Separation Logic with One Quantified Variable

D. Larchey-Wendling[2]

Joint work with S. Demri[1], D. Galmiche[2] and D. Méry[2]

[1] NYU – CNRS and [2] LORIA – CNRS – UL

CSR, June 2014

Overview

1. Separation Logic in a Nutshell
2. Separation Logic 1SL1
3. Expressive Completeness
4. Some remarks on MC and SAT

Separation logic

- Introduced by Ishtiaq, Reynolds, O'Hearn, Pym.
- Extension of Hoare Logic by J.C. Reynolds with separating connectives.
- Reasoning about the heap with a strong form of locality built-in.
- $\phi * \psi$ is true whenever the heap can be divided into two disjoint parts, one satisfies ϕ, the other one ψ.
- $\phi \mathrel{-\!\!*} \psi$ is true whenever ϕ is true for a (fresh) disjoint heap, ψ is true for the combined heap.

Hoare triples

- Hoare triple: $\{\phi\}\ \texttt{PROG}\ \{\psi\}$ (total correctness).

- Rule of constancy:

$$\frac{\{\phi\}\ \texttt{PROG}\ \{\psi\}}{\{\phi \wedge \psi'\}\ \texttt{PROG}\ \{\psi \wedge \psi'\}}$$

  where no variable free in ψ' is modified by $\texttt{PROG}$.

- Unsoundness of the rule of constancy with pointers:

$$\frac{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z})\}\ [\mathtt{x}] := 4\ \{\mathtt{x} \mapsto 4\}}{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z}) \wedge \mathtt{y} \mapsto 3\}\ [\mathtt{x}] := 4\ \{\mathtt{x} \mapsto 4 \wedge \mathtt{y} \mapsto 3\}}$$

  (when $\mathtt{x} = \mathtt{y}$)
  $\mathtt{x} \mapsto \mathtt{z}$: “memory has a unique memory cell $\mathtt{x} \mapsto \mathtt{z}$”

When separation logic enters into the play

- Reparation with frame rule:

$$\frac{\{\phi\}\ \mathtt{PROG}\ \{\psi\}}{\{\phi * \psi'\}\ \mathtt{PROG}\ \{\psi * \psi'\}}$$

  where no variable free in ψ' is modified by $\mathtt{PROG}$.

- Strengthening precedent (SP)

$$\frac{\phi \Rightarrow \psi' \quad \{\psi'\}\ \mathtt{PROG}\ \{\psi\}}{\{\phi\}\ \mathtt{PROG}\ \{\psi\}}$$

- Checking entailment/validity/satisfiability in separation logic is a building block of the verification process.

Memory states for *n*SL (*n* record fields)

- Program variables $\mathtt{PVAR} = \{\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3, \ldots\}$.
- Memory state:
  - Store $\mathfrak{s} : \mathtt{PVAR} \rightarrow \mathtt{Val}$.
  - Heap $\mathfrak{h} : \mathtt{Loc} \rightharpoonup \mathtt{Val}^n$ with finite domain.

  $(\mathtt{Loc} = \{\mathfrak{l}, \mathfrak{l}', \ldots\}, \mathtt{Val} = \mathbb{N} \uplus \mathtt{Loc} \uplus \{\mathit{nil}\})$
- Simplification: $\mathtt{Loc} = \mathtt{Val} = \mathbb{N}$ (like low level memory).
- Disjoint heaps: $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \emptyset$ (noted $\mathfrak{h}_1 \perp \mathfrak{h}_2$).
- When $\mathfrak{h}_1 \perp \mathfrak{h}_2$, $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \stackrel{\text{def}}{=} \mathfrak{h}_1 \uplus \mathfrak{h}_2$.

Syntax and semantics for *n*SL

- Quantified variables $\mathtt{FVAR} = \{\mathtt{u}_1, \mathtt{u}_2, \mathtt{u}_3, \ldots\}$.

