On the Almighty Wand

Stéphane Demri
LSV, ENS Cachan, CNRS, INRIA Saclay IdF

Joint work with Rémi Brochenin and Etienne Lozes

November 16th, 2009
Pointer programs

- Pointer: reference to a memory cell (non fixed memory address).

- Dynamic memory allocation/deallocation. (mutable data structures)

- Examples of instructions:
  - \( y \rightarrow l := x \): write \( x \) to the \( l \)-field of the cell pointed to by \( y \),
  - \texttt{free } \( x \): deallocate the cell pointer to by \( x \),
  - \( x := \texttt{malloc} (i) \): allocate \( i \) memory cells and assign its address to \( x \).

- Specific properties for pointer programs:
  - No \texttt{null} dereference.
  - Memory leak: a memory cell can no longer be reached.
  - Shape analysis: checking the structure of the heap.
Reasoning about pointer programs

- Examples of logical specification languages
  - Separation logic [Reynolds, LICS 02]
  - Pointer assertion logic (PAL) [Jensen et al. 97]
  - TVLA [Lev-Ami & Sagiv, SAS 00]: abstract interpretation technique with Kleene’s logic (op. semantics in FOL + TC)
  - Evolution Logic [Yahav et al., ESOP 03]: to specify temporal properties of programs with dynamically evolving heaps.
Reasoning about pointer programs

- Examples of logical specification languages
  - Separation logic [Reynolds, LICS 02]
  - Pointer assertion logic (PAL) [Jensen et al. 97]
  - TVLA [Lev-Ami & Sagiv, SAS 00]: abstract interpretation technique with Kleene’s logic (op. semantics in FOL + TC)
  - Evolution Logic [Yahav et al., ESOP 03]: to specify temporal properties of programs with dynamically evolving heaps.

- Model-checking
  - Navigation Temporal Logic [Distefano & Katoen & Rensink, FSTTCS 04]
  - Regular model-checking [Bouajjani et al., TACAS 05]
  - Translation into counter automata [Bouajjani et al, CAV 06; Sangnier, PhD 08]
Memory states (I)
Memory states (I)

- Set of variables $\text{Var}$.
- Set of selectors/labels $\text{Lab}$.
- Set of values $\text{val} = \mathbb{N} \cup \{\text{nil}\}$.
- Set of stores: $S \overset{\text{def}}{=} \text{Var} \rightarrow \text{Val}$.
- Set of heaps:
  $\mathcal{H} \overset{\text{def}}{=} \mathbb{N} \rightarrow_{\text{fin}} (\text{Lab} \rightarrow_{\text{fin}+} \text{Val})$.

Memory state $(s, h)$

In the sequel, we restrict ourselves to two selectors only or to one selector only.
Disjoint heaps

- $h_1$ and $h_2$ are disjoint whenever $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$. Notation: $h_1 \perp h_2$.
- Disjointness does not concern records.
- Disjoint union $h_1 \ast h_2$ whenever $h_1 \perp h_2$. 
Disjoint heaps

- $h_1$ and $h_2$ are disjoint whenever $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$. Notation: $h_1 \perp h_2$.
- Disjointness does not concern records.
- Disjoint union $h_1 \ast h_2$ whenever $h_1 \perp h_2$.
- Disjoint heap graphs (with a unique selector and $\text{val} = \mathbb{N}$):
Separation logic

- Introduced by Reynolds, Pym and O’Hearn.

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

- $\mathcal{A} \ast \mathcal{B}$ is true whenever the heap can be divided into two disjoint parts, one satisfies $\mathcal{A}$, the other one $\mathcal{B}$. (second-order existential modality)

- $\mathcal{A} \rightarrow \ast \mathcal{B}$ is true whenever $\mathcal{A}$ is true for a (fresh) disjoint heap, $\mathcal{B}$ is true for the combined heap. (second-order universal modality)
Hoare triples

- Hoare triple: $\{A\} \mathsf{PROG} \{B\}$ (total correctness).

