Logical Investigations on Separation Logics

Lecture 1: First Steps in Separation Logics

Stéphane Demri (CNRS) & Morgan Deters (NYU)

ESSLLI 2015, Barcelona, August 2015
In Memoriam: Morgan Deters
What is in the course?

Complexity/Expressiveness/Decision Procedures for Separation Logics

- Lecture 1: First steps in separation logics.
- Lecture 2: Propositional separation logics.
- Lecture 3: Expressiveness of first-order separation logics.
- Lecture 4: Relationships with other logics.
- Lecture 5: Decision procedures.

but also first-order logic, temporal logics, data logics, QBF, model-checking, SMT, translations, etc.
What can you expect to learn?

- Basics of separation logics and relationships with other logics.
- Proof techniques to decide separation logics, to show undecidability or expressiveness results.
- Basics of mechanisation for separation logics.
- By contrast, the verification techniques with separation logics are partially touched.
Background

1. Necessary background
   • Basics of first-order logic.
   • Basics of temporal/modal logics.

2. Optional background
   • Basics of complexity theory
   • Minsky machines.
   • Basics of model-checking.
Course material

- Lecture notes (with exercises) available on
  www.lsv.fr/~demri/essllli15-course.html

- Slides available on a daily basis.

- Do not hesitate to contact me during ESSLLI’15 or to send me emails at demri@lsv.fr.
Formal Methods for Program Verification
Verification at the heart of computer science

- Digital systems are everywhere. Desktops, embedded systems, cellular phones, etc.

- Needs for verifying functional/security properties:
  - Hardware components
  - Software (programs, communication protocols, web applications, ...)

Formal verification is a process in which mathematical techniques are used to guarantee the correctness of a design with respect to some specified behavior.

[Halpern et al., BSL 01]

- See also Turing’s paper ”Checking a large routine” (1949).
Verification as a logical problem

- Properties are represented by logical formula. “The system $S$ never reaches a bad state” → $A G \neg \text{bad}$.

- Logical problems involve abstract models and formulae.

- Development of procedures to solve these problems. automata, analytic proof systems, ad-hoc methods . . .

- Ultimate goal: automatic verification.

- There are theoretical limits for this entreprise.
  - The halting problem for Turing machines is undecidable. [Turing, 37]
  - The set of valid first-order formulae is undecidable. [Church, JSL 36]
Methodology

- System, property $\mapsto$ model, logical formula.

- Logical problems:
  - Decision problems (model-checking, validity, ...)
  - Search problems (controller synthesis, query checking, ...)

- Analysis of the computational resources to solve the problems
  - Decision procedures vs. undecidability.
  - Complexity in time or memory space.
General approaches for verification

- Model-checking (automata, games, ...).
- Abstract interpretation (domain-specific models).
- Deduction verification (computer-aided proofs).
- etc.
Floyd/Hoare Logic
Floyd-Hoare logic

- Hoare triple: $\{\varphi\} \ C \ {\psi}\}$ (partial correctness).

  - Precondition $\varphi$.  
  - Postcondition $\psi$.  
  - Command/program $C$.  

[Hoare, C. ACM 69; Floyd, 1967]

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

- Proof system with axioms and deduction rules to derive new triples.

- Two commands can be composed:

$$
\begin{align*}
\{\varphi\} \ C_1 \ {\psi}\} \ {\psi}\} \ C_2 \ {\chi}\} \\
\{\varphi\} \ C_1; \ C_2 \ {\chi}\}
\end{align*}
$$
More rules

- Strengthening preconditions / weakening postconditions:

\[ \varphi \Rightarrow \varphi' \quad \{ \varphi' \} \text{C} \{ \psi \} \quad \psi \Rightarrow \psi' \]

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

- \( \varphi \Rightarrow \varphi' \): instance of the (logical) entailment problem, subproblem of the validity problem.