- Expressions: $e ::= \mathtt{x}_i \mid \mathtt{u}_j$

- Atomic formulae: $\pi ::= e = e' \mid e \hookrightarrow e_1, \ldots, e_n \mid \mathtt{emp}$

- Formulae in *n*SL

$$\phi ::= \bot \mid \pi \mid \phi \wedge \psi \mid \neg\phi \mid \phi * \psi \mid \phi \mathrel{-\!\!*} \psi \mid \exists\, \mathtt{u}_j\ \phi$$

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \mathtt{emp} \overset{\text{def}}{\Leftrightarrow} \mathrm{dom}(\mathfrak{h}) = \emptyset$.

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} e = e' \overset{\text{def}}{\Leftrightarrow} [e] = [e']$, with $[\mathtt{x}_i] \overset{\text{def}}{=} \mathfrak{s}(\mathtt{x}_i)$ and $[\mathtt{u}_j] \overset{\text{def}}{=} \mathfrak{f}(\mathtt{u}_j)$.

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} e \hookrightarrow e_1, \ldots, e_n \overset{\text{def}}{\Leftrightarrow} \mathfrak{h}([e]) = ([e_1], \ldots, [e_n])$.

Semantics for *n*SL

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi_1 * \phi_2 \overset{\text{def}}{\Leftrightarrow} \mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models_{\mathfrak{f}} \phi_1$, $(\mathfrak{s}, \mathfrak{h}_2) \models_{\mathfrak{f}} \phi_2$ for some $\mathfrak{h}_1, \mathfrak{h}_2$.

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi_1 \twoheadrightarrow \phi_2 \overset{\text{def}}{\Leftrightarrow}$ for all $\mathfrak{h}'$, if $\mathfrak{h} \perp \mathfrak{h}'$ and $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{f}} \phi_1$ then $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models_{\mathfrak{f}} \phi_2$.

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \exists\, \mathtt{u}_j\, \phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l} \in \mathbb{N}$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}'} \phi$ where $\mathfrak{f}' = \mathfrak{f}[\mathtt{u}_j \mapsto \mathfrak{l}]$ is the assignment equal to $\mathfrak{f}$ except that $\mathtt{u}_j$ takes the value $\mathfrak{l}$.

- Satisfiability problem:

  input: formula ϕ in *n*SL

  question: are there $(\mathfrak{s}, \mathfrak{h})$ and $\mathfrak{f}$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi$?

Satisfiability in fragments of *n*SL

- *n*SL: *n* record fields, unrestricted quantification
- *n*SL*i*: *n* record fields, at most *i* quantified variables
- *n*SL0 decidable and PSPACE-complete [Calcagno et al., 01]
- *n*SL undecidable for $n \geqslant 2$, by encoding finitary SAT of classical logic with a single binary relation [Calcagno et al., 01]
- 1SL and 1SL($-\!\!*$) undecidable [Brochenin, Demri & Lozes 08] by reduction to WSOL
- 1SL2 undecidable [Demri & Deters, submitted] by reduction to Minsky machines
- Our focus is on 1SL1: decidabilty and complexity

Summary of our contributions on 1SL1

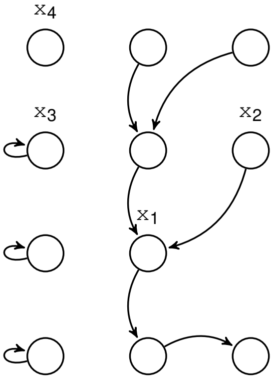
- 1SL1 = one record, one quantified var., *q* program vars.
- decomposition of heaps: core, loops, predecessors...
- given a bound α, a finite set of test formulae $\texttt{Test}_\alpha$
  - test the structure of the core + cardinality constraints
  - SAT of Boolean comb. of $\texttt{Test}_\alpha$ is NP-complete
- if two heaps cannot be distinguished by $\texttt{Test}_\alpha$, they cannot be distinguished by any ϕ s.t. $\texttt{th}(q, \phi) \leqslant \alpha$
- ϕ (with $\texttt{th}(q, \phi) \leqslant \alpha$) equiv. to Bool. comb. of $\texttt{Test}_\alpha$
- model check w.r.t. equiv. classes of heaps (w.r.t. $\texttt{Test}_\alpha$)
- give an abstract representation for these classes
- PSPACE algorithm for abstract MC and SAT

