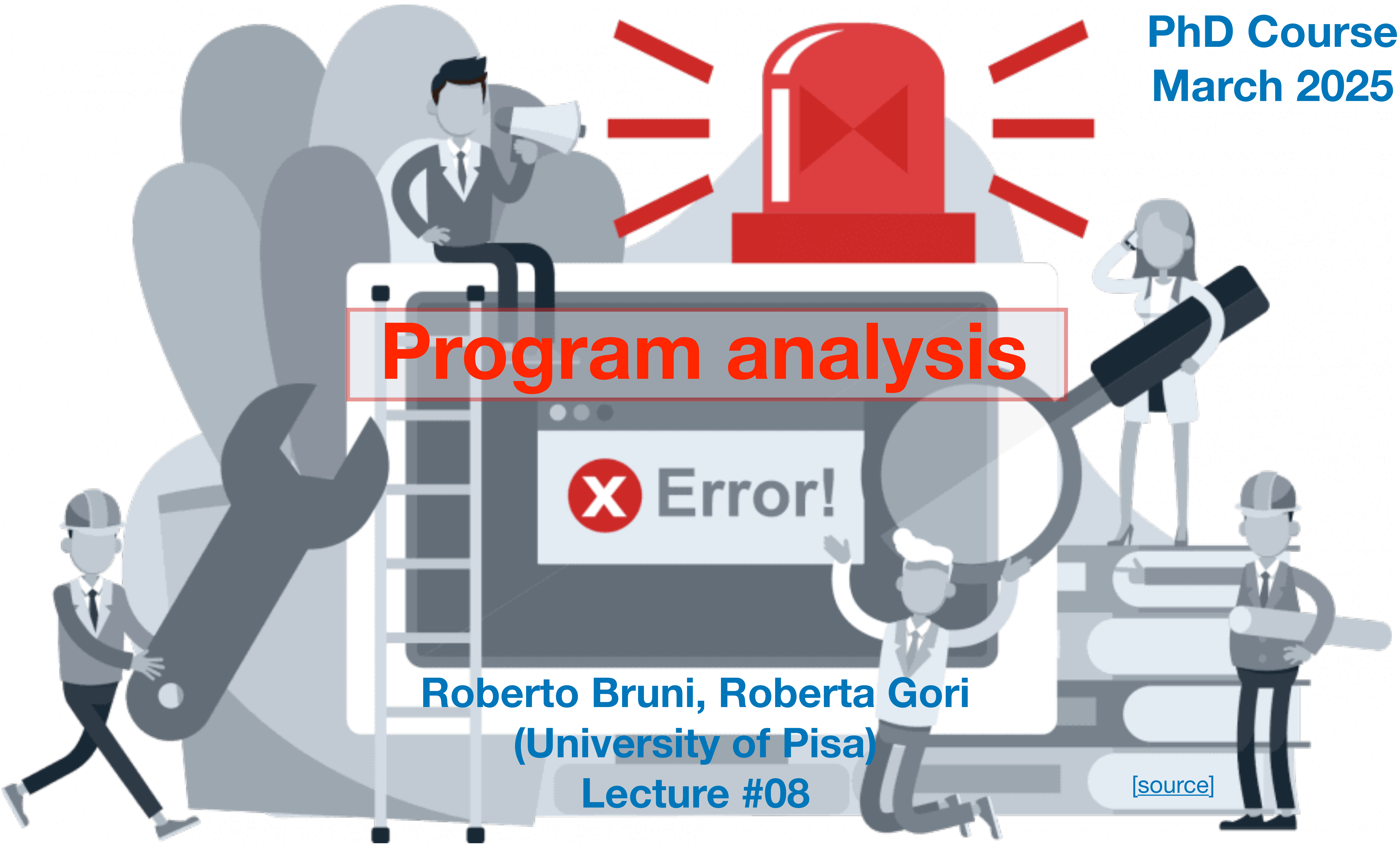
PhD Course
March 2025

Program analysis



Roberto Bruni, Roberta Gori
(University of Pisa)

Lecture #08

[source]