- About variable assignments:

\[ \{ \varphi[e/x] \} \ x := e \ \{ \varphi \} \]

- While rule:

\[ \{ \varphi \land B \} \text{C} \{ \varphi \} \]

\[ \{ \varphi \} \text{ while } B \ \text{do } \text{C} \{ \varphi \land \neg B \} \]
The rule of constancy

- Conditional rule:

\[
\begin{align*}
\{ \varphi \land B \} & \quad C_1 \quad \{ \psi \} \\
\{ \varphi \land -B \} & \quad C_2 \quad \{ \psi \}
\end{align*}
\]

\[
\{ \varphi \} \text{ if } B \text{ then } C_1 \text{ else } C_2 \quad \{ \psi \}
\]

- Rule of constancy:

\[
\begin{align*}
\{ \varphi \} & \quad C \quad \{ \psi \} \\
\{ \varphi \land \psi' \} & \quad C \quad \{ \psi \land \psi' \}
\end{align*}
\]

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

\[
\begin{align*}
\{ x = 3 \} & \quad x := 4; \quad z := x \quad \{ x = 4 \} \\
\{ x = 3 \land y = 8 \} & \quad x := 4; \quad z := x \quad \{ x = 4 \land y = 8 \}
\end{align*}
\]
Partial correctness and relative completeness

- Small-step operational semantics:
  \[ s, C \leadsto s', C' \]
  where \( s \) and \( s' \) are variable assignments \( \{x_1, x_2, \ldots\} \rightarrow \mathbb{V} \).

- Termination: \( s, C \leadsto^{*} s', \text{skip} \).

- Validity of \( \{\varphi\} C \{\psi\} \):
  \[ s, C \leadsto^{*} s', \text{skip} \text{ and } s \models \varphi \text{ imply } s' \models \psi \]

- Partial correctness: every derivable triple is valid.

- Relative completeness: every valid triple \( \{\varphi\} C \{\psi\} \) is derivable in the proof system.
Dijkstra’s weakest preconditions

- $wp(C, \psi)$: minimal precondition $\varphi$ that validates $\{\varphi\} \subseteq \{\psi\}$.

- Completeness: the assertion logic $\mathcal{L}$ can express the weakest preconditions for any $C$ and $\psi$.

- The first proof for relative completeness uses weakest preconditions and a structural induction on $C$.  

  [Cook, SIAM 78]

- Validity for $\{\varphi\} \subseteq \{\psi\}$:
  1. to compute $wp(C, \psi)$,
  2. to check the validity of $\varphi \Rightarrow wp(C, \psi)$.
Computing weakest preconditions (examples)

\[
\begin{align*}
wp(\text{skip}, \psi) & \quad \overset{\text{def}}{=} \quad \psi \\
wp(x := e, \psi) & \quad \overset{\text{def}}{=} \quad \psi[e/x] \\
wp(C_1; C_2, \psi) & \quad \overset{\text{def}}{=} \quad \wp(C_1, \wp(C_2, \psi)) \\
wp(\text{if } B \text{ then } C_1 \text{ else } C_2, \psi) & \quad \overset{\text{def}}{=} \quad (B = \top \land \wp(C_1, \psi)) \land (B = \bot \land \wp(C_2, \psi)) \\
wp(\text{while } B \text{ do } C, \psi) & \quad \overset{\text{def}}{=} \quad l \land \forall y_1, \ldots, y_k
\left( (((B = \top \land l) \Rightarrow \wp(C, l)) \land
((B = \bot \land l) \Rightarrow \psi)) [y_i/x_i] \right)
\end{align*}
\]

\((x_1, \ldots, x_k \text{ are the assigned variables in } C)\)
When separation logic enters into the play

\[
\begin{align*}
  x & := \textbf{cons}(e) \quad \text{allocation} \\
  x & := \llbracket e \rrbracket \quad \text{lookup/fetch assignment} \\
  \llbracket e \rrbracket & := e' \quad \text{mutation/heap assignment} \\
  \text{dispose}(e) & \quad \text{deallocation(pointer disposal)}
\end{align*}
\]

Heap \( \mathcal{h} \): finite set of pairs made of a location and a value in \( \mathcal{V} \)

\[
(s, \mathcal{h} \cup \{ \llbracket e \rrbracket \mapsto n \}), \llbracket e \rrbracket := e' \leadsto (s, \mathcal{h} \cup \{ \llbracket e \rrbracket \mapsto \llbracket e' \rrbracket \}), \text{skip}
\]

▶ Rule of constancy:

\[
\begin{align*}
\{ \varphi \} & \subset \{ \psi \} \\
\{ \varphi \land \psi' \} & \subset \{ \psi \land \psi' \}
\end{align*}
\]