Separation Logic 1SL1

Memory states (one field)

- Memory state $(\mathfrak{s}, \mathfrak{h})$:
  - Store $\mathfrak{s} : \mathtt{PVAR} \rightarrow \mathbb{N}$.

  - Heap $\mathfrak{h} : \mathbb{N} \rightharpoonup \mathbb{N}$ with finite domain.
    Graph of a unary function with finite domain.



Specialization for 1SL1 (one field, one quantified variable)

- Expressions: $e ::= \mathtt{x}_i \mid \boxed{\mathtt{u}}$
- Atomic formulae: $\pi ::= e = e' \mid \boxed{e \hookrightarrow e'} \mid \mathtt{emp}$
- Formulae in 1SL1

$$\phi ::= \bot \mid \pi \mid \phi \wedge \psi \mid \neg \phi \mid \phi * \psi \mid \phi \mathrel{-\!\!*} \psi \mid \boxed{\exists\ \mathtt{u}\ \phi}$$

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \mathtt{emp} \overset{\text{def}}{\Leftrightarrow} \mathrm{dom}(\mathfrak{h}) = \emptyset$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} e = e' \overset{\text{def}}{\Leftrightarrow} [e] = [e']$, with $[\mathtt{x}_i] \overset{\text{def}}{=} \mathfrak{s}(\mathtt{x}_i)$ and $[\mathtt{u}] \overset{\text{def}}{=} \mathfrak{l}$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} e \hookrightarrow e' \overset{\text{def}}{\Leftrightarrow} [e] \in \mathrm{dom}(\mathfrak{h})$ and $\mathfrak{h}([e]) = [e']$.

Semantics for 1SL1

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi_1 * \phi_2 \overset{\text{def}}{\Leftrightarrow} \mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models_{\mathfrak{l}} \phi_1$, $(\mathfrak{s}, \mathfrak{h}_2) \models_{\mathfrak{l}} \phi_2$ for some $\mathfrak{h}_1, \mathfrak{h}_2$.

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi_1 \mathrel{-\!\!*} \phi_2 \overset{\text{def}}{\Leftrightarrow}$ for all $\mathfrak{h}'$, if $\mathfrak{h} \perp \mathfrak{h}'$ and $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{l}} \phi_1$ then $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models_{\mathfrak{l}} \phi_2$.

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \exists \mathtt{u} \ \phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}' \in \mathbb{N}$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}'} \phi$.

- Satisfiability problem:

  input: formula ϕ in 1SL1

  question: are there $(\mathfrak{s}, \mathfrak{h})$ and $\mathfrak{l}$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi$?

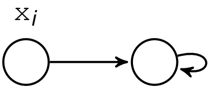
- Between 1SL0 (PSPACE) and 1SL2 (undecidable)

Simple properties stated in 1SL1

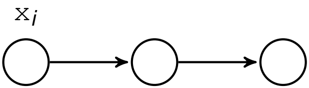
- The domain of the heap has at least *k* elements:
  $\neg \mathtt{emp} * \cdots * \neg \mathtt{emp}$ (*k* times).

- The variable $\mathtt{x}_i$ is allocated in the heap:
  $\mathtt{alloc}(\mathtt{x_i}) \stackrel{\text{def}}{=} (\mathtt{x_i} \hookrightarrow \mathtt{x_i}) \mathrel{-\!\!*} \bot$.