- Rule of constancy:

  \[
  \begin{align*}
  \{A\} \mathsf{PROG} \{B\} & \quad \Rightarrow \\
  \{A \land B'\} \mathsf{PROG} \{B \land B'\}
  \end{align*}
  \]

  where no variable free in $B'$ is modified by $\mathsf{PROG}$.

- Unsoundness of the rule of constancy in separation logic:

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

  (when $x = y$)

  $x \mapsto z$: "memory has a unique memory cell $x \mapsto z$"
When separation logic enters into the play

- Reparation with frame rule:

\[
\begin{align*}
\{ \mathcal{A} \} & \text{ PROG } \{ \mathcal{B} \} \\
\{ \mathcal{A} \ast \mathcal{B}' \} & \text{ PROG } \{ \mathcal{B} \ast \mathcal{B}' \}
\end{align*}
\]

where no variable free in \( \mathcal{B}' \) is modified by \text{ PROG }.

- Strengthening precedent (SP)

\[
\begin{align*}
\mathcal{A} \Rightarrow \mathcal{B}' & \quad \{ \mathcal{B}' \} \text{ PROG } \{ \mathcal{B} \} \\
\{ \mathcal{A} \} & \text{ PROG } \{ \mathcal{B} \}
\end{align*}
\]

- Checking validity/satisfiability in separation logic is a building block of the verification process.
Standard inference rules for mutation

- Local form (MUL)

\[
\{(\exists z. x \mapsto z)\} \begin{array}{c} \end{array} x := \begin{array}{c} \end{array} y \{x \mapsto y\}
\]

- Global form (MUG)

\[
\{(\exists z. x \mapsto z) \ast A\} \begin{array}{c} \end{array} x := \begin{array}{c} \end{array} y \{x \mapsto y \ast A\}
\]

- Backward-reasoning form (MUBR)

\[
\{(\exists z. x \mapsto z) \ast ((x \mapsto y) \ast A)\} \begin{array}{c} \end{array} x := \begin{array}{c} \end{array} y \{A\}
\]
Memory states (II)

- Set of variables $\text{Var} = \{x, y, z, \ldots\}$.
- Set of locations $\text{Loc} = \{l, l', \ldots\}$.
- Set of values $\text{Val} = \mathbb{N} \cup \text{Loc} \cup \{\text{nil}\}$.
Memory states (II)

- Set of variables $\text{Var} = \{x, y, z, \ldots\}$.

- Set of locations $\text{Loc} = \{l, l', \ldots\}$.

- Set of values $\text{Val} = \mathbb{N} \cup \text{Loc} \cup \{\text{nil}\}$.

- Memory state:
  - Store $s : \text{Var} \rightarrow \text{Val}$.
  - Heap $h : \text{Loc} \nrightarrow \text{Val} \times \text{Val}$ with finite domain.

- Simplification: $\text{Loc} = \text{Val} = \mathbb{N}$. 
Memory states (II)

- Set of variables $\text{Var} = \{x, y, z, \ldots\}$.
- Set of locations $\text{Loc} = \{l, l', \ldots\}$.
- Set of values $\text{Val} = \mathbb{N} \cup \text{Loc} \cup \{\text{nil}\}$.

Memory state:
- Store $s: \text{Var} \rightarrow \text{Val}$.
- Heap $h: \text{Loc} \rightarrow \text{Val} \times \text{Val}$ with finite domain.

Simplification: $\text{Loc} = \text{Val} = \mathbb{N}$.

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

When $h_1 \perp h_2$, $h_1 \ast h_2 \overset{\text{def}}{=} h_1 \cup h_2$. 
Separation logic with two record fields

- Formulae:

\[ A := \neg A \mid A \land A \mid \exists x \ A \mid x \leftrightarrow y, z \mid x = y \mid A * A \mid A * A \]

- Satisfaction relation:

\( (s, h) \models \neg A \) iff not \( (s, h) \models A \)
\( (s, h) \models A \land B \) iff \( (s, h) \models A \) and \( (s, h) \models B \)
\( (s, h) \models \exists x \ A \) iff there is \( l \in \text{Loc} \) s.t. \( (s[x \leftrightarrow l], h) \models A \)
\( (s, h) \models x \leftrightarrow y, z \) iff \( h(s(x)) = (s(y), s(z)) \)
\( (s, h) \models x = y \) iff \( s(x) = s(y) \)
\( (s, h) \models A_1 * A_2 \) iff there are two heaps \( h_1, h_2 \) such that \( h = h_1 * h_2, (s, h_1) \models A_1 \) & \( (s, h_2) \models A_2 \),
\( (s, h) \models A_1 \rightarrow A_2 \) iff for all heaps \( h' \perp h \),
\( (s, h') \models A_1 \) then \( (s, h' * h) \models A_2 \).
Relationship between $\ast$ and $\neg\ast$

- $\neg\ast$ is the *adjunct* of $\ast$:

  $$(A \ast B) \Rightarrow C \text{ is valid iff } A \Rightarrow (B \ast C) \text{ is valid.}$$
Relationship between $\ast$ and $\ast^-$

- $\ast^-$ is the **adjunct** of $\ast$:
  
  $$(A \ast B) \Rightarrow C \text{ is valid iff } A \Rightarrow (B \ast^- C) \text{ is valid.}$$

- ...but the formula below is not valid
  
  $$((A \ast B) \Rightarrow C) \iff (A \Rightarrow (B \ast^- C))$$
Relationship between $\ast$ and $\ast$:

- $\ast$ is the *adjunct* of $\ast$:
  
  $$(A \ast B) \Rightarrow C \text{ is valid iff } A \Rightarrow (B \ast C) \text{ is valid.}$$

- ... but the formula below is not valid
  
  $$((A \ast B) \Rightarrow C) \iff (A \Rightarrow (B \ast C))$$

- *Septraction* $\ast$: existential version of $\ast$.

  $A \narrow B \overset{\text{def}}{=} \neg (A \ast \neg B)$

  $$(s, h) \models A \narrow B \text{ iff there is } h' \perp h \text{ such that } (s, h') \models A \text{ and } (s, h' \ast h) \models B.$$
Undecidability
[Calcagno & Yang & O’Hearn, APLAS 01]

- Reduction from finitary satisfiability for classical predicate logic restricted to a single binary predicate symbol, see e.g. [Trakhtenbrot, 50].

- \( D(x) \overset{\text{def}}{=} x \leftrightarrow \text{nil, nil.} \)

- Translation

  \[ \exists x, \text{nil} \ D(x) \land (\neg \exists y, z \ \text{nil} \leftrightarrow y, z) \land t(A) \]

  - \( t \) is homomorphically for Boolean connectives.
  - \( t(R(x, y)) = D(x) \land D(y) \land \exists z \ z \leftrightarrow x, y. \)
  - \( t(\exists x \ B) \overset{\text{def}}{=} \exists x \ D(x) \land t(B). \)
Undecidability
[Calcagno & Yang & O’Hearn, APLAS 01]

- Reduction from finitary satisfiability for classical predicate logic restricted to a single binary predicate symbol, see e.g. [Trakhtenbrot, 50].

- \( D(x) \overset{\text{def}}{=} x \leftrightarrow \text{nil, nil} \).

- Translation

  \[ \exists x, \text{nil} \ D(x) \land (\neg \exists y, z \ \text{nil} \leftrightarrow y, z) \land t(A) \]

  - \( t \) is homomorphous for Boolean connectives.

  - \( t(\mathcal{R}(x, y)) = D(x) \land D(y) \land \exists z \ z \leftrightarrow x, y \).

  - \( t(\exists x \ B) \overset{\text{def}}{=} \exists x \ D(x) \land t(B) \).

What is the decidability status with a unique selector?
Complexity of propositional fragments
[Calcagno & Yang & O’Hearn, APLAS 01]

- Model-checking and satisfiability for propositional separation logic is PSPACE-complete.

- See complexity of other fragments in [Reynolds, LICS 02].
Separation logic with one field
Memory states (one field)

- Memory state:
  - Store $s : \text{Var} \rightarrow \mathbb{N}$.
  - Heap $h : \mathbb{N} \rightarrow \mathbb{N}$ with finite domain.
    Graph of a unary function with finite domain.