Separation SIL

OOPSLA 2025

Revealing Sources of (Memory) Errors via Backward Analysis

FLAVIO ASCARI, University of Pisa, Italy
ROBERTO BRUNI, University of Pisa, Italy
ROBERTA GORI, University of Pisa, Italy
FRANCESCO LOGOZZO, Meta Platforms, USA

Sound over-approximation methods are effective for proving the absence of errors, but inevitably produce false alarms that can hamper programmers. In contrast, under-approximation methods focus on bug detection and are free from false alarms. In this work, we present two novel proof systems designed to locate the source of errors via backward under-approximation, namely Sufficient Incorrectness Logic (SIL) and its specialization for handling memory errors, called Separation SIL. The SIL proof system is minimal, sound and complete for Lisbon triples, enabling a detailed comparison of triple-based program logics across various dimensions, including negation, approximation, execution order, and analysis objectives. More importantly, SIL lays the foundation for our main technical contribution, by distilling the inference rules of Separation SIL, a sound and (relatively) complete proof system for automated backward reasoning in programs involving pointers and dynamic memory allocation. The completeness result for Separation SIL relies on a careful crafting of both the assertion language and the rules for atomic commands.

CCS Concepts: • **Theory of computation** → **Logic and verification**; *Proof theory*; *Hoare logic*; **Separation logic**; *Programming logic*.

Additional Key Words and Phrases: Sufficient Incorrectness Logic, Incorrectness Logic, Outcome Logic

ACM Reference Format:
Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. 2025. Revealing Sources of (Memory) Errors via Backward Analysis. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 127 (April 2025), 28 pages. https://doi.org/10.1145/3720486

1 Introduction

Formal methods aim to automate the improvement of software reliability and security. Notable success stories are, e.g., the Astrée static analyzer [Blanchet et al. 2003], the SLAM model checker [Ball and Rajamani 2001], the certified C compiler CompCert [Leroy 2009], VCC for safety properties verification [Cohen et al. 2009], and the Frama-C platform for the integration of many C code analyses [Baudin et al. 2021]. Despite that, effective program correctness methods struggle to reach mainstream adoption, mostly because they exploit over-approximation to handle decidability issues and false positives are seen as a distraction by expert programmers. Being free from false positives is possibly the reason why *under-approximation* approaches for bug-finding, such as testing and bounded model checking, are preferred in industrial applications. Incorrectness Logic (IL) [O'Hearn 2020] is a new program logic for bug-finding: *any error state found in the post can be produced by some input states that satisfy the pre.* However, IL triples are not able to characterize precisely *the input states that are responsible for a given error.* This is possibly rooted in the *forward* flavor of the under-approximation, which follows the ordinary direction of code execution.