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

▶ Unsoundness of the rule of constancy with pointers:

\[
\begin{align*}
\{ \varphi_1 \} \ [x] & := 4 \ \{ \varphi_2 \} \\
\{ \varphi_1 \land [y] = 3 \} \ [x] & := 4 \ \{ \varphi_2 \land [y] = 3 \} \quad \text{if } x = y \text{ then } [x] = [y]
\end{align*}
\]
Frame rule and separating conjunction

- Frame rule:

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

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

\[
\{ [x] = 5 \} \ [x] := 4 \ {[[x] = 4]} \\
{[[x] = 5 \ast [y] = 3]} \ [x] := 4 \ {[[x] = 4 \ast [y] = 3]}
\]

- \((s, h) \models [x] = 5 \ast [y] = 3\) implies \( x \neq y \).

- Separation logic

  **assertion logic** + programming language + deduction rules

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

  See also [Burstall, MI 72]

- Extension of Hoare logic with separating connectives.

  [Reynolds, LICS’02]
A taste of separation logic

- $\varphi \ast \psi$ is true if the heap can be divided into two disjoint parts, one satisfies $\varphi$, the other one $\psi$.

- $\varphi \rightarrow \ast \psi$ is true if whenever $\varphi$ is true for a (fresh) disjoint heap, $\psi$ is true for the combined heap.
  (useful for memory allocation)

- Models can be finite graphs (memory states).

- While evaluating a formula, models can be updated.

- Automatic program analysis.
  See tools Smallfoot, Space Invader, Abductor, Slayer, etc.

- Logical foundations.
  Proof systems, complexity, expressive power, etc.
What else for reasoning about pointer programs?

- Examples of logical specification languages
  - 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.
  - Tool VeriFast [Vogels & Jacobs & Piessens, 2015]

- 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]

- etc.
Why separation logics today?

➢ Deductive verification of programs with mutable data structures.

➢ Strong logical foundations (topic of the course).

➢ Tools & SL-COMP’14, SL-COMP’15 etc.

➢ Convergence between
  1. SMT technology, and
  2. verification techniques with separations logics.
  (two active subfields in Automated Reasoning and Computer-Aided Verification)
A Core Version of Separation Logic
Memory states with one record field

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

- **Memory state** $(\varsigma, \eta)$:
  - **Store** $\varsigma : \text{PVAR} \rightarrow \mathbb{N}$.
  - **Heap** $\eta : \mathbb{N} \rightarrow \mathbb{N}$ with finite domain.

1. No distinction between locations(addresses) and values.
2. $\eta \approx (\mathbb{N}, \mathcal{R})$ with finite and functional $\mathcal{R}$.
3. Finite forests of finite “trees” with cycles as roots.

![Diagram of memory states with one record field]

In the diagram, nodes represent memory states, and edges represent transitions. The root nodes (12) indicate cycles, and the arrows show the direction of transition between states.
Disjoint heaps – Smooth separation

- 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 \( h_1 \uplus h_2 \).
A heap with \( k \geq 1 \) record fields is a partial function \( h : \mathbb{N} \rightarrow \mathbb{N}^k \) with finite domain.

\( h \subseteq h' \): the heap \( h' \) is a conservative extension of \( h \) (in particular, \( \text{dom}(h) \subseteq \text{dom}(h') \)).

\( k\text{SL} \): first-order separation logic with \( k \) record fields.
Syntax and semantics for \( kSL \)

- **Quantified variables** \( \text{FVAR} = \{u_1, u_2, u_3, \ldots\} \).
- **Expressions**: \( e ::= x_i | u_j \)
- **Atomic formulae**: \( \pi ::= e = e' | e \leftrightarrow e_1, \ldots, e_k | \text{emp} | \bot \)
- **Formulae**: \( \varphi ::= \pi | \varphi \land \psi | \neg \varphi | \varphi \ast \psi | \varphi \rightarrow \psi | \exists u \varphi \)
- **Model** \((s, h)\) and variable assignment \( f \).