At most one value in a location.

Values are only locations.
Memory states (one field)

- Memory state:
  - Store $s : \text{Var} \rightarrow \mathbb{N}$.
  - Heap $h : \mathbb{N} \rightarrow \mathbb{N}$ with finite domain.
    Graph of a unary function with finite domain.

At most one value in a location.

Values are only locations.

- Number of predecessors $\#l$: cardinal of $\{l' : h(l') = l\}$.
  $\#10 \geq 2$.  

![Graph of a unary function with finite domain.](image-url)
Syntax and semantics (bis)

\[ \mathcal{A} := \neg \mathcal{A} \mid \mathcal{A} \land \mathcal{A} \mid \exists x. \mathcal{A} \mid x \leftrightarrow y \mid x = y \mid \mathcal{A} \ast \mathcal{A} \mid \mathcal{A} \ast \ast \mathcal{A} \]

- Satisfaction relation:

\[ (s, h) \models \neg \mathcal{A} \iff \text{not } (s, h) \models \mathcal{A} \]
\[ (s, h) \models \mathcal{A} \land \mathcal{B} \iff (s, h) \models \mathcal{A} \text{ and } (s, h) \models \mathcal{B} \]
\[ (s, h) \models \exists x. \mathcal{A} \iff \text{there is } l \in \text{Loc} \text{ s.t. } (s[x \mapsto l], h) \models \mathcal{A} \]
\[ (s, h) \models x \leftrightarrow y \iff h(s(x)) = s(y) \]
\[ (s, h) \models x = y \iff s(x) = s(y) \]
\[ (s, h) \models \mathcal{A}_1 \ast \mathcal{A}_2 \iff \text{there are two heaps } h_1, h_2 \text{ such that} \]
\[ h = h_1 \ast h_2, (s, h_1) \models \mathcal{A}_1 \text{ and } (s, h_2) \models \mathcal{A}_2 \]
\[ (s, h) \models \mathcal{A}_1 \ast \ast \mathcal{A}_2 \iff \text{for all heaps } h', \downarrow h, \]
\[ \text{if } (s, h') \models \mathcal{A}_1 \text{ then } (s, h' \ast h) \models \mathcal{A}_2. \]
A selection of properties in $\mathbf{SL}$

- The value of $x$ is in the domain of the heap:
  \[ \text{alloc}(x) \stackrel{\text{def}}{=} \exists y \ x \leftrightarrow y. \]

- The heap has a unique cell $x \leftrightarrow y$:
  \[ x \leftrightarrow y \stackrel{\text{def}}{=} x \leftrightarrow y \land \neg \exists z \ z \neq x \land \text{alloc}(z) \]

- The domain of the heap is empty: $\text{emp} \stackrel{\text{def}}{=} \neg \exists x. \text{alloc}(x)$

- $x$ has at least $n$ predecessors (two options):
  \[ \exists x_1, \ldots, x_n. \ \bigwedge_{i \neq j} x_i \neq x_j \land \bigwedge_{i=1}^{n} x_i \leftrightarrow x \]
  \[ \underbrace{(\exists y. y \leftrightarrow x) \ast \cdots \ast (\exists y. y \leftrightarrow x)}_{n \text{ times}} \ast \top \]
Properties about lists in $\mathit{SL}(\ast)$

- The properties below can be expressed in $\mathit{SL}(\ast)$:
  - $(s, h)$ contains *only* a list between $x$ and $y$: $\lambda s(x, y)$.
  - There is a list between $x$ and $y$: $x \rightarrow^* y$.

- List properties and other recursive properties can be easily expressed in second-order logics.
Weak second-order logic $SO$
(or how to speak differently about memory states)

- Family $(\text{VAR}^i)_{i \geq 1}$ of second-order variables interpreted as finite relations.

- Environment $\mathcal{E}$: valuation for variables in $(\text{VAR}^i)_{i \geq 1}$.