- The variable $\mathtt{x}_i$ points to a location that is a loop:
  $\mathtt{toloop}(\mathtt{x_i}) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ (\mathtt{x}_i \hookrightarrow \mathtt{u} \wedge \mathtt{u} \hookrightarrow \mathtt{u})$.

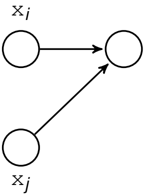


- The variable $\mathtt{x}_i$ points to a location that is allocated:
  $\mathtt{toalloc}(\mathtt{x_i}) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ (\mathtt{x_i} \hookrightarrow \mathtt{u} \wedge \mathtt{alloc}(\mathtt{u}))$.

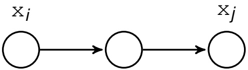


More properties

- Variables $\mathtt{x}_i$ and $\mathtt{x}_j$ point to a shared location:
  $\mathtt{conv}(\mathtt{x}_i, \mathtt{x}_j) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ (\mathtt{x}_\mathtt{i} \hookrightarrow \mathtt{u} \wedge \mathtt{x}_\mathtt{j} \hookrightarrow \mathtt{u})$.



- there is a location between $\mathtt{x}_i$ and $\mathtt{x}_j$:
  $\mathtt{inbetween}(\mathtt{x}_i, \mathtt{x}_j) \stackrel{\text{def}}{=} \exists \mathtt{u}\ (\mathtt{x}_i \hookrightarrow \mathtt{u} \wedge \mathtt{u} \hookrightarrow \mathtt{x}_j)$.



What Else?

Partition one: loops, predecessors, etc.

- $\mathrm{pred}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \bigcup_i \mathrm{pred}(\mathfrak{s}, \mathfrak{h}, i)$ with
  $\mathrm{pred}(\mathfrak{s}, \mathfrak{h}, i) \stackrel{\text{def}}{=} \{\mathfrak{l}' : \mathfrak{h}(\mathfrak{l}') = \mathfrak{s}(\mathrm{x}_i)\}$ for every $i \in [1, q]$.

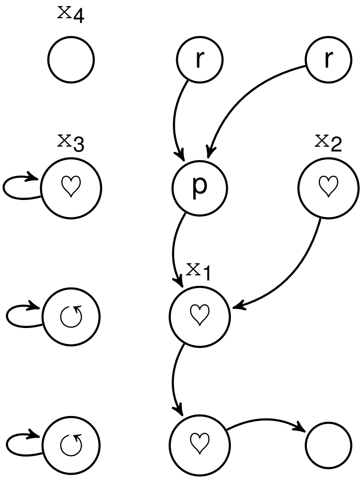
- $\mathrm{loop}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \{\mathfrak{l} \in \mathrm{dom}(\mathfrak{h}) : \mathfrak{h}(\mathfrak{l}) = \mathfrak{l}\}$.

- $\mathrm{rem}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{dom}(\mathfrak{h}) \setminus (\mathrm{pred}(\mathfrak{s}, \mathfrak{h}) \cup \mathrm{loop}(\mathfrak{s}, \mathfrak{h}))$.

- $\boxed{\mathrm{dom}(\mathfrak{h}) = \mathrm{rem}(\mathfrak{s}, \mathfrak{h}) \uplus (\mathrm{pred}(\mathfrak{s}, \mathfrak{h}) \cup \mathrm{loop}(\mathfrak{s}, \mathfrak{h})).}$

Partition two: introducing the core

- $\mathrm{ref}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{dom}(\mathfrak{h}) \cap \mathfrak{s}(\mathcal{V})$; $\mathrm{acc}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{dom}(\mathfrak{h}) \cap \mathfrak{h}(\mathfrak{s}(\mathcal{V}))$.
- $\heartsuit(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{ref}(\mathfrak{s}, \mathfrak{h}) \cup \mathrm{acc}(\mathfrak{s}, \mathfrak{h})$; $\overline{\heartsuit}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{dom}(\mathfrak{h}) \setminus \heartsuit(\mathfrak{s}, \mathfrak{h})$.