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

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

\[
(s, h) \models f e \leftrightarrow e_1, \ldots, e_k \\
\overset{\text{def}}{\iff} \\
[e] \in \text{dom}(h) \text{ and } h([e]) = ([e_1], \ldots, [e_k])
\]
Binary modality: separating conjunction

\[(s, h) \models f \varphi_1 * \varphi_2\]

\[\text{def} \iff\]

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

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

\[\varphi_1 * \varphi_2 \iff \varphi_2 * \varphi_1 \quad (\varphi_1 * \varphi_2) * \varphi_3 \iff \varphi_1 * (\varphi_2 * \varphi_3) \quad \varphi * \text{emp} \iff \varphi\]

\[\varphi * \varphi \not\iff \varphi\]
universal quantifies over an infinite set!

$$(s, h) \models_f \varphi_1 \rightarrow^* \varphi_2$$

$\iff$

for all $h'$,

if $h \perp h'$ and $(s, h') \models_f \varphi_1$,

then $(s, h \cup h') \models_f \varphi_2$

$\varphi^*(\varphi \rightarrow^* \psi) \Rightarrow \psi \quad (\text{emp} \rightarrow^* \varphi) \Leftrightarrow \varphi \quad \varphi_1 \rightarrow^*(\varphi_2 \rightarrow^* \psi) \Leftrightarrow (\varphi_1 \ast \varphi_2) \rightarrow^* \psi$

As in van Benthem’s sabotage modal logic or in logics of public announcements, the model can be modified during the evaluation of formulae. (see also [Aucher et al, M4M’07])
Semantics – Variants

- \((s, h) \models f \exists u \varphi \iff \text{there is } l \in \mathbb{N} \text{ such that } (s, h) \models f[u \mapsto l] \varphi\)
  where \(f[u \mapsto l]\) is the assignment equal to \(f\) except that \(u\) takes the value \(l\).

- Precise points-to atomic formula \(x \mapsto y_1, \ldots, y_k:\)
  \[(x \mapsto y_1, \ldots, y_k) \land \neg (\neg \text{emp} \land \neg \text{emp})\]

- No weakening: \((x \mapsto y) \ast (x' \mapsto y') \not\Rightarrow (x \mapsto y)\).

- No contraction: \((x \mapsto y) \not\Rightarrow (x \mapsto y) \ast (x \mapsto y)\).

- \((x \mapsto y) \iff (x \mapsto y) \ast \top\).

- In other refined models, \(x \mapsto y_1, \ldots, y_k\) is understood as
  \[(x \mapsto y_1) \ast (x + 1 \mapsto y_2) \ast \cdots \ast (x + (k - 1) \mapsto y_k)\]
Fragments

- $kSLk': kSL$ with at most $k'$ quantified variables.
- $kSLk'(\neg *)$ denotes the fragment of $kSLk'$ without $\ast$.
- $kSLk'(\ast)$ denotes the fragment of $kSLk'$ without $\neg \ast$.
- $kSL$ viewed as a syntactic fragment of $(k + 1)SL$. (by default, we deal with $1SL$)
Memory state representation with $k = 2$
Circular list $(v_1, v_2, v_3)$

$h(l_1) = (v_1, l_2) \quad h(l_2) = (v_2, l_3) \quad h(l_3) = (v_3, l_1)$

$(s, h) \models (\exists u \ x \mapsto u, y) \ast (\exists u \ y \mapsto u, z) \ast (\exists u \ z \mapsto u, x)$
Memory state representation with $k = 1$

\[ x = l_1 \quad y = l_2 \quad z = l_3 \]

\[ h(l_1) = l_2 \quad h(l_2) = l_3 \quad h(l_3) = l_1 \]

\[(s, h) \models x \leftrightarrow y \land z \leftrightarrow x \quad (s, h) \models x \leftrightarrow y \ast y \leftrightarrow z \ast z \leftrightarrow x\]
Decision problems

- \( \mathcal{L} \): some fragment of \( kSL \).

- Satisfiability problem:
  - input: formula \( \varphi \) in \( \mathcal{L} \).
  - question: are there \((s, h)\) and \(f\) such that \((s, h) \models_f \varphi\)?

- \( \varphi \models \psi \): for all \((s, h)\), \((s, h) \models \varphi\) implies \((s, h) \models \psi\).

- Entailment problem: \( \varphi \models \psi \) ?