- Satisfaction relation:
  $$(s, h), \mathcal{E} \models \exists P A \iff \text{there is a finite subset } R \text{ of } \text{Loc}^n,$$
  $$\text{such that } (s, h), \mathcal{E}[P \leftrightarrow R] \models A$$
  $$(s, h), \mathcal{E} \models P(x_1, \ldots, x_n) \iff (s(x_1), \ldots, s(x_n)) \in \mathcal{E}(P)$$

- Fragments: $\text{MSO}$ (only $\text{VAR}^1$) & $\text{DSO}$ (only $\text{VAR}^2$)

- $L \subseteq L'$ whenever for every $A \in L$, there is $A' \in L'$ that holds true in the same memory states.
SL ⊆ DSO (internalization of SL semantics)

- **Abbreviations:**
  - $heap(P) \overset{\text{def}}{=} \forall x, y, z. xPy \land xPz \Rightarrow y = z$,
  - $P = Q \ast R \overset{\text{def}}{=} \forall x, y. (xPy \iff (xQy \lor xRy)) \land \lnot (xQy \land xRy)$.

- **Translation** $\exists P. (\forall x, y. xPy \iff x \leftrightarrow y) \land t_P(A)$:

  $t_P(x \leftrightarrow y) \overset{\text{def}}{=} xPy$

  $t_P(B \ast C) \overset{\text{def}}{=} \exists Q, Q'. P = Q \ast Q' \land t_Q(B) \land t_{Q'}(C)$

  $t_P(B \rightarrow C) \overset{\text{def}}{=} \forall Q. ((\exists Q'. heap(Q') \land Q' = Q \ast P) \land heap(Q) \land t_Q(B) \Rightarrow (\exists Q'. heap(Q') \land Q' = Q \ast P \land t_{Q'}(C))$
Complexity of $\mathbb{SL}(\ast)$
$\text{SL}(\ast)$ is decidable

- Weak monadic 2nd order theory of $(D, f, =)$ where
  - $D$ is a countable set,
  - $f$ is a unary function,
  - $=$ is equality,

is decidable.  \[\text{[Rabin, Trans. of AMS 69]}\]

- $\text{MSO}$ can be translated into this theory.

- $\text{SL}(\ast) \sqsubseteq \text{MSO}$. 

\[ S\mathcal{L}(\ast) \text{ is not elementary recursive} \]

(\text{lists as finite words})

- FO3 over finite words is not elementary recursive. [Stockmeyer, PhD 74]

- Encoding a word by a list: position \( i \) has letter \( a_j \) iff the \((i + 1)\)th location has \( j \) predecessors.

- Word formula \( B_{\text{word}} \):

\[
(x_{\text{beg}} \rightarrow^+ x_{\text{end}}) \land (\forall x \ (x_{\text{beg}} \rightarrow^+ x) \land (x \rightarrow^+ x_{\text{end}}) \Rightarrow \#x \leq |\Sigma|)\]

- Translation of \( \mathcal{A} \): \( B_{\text{word}} \land t(\mathcal{A}) \)
  
  - \( t(x < y) \overset{\text{def}}{=} (x \rightarrow^+ y) \),
  
  - \( t(\forall x \ B) \overset{\text{def}}{=} \forall x. \ (x_{\text{beg}} \rightarrow^+ x) \land (x \rightarrow^+ x_{\text{end}}) \Rightarrow t(B) \),
  
  - \( t(\mathcal{P}_{a_i}(x)) \overset{\text{def}}{=} \#x = i \)

  (shortcut for a formula in \( S\mathcal{L}(\ast) \) of size \( O(i) \))
SL(∗) is not the ultimate decidable fragment!

- MSO is strictly more expressive than SL(∗) (and decidable).
  [Antonopoulos & Dawar, FOSSACS’09]
\(SL(*)\) is not the ultimate decidable fragment!

- **MSO** is strictly more expressive than \(SL(*)\) (and decidable).
  
  [Antonopoulos & Dawar, FOSSACS’09]

- Satisfiability for \(SL(*) + \neg\*)^n\) is also decidable.

\[(s, h) \models A_1 \neg\*)^n A_2 \text{ iff there is } h' \perp h \text{ such that } |\text{dom}(h')| \leq n, (s, h') \models A_1 \text{ and } (s, h \ast h') \models A_2.\]
SL(∗) is not the ultimate decidable fragment!