Locations outside of the core

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

- $\mathrm{pred}_{\overline{\heartsuit}}(\mathfrak{s}, \mathfrak{h}, i) \stackrel{\text{def}}{=} \mathrm{pred}(\mathfrak{s}, \mathfrak{h}, i) \setminus \heartsuit(\mathfrak{s}, \mathfrak{h})$.

- $\mathrm{loop}_{\overline{\heartsuit}}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{loop}(\mathfrak{s}, \mathfrak{h}) \setminus \heartsuit(\mathfrak{s}, \mathfrak{h})$.

- $\mathrm{rem}_{\overline{\heartsuit}}(\mathfrak{s}, \mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{rem}(\mathfrak{s}, \mathfrak{h}) \setminus \heartsuit(\mathfrak{s}, \mathfrak{h})$.

- $\boxed{\mathrm{dom}(\mathfrak{h}) = \heartsuit(\mathfrak{s}, \mathfrak{h}) \uplus \mathrm{pred}_{\overline{\heartsuit}}(\mathfrak{s}, \mathfrak{h}) \uplus \mathrm{loop}_{\overline{\heartsuit}}(\mathfrak{s}, \mathfrak{h}) \uplus \mathrm{rem}_{\overline{\heartsuit}}(\mathfrak{s}, \mathfrak{h}).}$

Test formulae

- $\mathtt{Equality} \stackrel{\text{def}}{=} \{\mathtt{x}_i = \mathtt{x}_j \mid i, j \in [1, q]\}$.

- $\mathtt{Pattern} \stackrel{\text{def}}{=}$
  $\{\mathtt{x}_i \hookrightarrow \mathtt{x}_j, \mathtt{conv}(\mathtt{x}_\mathtt{i}, \mathtt{x}_\mathtt{j}), \mathtt{inbetween}(\mathtt{x}_i, \mathtt{x}_j) \mid i, j \in [1, q]\}$
  $\cup \{\mathtt{toalloc}(\mathtt{x}_i), \mathtt{toloop}(\mathtt{x}_i), \mathtt{alloc}(\mathtt{x}_\mathtt{i}) \mid i \in [1, q]\}$.

- $\mathtt{Extra}^{\mathtt{u}} \stackrel{\text{def}}{=}$
  $\{\mathtt{u} \hookrightarrow \mathtt{u}, \mathtt{alloc}(\mathtt{u})\} \cup \{\mathtt{x}_i = \mathtt{u}, \mathtt{x}_i \hookrightarrow \mathtt{u}, \mathtt{u} \hookrightarrow \mathtt{x}_i \mid i \in [1, q]\}$.

- $\mathtt{Size}_\alpha \stackrel{\text{def}}{=}$
  $\{\# \mathtt{pred}^{\mathtt{i}}_{\overline{\heartsuit}} \geqslant \mathtt{k} \mid i \in [1, q], k \in [1, \alpha]\}$
  $\cup \{\# \mathtt{loop}_{\overline{\heartsuit}} \geqslant \mathtt{k}, \# \mathtt{rem}_{\overline{\heartsuit}} \geqslant \mathtt{k} \mid k \in [1, \alpha]\}$.

- $\mathtt{Test}^{\mathtt{u}}_\alpha \stackrel{\text{def}}{=} \mathtt{Equality} \cup \mathtt{Pattern} \cup \mathtt{Size}_\alpha \cup \mathtt{Extra}^{\mathtt{u}} \cup \{\perp\}$.

Counting loops outside of the core

- Needed for expressing test formulae in 1SL1 !

- $\mathtt{T} \stackrel{\text{def}}{=} \{\mathtt{alloc}(\mathtt{x}_1), \ldots, \mathtt{alloc}(\mathtt{x}_q)\} \cup \{\mathtt{toalloc}(\mathtt{x}_1), \ldots, \mathtt{toalloc}(\mathtt{x}_q)\}$.