- Validity problem, . . .

\[
\begin{array}{c}
\varphi \Rightarrow \varphi' \quad \{\varphi'\} \subseteq \{\psi\} \quad \psi \Rightarrow \psi' \\
\{\varphi\} \subseteq \{\psi'\} \quad \text{strengthen/weaken}
\end{array}
\]
Standard inference rules for mutation

\[
x := \text{cons}(e) \quad \text{allocation}
\]
\[
x := [e] \quad \text{lookup}
\]
\[
[e] := e' \quad \text{mutation}
\]
\[
dispose(e) \quad \text{deallocation}
\]

\[
(s, h), x := \text{cons}(e) \leadsto (s[x \mapsto n], h \uplus \{n \mapsto [e]\}), \text{skip}
\]

\[
n \mapsto [e] \text{ is a new memory cell}
\]

\[
(s, h), \text{dispose}(y) \leadsto (s, h \setminus \{[y] \mapsto m\}), \text{skip}
\]

where \([y] \in \text{dom}(h)\) and \(h([y]) = m\)

\[
(s, h \uplus \{[e] \mapsto n\}), [e] := e' \leadsto (s, h \uplus \{[e] \mapsto [e']\}), \text{skip}
\]
More inference rules

- **Local form (MUL)**

\[
\{\exists u \; e \mapsto u\} [e] := e' \; \{e \mapsto e'\}
\]

- **Global form (MUG) \approx MUL + frame rule**

\[
\{(\exists u \; e \mapsto u) \ast \varphi\} [e] := e' \; \{e \mapsto e' \ast \varphi\}
\]

- **Backward-reasoning form (MUBR)**

\[
\{(\exists u \; e \mapsto u) \ast (e \mapsto e' \ast \varphi)\} [e] := e' \; \{\varphi\}
\]
Global rules for deallocation and allocation

\[(\exists u \ e \mapsto u) \ast \varphi \} \ dispose(e) \{\varphi\}\]

\[\{\varphi\} \ x ::= \ cons(e) \ {(x \mapsto e) \ast \varphi}\]

\[x \ not \ free \ in \ e \ and \ in \ \varphi\]
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:

  $$(x \mapsto x) \triangleright \bot$$

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

  $$\forall u \ (u \mapsto e \triangleright u \mapsto e')$$
Helpful macro: septraction

- **Septraction** $\overrightarrow{*}$: existential version of $\rightarrow$.

\[ \varphi_1 \overrightarrow{*} \varphi_2 \overset{\text{def}}{=} \neg (\varphi_1 \rightarrow \neg \varphi_2) \]

\[ \mathcal{h} \models_f \varphi_1 \overrightarrow{*} \varphi_2 \]

iff

there is $\mathcal{h'} \perp \mathcal{h}$ such that $\mathcal{h'} \models_f \varphi_1$ and $\mathcal{h'} \cup \mathcal{h} \models_f \varphi_2$. 
Relationship between $*$ and $\neg*$

- $\neg*$ is the **adjunct** of $*$:
  \[(\varphi \ast \psi) \Rightarrow \chi \text{ is valid iff } \varphi \Rightarrow (\psi \ast \chi) \text{ is valid.}\]

- ...but the formula below is not valid
  \[((\varphi \ast \psi) \Rightarrow \chi) \iff (\varphi \Rightarrow (\psi \ast \chi))\]

- In Lecture 3, we show the relationships between 1SL, 1SL($*$) and 1SL($\neg*$).

- In Lectures 2 & 5, we analyse the relationships between 1SL0, 1SL0($*$) and 1SL0($\neg*$).
Simple properties stated in 1SL

- The value of $\overline{u}$ is in the domain of the heap:
  \[
  \text{alloc}(\overline{u}) \overset{\text{def}}{=} \exists u \overline{u} \leftrightarrow u \quad \text{(variant of } (\overline{u} \leftrightarrow \overline{u}) \rightarrow \bot)\]

- Unique cell $x_1 \mapsto x_2$:
  \[
  x_1 \mapsto x_2 \overset{\text{def}}{=} x_1 \leftrightarrow x_2 \land \neg \exists u \ (u \neq x_1 \land ((u \leftrightarrow u) \rightarrow \bot))
  \]

- The domain $\text{dom}(h)$ has exactly one location:
  \[
  \text{size} = 1 \overset{\text{def}}{=} \neg \text{emp} \land \neg (\neg \text{emp} \ast \neg \text{emp})
  \]

- The domain $\text{dom}(h)$ has exactly two locations:
  \[
  \text{size} = 2 \overset{\text{def}}{=} (\neg \text{emp} \ast \neg \text{emp}) \land \neg (\neg \text{emp} \ast \neg \text{emp} \ast \neg \text{emp})
  \]

- Generalisation to $\text{size} \sim \alpha$.  