- MSO is strictly more expressive than SL(∗) (and decidable). [Antonopoulos & Dawar, FOSSACS’09]

- Satisfiability for SL(∗ +  ¬∗n) is also decidable. 
  \[(s, h) \models A_1  \neg* n A_2 \text{ iff there is } h' \perp h \text{ such that } |\text{dom}(h')| \leq n, (s, h') \models A_1 \text{ and } (s, h \ast h') \models A_2.\]

- Fragment L:

  \[A ::= \bot | x \mapsto y | \text{size } \leq k | \text{size } = k | A \ast A | A \lor A | A \land A\]

- Pushing the decidability border further! Satisfiability for SL restricted to formulae such that the left argument of any \(\neg\ast\)-formula belongs to L is decidable.
$SL(\rightarrow *)$ is equivalent to $SO$

[Brochenin & Demri & Lozes, CSL’08]
Proof schema for the equivalence

- $\text{SL}(\rightarrow) \subseteq \text{SL} \subseteq \text{DSO} \& \text{SO} \subseteq \text{DSO}$.  

- $\text{DSO} \subseteq \text{SL}(\rightarrow)$.  
  Encoding finite set of pairs by specialized patterns in memory.

- All translations are in logarithmic space.
Key ingredient: comparing numbers of predecessors

- \( \tilde{\#}x + c \preceq \tilde{\#}y + c' \) can be expressed in \( SL(\sim) \):
  - \( \preceq \in \{<, >, \leq, \geq, =\} \) and \( c, c' \in \mathbb{N} \),
  - by a formula of quadratic size in \( (c + c') \).
Key ingredient: comparing numbers of predecessors

• $\tilde{\#}x + c \preceq \tilde{\#}y + c'$ can be expressed in $\mathbb{SL}(-\star)$:
  • $\preceq \in \{<, >, \leq, \geq, =\}$ and $c, c' \in \mathbb{N}$,
  • by a formula of quadratic size in $(c + c')$.

• For instance, $\tilde{\#}x + c \leq \tilde{\#}y + c'$ is equivalent to:
  
  \[
  \forall n \quad \tilde{\#}y - c \leq n \implies \tilde{\#}x - c' \leq n.
  \]

1. $\tilde{\#}y - c \leq n$ is encoded by adding extra arrows in a controlled way.
2. The cardinal of the domain of the extra heap is precisely $n$. 
Key ingredient: comparing numbers of predecessors

- \( \tilde{\#}x + c \cong \tilde{\#}y + c' \) can be expressed in \( \mathbb{S}L(\rightarrow^*) \):
  - \( \cong \in \{<, >, \leq, \geq, =\} \) and \( c, c' \in \mathbb{N} \),
  - by a formula of quadratic size in \( (c + c') \).

- For instance, \( \tilde{\#}x + c \leq \tilde{\#}y + c' \) is equivalent to:
  \[ \forall n \; \tilde{\#}y - c \leq n \text{ implies } \tilde{\#}x - c' \leq n. \]

1. \( \tilde{\#}y - c \leq n \) is encoded by adding extra arrows in a controlled way.
2. The cardinal of the domain of the extra heap is precisely \( n \).

- Finite runs of Minsky machines can be encoded as memory states.
  ...but establishing \( \mathbb{D}SO \subseteq \mathbb{S}L(\rightarrow^*) \) is stronger than showing undecidability.
Elementary bits: the markers

- A *marker* is a specific pattern in the memory heap.
- A marker of *degree* $n$ and endpoint $l$.

\[
l_1 \quad \ldots \quad l_n
\]

with $\#l_1 = \ldots = \#l_n = 0$

- The location $l_0$ is an *extremity* in the marker ($\text{extr}(z)$).
A discipline on quantifications

- Quantification over $P_i$ can only occur in the scope of quantifications over $P_1, \ldots, P_{i-1}$.