- $\mathfrak{f} : \mathtt{T} \rightarrow \{0, 1\}$.

$$\phi_{\mathfrak{f}} \stackrel{\text{def}}{=} \bigwedge\{\psi \mid \psi \in \mathtt{T} \text{ and } \mathfrak{f}(\psi) = 1\} \wedge \bigwedge\{\neg\psi \mid \psi \in \mathtt{T} \text{ and } \mathfrak{f}(\psi) = 0\}$$

- $\#\,\mathtt{loop}_{\overline{\heartsuit}} \geqslant \mathtt{k} \stackrel{\text{def}}{=} \bigvee_{\mathfrak{f}} \phi_{\mathfrak{f}} \wedge \left(\phi_{\mathfrak{f}}^{\mathrm{pos}} * (\#\,\mathtt{loop} \geqslant \mathtt{k})\right)$ with
  - $\phi_{\mathfrak{f}}^{\mathrm{pos}}$ = the positive part of $\phi_{\mathfrak{f}}$.
  - $\#\,\mathtt{loop} \geqslant \mathtt{k} \stackrel{\text{def}}{=} (\exists \mathtt{u}\ \mathtt{u} \hookrightarrow \mathtt{u}) * \cdots * (\exists \mathtt{u}\ \mathtt{u} \hookrightarrow \mathtt{u})$ (*k* times).

Deciding satisfiability for test formulae

- Satisfiability of conjunctions of $\mathtt{Test}^{\mathtt{u}}_{\alpha}/\neg\mathtt{Test}^{\mathtt{u}}_{\alpha}$ can be checked in polynomial time (with bounds in binary).
- Polynomial-time decision based on a saturation algorithm (see rules)

$$\frac{\phi \vdash \mathtt{x}_i \hookrightarrow \mathtt{x} \quad \phi \vdash \mathtt{x} \hookrightarrow \mathtt{y} \quad \phi \vdash \mathtt{x} = \mathtt{y}}{\phi \vdash \mathtt{toloop}(\mathtt{x}_\mathtt{i})}$$

$$\frac{\phi \vdash \mathtt{conv}(\mathtt{x}_i, \mathtt{x}_j) \quad \phi \vdash \mathtt{toloop}(\mathtt{x}_\mathtt{i})}{\phi \vdash \mathtt{toloop}(\mathtt{x}_\mathtt{j})}$$

$$\frac{\phi \vdash \neg\mathtt{alloc}(\mathtt{x}_i)}{\phi \vdash \neg\mathtt{toloop}(\mathtt{x}_\mathtt{i})}$$

- Satisfiability problem for Boolean combinations of test formulae in the set $\bigcup_{\alpha \geqslant 1} \mathtt{Test}^{\mathtt{u}}_{\alpha}$ is NP-complete.

Expressive Completeness

Memory threshold

- for any formula of 1SL1 with at most q program variables
- $\mathtt{th}(q, \phi) \stackrel{\text{def}}{=} 1$ for every atomic formula ϕ.
- $\mathtt{th}(q, \phi_1 \wedge \phi_2) \stackrel{\text{def}}{=} \mathtt{max}(\mathtt{th}(q, \phi_1), \mathtt{th}(q, \phi_2))$.
- $\mathtt{th}(q, \neg\phi_1) \stackrel{\text{def}}{=} \mathtt{th}(q, \phi_1)$ and $\mathtt{th}(q, \exists\, \mathtt{u}\ \phi_1) \stackrel{\text{def}}{=} \mathtt{th}(q, \phi_1)$.
- $\mathtt{th}(q, \phi_1 * \phi_2) \stackrel{\text{def}}{=} \mathtt{th}(q, \phi_1) + \mathtt{th}(q, \phi_2)$.
- $\mathtt{th}(q, \phi_1 \mathrel{-\!\!*} \phi_2) \stackrel{\text{def}}{=} q + \mathtt{max}(\mathtt{th}(q, \phi_1), \mathtt{th}(q, \phi_2))$.
- For all ϕ built over $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$, $1 \leqslant \mathtt{th}(q, \phi) \leqslant q \times |\phi|$.

