciding satisfiability for test formulae

- Satisfiability of conjunctions of test formulae or their negation can be checked in polynomial time.
- Satisfiability problem for Boolean combinations of test formulae is $\text{NP}$-complete.
- Polynomial-time decision procedure based on a saturation algorithm.

\[
\text{conv}(x_i, x_j) \quad \text{toloop}(x_i) \quad \frac{\text{toloop}(x_i)}{\text{toloop}(x_j)}
\]
Expressiveness/Decision Procedure for 1SL1
Quantifier elimination

- Any $\phi$ in 1SL1 (with $q$ program variables) is equivalent to a Boolean combination $\phi'$ of test formulae in $\text{Test}^u_{q \times |\phi|}$. [CSR’14]

- Any satisfiable $\phi$ in 1SL1 has a polynomial-size model.

- 1SL2 is strictly more expressive than 1SL1.

- 1SL1 cannot distinguish the two models below, 1SL2 can:

$$x_1 \rightarrow \bullet \rightarrow \bullet \rightarrow x_2 \quad | \quad x_1 \rightarrow \bullet \rightarrow \bullet \quad \odot \rightarrow x_2$$

Expressiveness/Decision Procedure for 1SL1
Abstract memory states $\approx$ atoms

Abstract memory state: $a = ((V, E), l, r, p_1, \ldots, p_q)$.

$V_{par} \subseteq V$ partition of $\{x_1, \ldots, x_q\}$.

Expressiveness/Decision Procedure for 1SL1
Abstract memory states ≈ atoms

Abstract memory state: \((a = (V, E), l, r, p_1, \ldots, p_q))\).

\(V_{\text{par}} \subseteq V\) partition of \(\{x_1, \ldots, x_q\}\).
Abstract memory states $\approx$ atoms

$l = 2, r = 2, p_1 = 1, p_2 = p_3 = p_4 = 0.$

Abstract memory state: $a = ((V, E), l, r, p_1, \ldots, p_q).

V_{par} \subseteq V$ partition of $\{x_1, \ldots, x_q\}.$
Abstraction

• $\text{abs}(s, h)$: abstract memory states abstracting the memory state $(s, h)$.

• Isomorphic $\alpha$ and $\alpha'$: identical partition, isomorphic graphs, identical numerical values.

• Given $q, \alpha \geq 1$, the number of abstract memory states is exponential in $q + \alpha$ and each abstract memory state can be encoded in space $q + \log(\alpha)$. 
Pointed abstract memory states

- Need for abstracting the interpretation of variable $u$.

- Pointed abstract memory states $(a, u)$ with $u$ in the set

$$V^{d \leq 1} \cup \{ \overline{D}, P(1), \ldots, P(q), L, R \}$$

(conditions apply)

- $\text{abs}(s, h, l)$ defined accordingly.

- $(a, u)$ and $(a', u')$ are isomorphic $\iff a$ and $a'$ are isomorphic and, $u = u'$ or $u$ and $u'$ are related by the isomorphism.
Abstract separation

\[ *_a((a, u), (a_1, u_1), (a_2, u_2)) \overset{\text{def}}{\iff} \]

there exist \( l \in \mathbb{N} \), a store \( s \) and disjoint heaps \( h_1 \) and \( h_2 \) such that \( \text{abs}(s, h_1 \cup h_2, l) = (a, u) \), \( \text{abs}(s, h_1, l) = (a_1, u_1) \) and \( \text{abs}(s, h_2, l) = (a_2, u_2) \).

Given \( q, \alpha \geq 1 \), the ternary relation \( *_a \) can be decided in polynomial time in \( q + \log(\alpha) \) for all the pointed abstract memory states over \((q, \alpha)\).
Polynomial-space model-checking algorithm

1: if $\psi$ is atomic then return AMC(((a, u), $\psi$));
2: if $\psi = \neg \psi_1$ then return not MC(((a, u), $\psi_1$));
3: if $\psi = \psi_1 \land \psi_2$ then return (MC(((a, u), $\psi_1$)) and MC(((a, u), $\psi_2$)));
4: if $\psi = \exists u \psi_1$ then return $\top$ iff there is u' such that MC(((a, u'), $\psi_1$)) = $\top$;
5: if $\psi = \psi_1 * \psi_2$ then return $\top$ iff there are (a₁, u₁) and (a₂, u₂) such that $\ast_a(((a, u), (a_1, u_1), (a_2, u_2))$ and MC(((a₁, u₁), $\psi_1$)) = MC(((a₂, u₂), $\psi_2$)) = $\top$;
6: if $\psi = \psi_1 \ast \psi_2$ then return $\bot$ iff for some (a', u') and (a'', u'') such that $\ast_a(((a'', u''), (a', u'), (a, u)))$, MC(((a', u'), $\psi_1$)) = $\top$ and MC(((a'', u''), $\psi_2$)) = $\bot$;
A Taste of Other Decision Procedures
A selection of techniques

- Tableaux calculus for 2SL0 [Galmiche & Mery, JLC 10].
  (satisfiability for 2SL0 is $\text{PSPACE}$-complete only)

- Graph-based (semantical) methods.
  [Cook et al., CONCUR’11; Enea & Saveluc & Sighireanu, ESOP’13]

- Translating into FO or propositional calculus.
  [Lozes, SPACE’04; Calcagno & Gardner & Hague, FOSSACS’05]
A robust framework: using SMT solvers

- Convergence of SAT and theory reasoning: Satisfiability Modulo Theories (SMT).

- Mature and robust tools: CVC4, MathSAT, Yices, Z3, etc.

- SMT solvers at the heart of verification tools.

- Pointer logic is one of the new theories being developed.

- Recent works using SMT solvers for separation logic.
  - E.g., translating SLLB into the logic of graph reachability and stratified sets (GRASS).

  [Piskac & Wies & Zufferey, CAV’13]

See also [Navarro Pérez & A. Rybalchenko, APLAS’13]
Concluding remarks

- 1SL2 and 1SL1 have exact expressive power characterization.

- At the heart of verification methods for programs with mutable data structures.

- For the first time, SMT-COMP 2014 run a competition with SMT solvers for separation logic as an “off” event.

- Many pending questions, e.g.:
  1. How to design decidable fragments with the magic wand?
  2. Design tractable fragments useful for formal verification.
  3. Proof methods for separation logic.
  4. Alternative memory models.