- Quantifier depth of $B$ in $A$: maximal $i$ such that this occurrence of $B$ is in the scope of $\exists P_i$.

- Translation map of the form $t_i(B)$ depending of the quantifier depth $i$. 
Principle to encode an environment

- A pair \((l, l') \in \mathcal{E}(\mathcal{P}_i)\) is encoded by markers of consecutive degree \(N\) and \(N + 1\).

- The markers are introduced with septraction operator \(\neg\).
How to identify the original heap $h$

- No location has more than $k$ predecessors in $h$ where $s(z^m_0)$ is the endpoint of some new $k$-marker.

- Spectrum: sequence of degrees of new markers

  $\bullet \circ \bullet \circ \bullet \cdots \circ \bullet \circ \bullet \circ \bullet \circ \bullet \cdots \circ \bullet \circ \bullet \circ \bullet \circ \bullet

  \overset{n}{\bullet} : \text{There is a unique extremity } l \text{ with } \#l = n

  (in the environment part)

- A discipline for adding new markers

  \[
  \begin{align*}
  z^m_0 & \circ \bullet \cdots \circ z^M_0 : \circ z^m_1 & \cdots \circ z^M_1 = z^m_0 & \circ \bullet \cdots \circ z^M_1 \\
  \text{encodes } & \mathcal{E}(P_1) \\
  \end{align*}
  \]

  \[
  \begin{align*}
  z^m_0 & \circ \bullet \cdots \circ z^M_i : \circ z^m_{i+1} & \cdots \circ z^M_{i+1} = z^m_0 & \circ \bullet \cdots \circ z^M_{i+1} \\
  \text{encodes } & \mathcal{E}(P_{i+1})
  \end{align*}
  \]
Translating $P_j(x, y)$ – Summary

- $(l, l') \in E(P_i)$ iff there are markers with respective endpoint $l$ and $l'$ whose degrees are consecutive values strictly between $\tilde{\#}z_i^m$ and $\tilde{\#}z_i^M$.

- $z_i^m$ and $z_i^M$ are interpreted as locations outside the original memory heap.

- $\tilde{\#}z_i^m$ is strictly greater than the degree of any location in the original memory heap.

- Translation $t_i(P_j(x, y))$:

$$\exists z, z' (z \leftrightarrow x) \land (z' \leftrightarrow y) \land (\#z > \#z_j^m) \land (\#z' < \#z_j^M) \land (\#z' = 1 + \#z) \land extr(z) \land extr(z')$$
Translation

• Translation of $\exists P_i B$ at the $(i - 1)$ quantification depth:

$$\exists z_i^m, z_i^M \text{isol}(z_i^m) \land \text{isol}(z_i^M) \land$$

$$(\bullet \circ \bullet \cdots \circ \bullet \rightarrow (\bullet \circ \bullet \cdots \circ \bullet \land t_i(B)))$$

isol($x$) is an abbreviation for $\neg \exists y. (x \leftrightarrow y) \lor (y \leftrightarrow x)$.

• $t_i$ is the identity for $x = y$ and $x \leftrightarrow y$.

• $t_i(\exists x B)$ is defined as $\exists x \text{notonenv}(x) \land t_i(B)$ where $\text{notonenv}(x)$ guarantees that $x$ is not interpreted as a location used to encode environments.
Conclusion
Summary

This is mainly about $SL$ with one selector!

- $SL$ is as expressive as $SO$.

- Satisfiability/validity problem for $SL$ is undecidable.

- $SL(\neg \ast) \equiv SL$: $\ast$ is redundant in $SL$.

- $SL(\ast)$ is decidable with non-elementary complexity.
Summary

This is mainly about $SL$ with one selector!

- $SL$ is as expressive as $SO$.

- Satisfiability/validity problem for $SL$ is undecidable.

- $SL(\neg \ast) \equiv SL$: $\ast$ is redundant in $SL$.

- $SL(\ast)$ is decidable with non-elementary complexity.

$SL(\neg \ast) \equiv SL \equiv SO$ also holds with more than one selector.
(auxiliary memory cells are even easier to identify)