Numbers of predecessors

- \( u \) has at least \( \alpha \) predecessors:
  \[
  \#u \geq \alpha \overset{\text{def}}{=} (\exists \, \bar{u} \, (\bar{u} \leftrightarrow u)) \ast \cdots \ast (\exists \, \bar{u} \, (\bar{u} \leftrightarrow u))
  \]

- \( u \) has at most \( \alpha \) predecessors: \( \#u \leq \alpha \overset{\text{def}}{=} \neg (\#u \geq \alpha + 1) \).

- \( u \) has exactly \( \alpha \) predecessors:
  \[
  \#u = \alpha \overset{\text{def}}{=} (\#u \geq \alpha) \land \neg (\#u \geq \alpha + 1)
  \]
More about numbers of predecessors

- \( \sim \in \{\leq, \geq, =\} \) and \( i, \alpha \geq 0 \).

\[
\begin{align*}
\#u^0 \sim \alpha & \overset{\text{def}}{=} \#u \sim \alpha \\
\#u^{i+1} \sim \alpha & \overset{\text{def}}{=} \exists \ u \ u \leftrightarrow \bar{u} \land \#u^i \sim \alpha \\
\#u^{-i-1} \sim \alpha & \overset{\text{def}}{=} \exists \ u \ (\bar{u} \leftrightarrow u) \land \#u^{-i} \sim \alpha
\end{align*}
\]

- \( \#u^6 \geq 2 \) states that there is a (necessarily unique) location at distance 6 from \( u \) and its number of predecessors is greater than or equal to 2.

- \( \#u^{-5} \leq 2 \) states that there is a (not necessarily unique) location at distance \(-5\) from \( u \) and its number of predecessors is not strictly greater than 2.
Reachability predicate in 1SL2(\(*\))

- \(u \neq \overline{u}\) and path from \(u\) to \(\overline{u}\) and nothing else except loops:

\[
\text{reach}'(u, \overline{u}) \overset{\text{def}}{=} \#u = 0 \land \text{alloc}(u) \land \neg\text{alloc}(\overline{u}) \land \\
\forall \overline{u} ((\text{alloc}(\overline{u}) \land \#\overline{u} = 0) \Rightarrow \overline{u} = u) \land \\
\forall u ((\#u \neq 0 \land u \neq \overline{u}) \Rightarrow (\#u = 1 \land \text{alloc}(u)))
\]

- There is a path from \(u\) to \(\overline{u}\):

\[
\text{reach}(u, \overline{u}) \overset{\text{def}}{=} u = \overline{u} \lor (\top \ast \text{reach}'(u, \overline{u}))
\]

- Path from \(u\) to \(\overline{u}\) and nothing else:

\[
\text{sreach}(u, \overline{u}) \overset{\text{def}}{=} \text{reach}(u, \overline{u}) \land \neg(\neg\text{emp} \ast \text{reach}(u, \overline{u}))
\]

\(\text{sreach}\): strict reachability (a.k.a. list segment predicate \(\text{ls}\)).
Expressive power / Decidability / Complexity

1SL ≡ 1DSOL ≡ 1WSOL ≡ 1SL(−∗), undec.

1SL2, undec.               1SL(*), dec., non-elem.

1SL1, PSPACE-C 1SL2(−∗) ≡ 1DSOL, undec. L3 1SL2(*), non-elem. L4

1SL0, PSPACE-C L2&5

1SLk = 1SL restricted to k quantified variables

- [Calcagno & Yang & O’Hearn, FSTTCS’01] 1SL0
- [Brochenin & Demri & Lozes, IC 12] 1SL(−∗)
- [Demri & Galmiche & Larchey-Wendling & Mery, CSR’14] 1SL1
- [Demri & Deters, CSL-LICS’14] 1SL2(−∗)
- [Demri & Deters, TOCL 15] 1SL2(*)
Conclusion

- Lecture 1: First steps in separation logics.
  - Introduction to separation logics
  - Playing with formulae.

- Lecture 2: Propositional separation logics.
  - \text{PSPACE}-hardness of $k\text{SL0}$ fragments.
  - Presentation of complexity results.
  - From the halting problem to a propositional separation logic.

- Lecture 3: Expressiveness of first-order separation logics.

- Lecture 4: Relationships with other logics.

- Lecture 5: Decision procedures.