α-equivalence, correctness of abstraction

- α-equivalence: indistinguishability with respect to test formula $\psi \in \mathtt{Test}^{\mathrm{u}}_{\alpha}$:

$$(\mathfrak{s}, \mathfrak{h}, \mathfrak{l}) \simeq_{\alpha} (\mathfrak{s}', \mathfrak{h}', \mathfrak{l}') \text{ whenever } (\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \psi \text{ iff } (\mathfrak{s}', \mathfrak{h}') \models_{\mathfrak{l}'} \psi$$

- Cardinality constraints are precise up to α.

if $(\mathfrak{s}, \mathfrak{h}, \mathfrak{l}) \simeq_{\alpha} (\mathfrak{s}', \mathfrak{h}', \mathfrak{l}')$

then

$(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi \text{ iff } (\mathfrak{s}', \mathfrak{h}') \models_{\mathfrak{l}'} \phi$

for any ϕ s.t. $\mathtt{th}(q, \phi) \leqslant \alpha$

- Hence formulae of threshold below α do not distinguish more memory states than those formulae in $\mathtt{Test}^{\mathrm{u}}_{\alpha}$

Quantifier elimination

- Any ϕ in 1SL1 (with q program variables) is equivalent to a Boolean combination ϕ' of test formulae in $\mathtt{Test}^{\mathtt{u}}_{\mathtt{th}(q,\phi)}$.
- $\alpha = \mathtt{th}(q, \phi)$.
- $\mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l}) \stackrel{\text{def}}{=} \left[\begin{array}{cc} & \{\psi \mid \psi \in \mathtt{Test}^{\mathtt{u}}_{\alpha} \text{ and } (\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \psi\} \\ \cup & \{\neg\psi \mid \psi \in \mathtt{Test}^{\mathtt{u}}_{\alpha} \text{ and } (\mathfrak{s}, \mathfrak{h}) \not\models_{\mathfrak{l}} \psi\} \end{array} \right]$
- Finiteness of $\mathtt{Test}^{\mathtt{u}}_{\alpha}$ entails $\mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$ is finite and $\bigwedge \mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$ is a well-defined atom.
- $(\mathfrak{s}', \mathfrak{h}') \models_{\mathfrak{l}'} \bigwedge \mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$ iff $(\mathfrak{s}, \mathfrak{h}, \mathfrak{l}) \simeq_{\alpha} (\mathfrak{s}', \mathfrak{h}', \mathfrak{l}')$.
  $\mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$ characterizes $(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$ up to α.
- $\bigwedge \mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$ spans a finite domain.
- $\phi' \stackrel{\text{def}}{=} \bigvee \{\bigwedge \mathcal{S}(\mathfrak{s}, \mathfrak{h}, \mathfrak{l}) \mid (\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi\}$ equivalent to ϕ.

non-constructive proof !

Corollaries

- Any satisfiable ϕ in 1SL1 has a polynomial-size model.

- 1SL2 is strictly more expressive than 1SL1.

- $\mathtt{Test}^{\mathtt{u}}_{\alpha}$ formulae cannot distinguish the two models below

$$\mathtt{x}_1 \rightarrow \bullet \rightarrow \bullet \rightarrow \mathtt{x}_2 \quad \Big| \quad \mathtt{x}_1 \rightarrow \bullet \rightarrow \bullet \qquad \circ \rightarrow \mathtt{x}_2$$

- hence neither can 1SL1.

- but 1SL2 can: $\exists \mathtt{u} \exists \mathtt{v}\ (\mathtt{x}_1 \hookrightarrow \mathtt{u} \wedge \mathtt{u} \hookrightarrow \mathtt{v} \wedge \mathtt{v} \hookrightarrow \mathtt{x}_2)$

Some remarks on MC and SAT

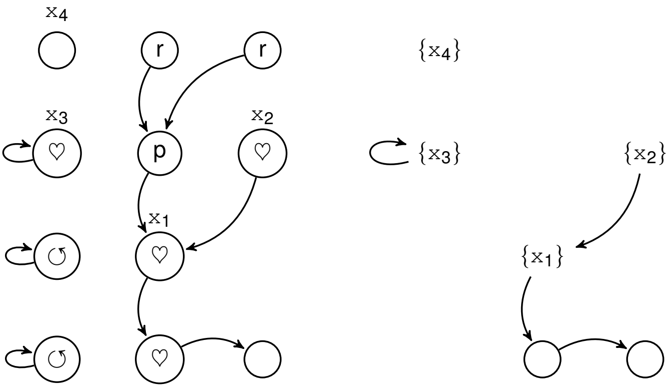
MC and SAT in 1SL1

- to check $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi_1 \twoheadrightarrow \phi_2$ we need to verify:

$$(\mathfrak{s}, \mathfrak{h}') \not\models_{\mathfrak{l}} \phi_1 \ \text{ or } \ (\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models_{\mathfrak{l}} \phi_2 \quad \boxed{\text{for any}}\ \mathfrak{h}' \perp \mathfrak{h}$$

- $(\mathfrak{s}, \emptyset) \models_{\mathfrak{l}} \neg(\top \twoheadrightarrow \neg\phi)$ iff there exists $\mathfrak{h}$ s.t. $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi$.
- $(\exists \mathfrak{h}, (\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \top * (\mathtt{emp} \wedge \phi))$ iff $(\mathfrak{s}, \emptyset) \models_{\mathfrak{l}} \phi$
- hence (MC) $\leftrightsquigarrow$ (SAT) in SL.
- for MC: transform the $\boxed{\text{for any}}$ into finite quantification
- indeed, given α, the test formula $\mathtt{Test}^{\mathtt{u}}_{\alpha}$
  - are finitely many, as well as their Boolean combinations
  - hence only finitely many classes for $(\mathfrak{s}, \mathfrak{h}, \mathfrak{l}) \simeq_{\alpha} (\mathfrak{s}', \mathfrak{h}', \mathfrak{l}')$
- any formula s.t. $\mathtt{th}(q, \phi) \leqslant \alpha$, the value of $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{l}} \phi$ only depends of the class of $(\mathfrak{s}, \mathfrak{h}, \mathfrak{l})$
- transform (infinite) ”for any” into (finite) ”for any class”

Abstract memory states $\approx$ atoms of $\mathtt{Test}^{\mathtt{u}}_{\alpha}$



$\mathfrak{l} = 2, \mathfrak{r} = 2, \mathfrak{p}_1 = 1, \mathfrak{p}_2 = \mathfrak{p}_3 = \mathfrak{p}_4 = 0.$

Abstract memory state: $\mathfrak{a} = ((V, E), \mathfrak{l}, \mathfrak{r}, \mathfrak{p}_1, \ldots, \mathfrak{p}_q)$.
$V_{\mathtt{par}} \subseteq V$ partition of $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$.

Abstract Model Checking in 1SL1

- we then prove that abstraction ”commutes” with MC
- we describe abstract composition/decomposition of heaps
- we present a MC algorithm on abstract memory states
- this MC algorithm runs in PSPACE
- PSPACE-hardness already holds for 1SL0
- hence MC in 1SL1 is PSPACE-complete
- the same complexity holds for SAT

Concluding remarks

- Quantifier elimination property for 1SL1 formulae.
- Conjunction of test formulae decidable in polynomial time.
- Satisfiability and model-checking problems for 1SL1 are PSPACE-complete.
- Also, restriction to q program variables in polynomial time.
- Possible extension with $k > 1$ record fields.