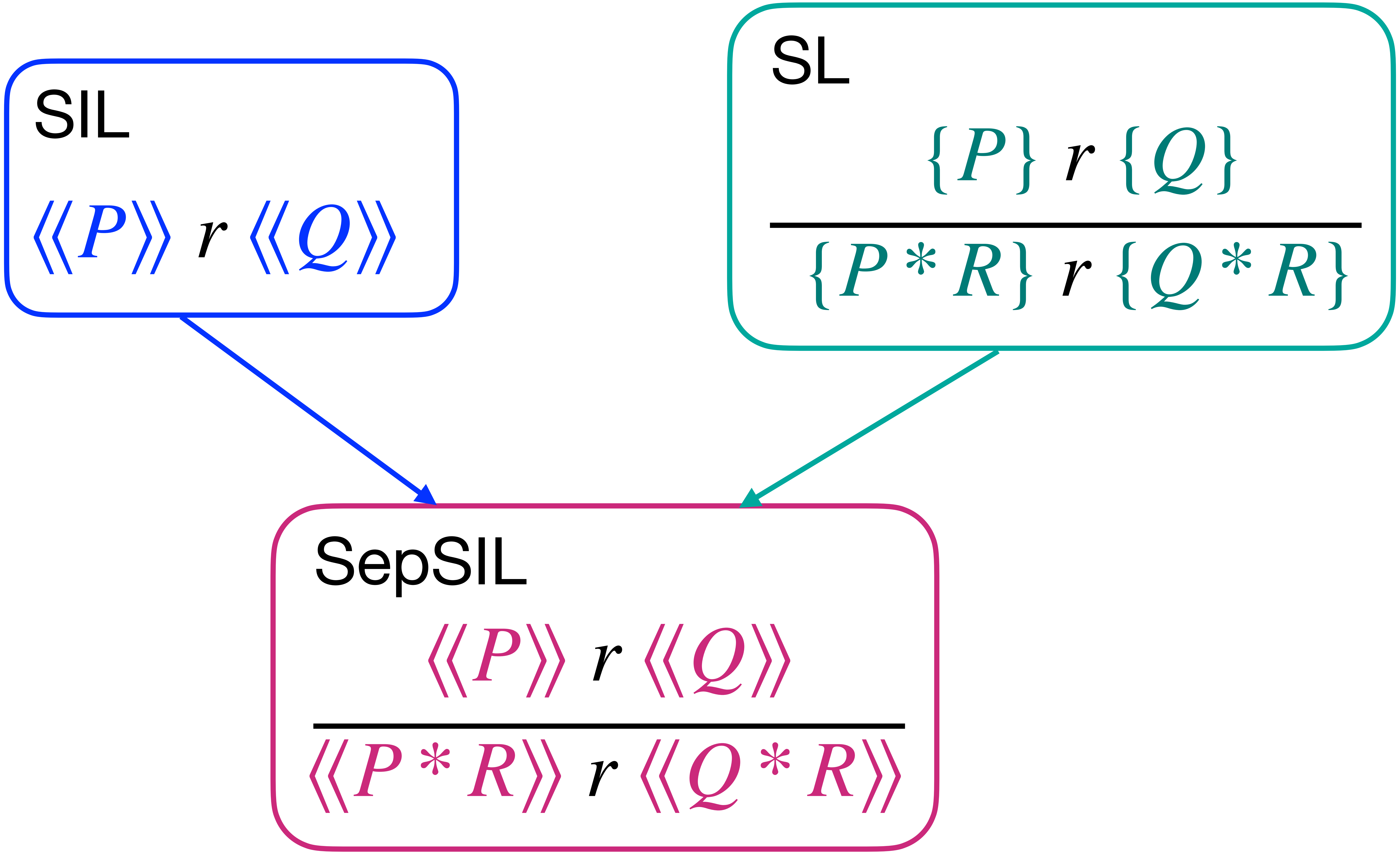
Authors' Contact Information: Flavio Ascari, University of Pisa, Pisa, Italy, flavio.ascari@phd.unipi.it; Roberto Bruni, University of Pisa, Pisa, Italy, bruni@di.unipi.it; Roberta Gori, University of Pisa, Pisa, Italy, roberta.gori@unipi.it; Francesco Logozzo, Meta Platforms, Seattle, USA, logozzo@meta.com.

ACM 2475-1421/2025/4-ART127
https://doi.org/10.1145/3720486

Proc. ACM Program. Lang., Vol. 9, No. OOPSLA1, Article 127. Publication date: April 2025.

"Separation SIL can yield more succinct postconditions and provide stronger guarantees than ISL and can support effective backward reasoning"

SepSIL = SIL + SL



Regular commands

regular command

atomic command

$$r ::= e \mid r_1 ; r_2 \mid r_1 + r_2 \mid r^\star$$

choice

Kleene star

$$e ::= \mathsf{skip} \mid b? \mid x := a \mid x := [y] \mid [x] := y \mid x := \mathsf{alloc}(\,) \mid \mathsf{free}(x)$$

simplified

$x := [y]$ // read

$[x] := y$ // write

Assertion language

assertion

$$
\begin{aligned}
P ::= \;& \mathsf{true} \mid \mathsf{false} \mid a_1 < a_2 \mid a_1 = a_2 \mid \ldots \\
\mid \;& \neg P \mid P_1 \wedge P_2 \mid \exists x \,.\, P \mid \ldots \\
\mid \;& \mathsf{emp} \\
\mid \;& a_1 \mapsto a_2 \\
\mid \;& P_1 * P_2 \\
\mid \;& x \not\mapsto
\end{aligned}
$$

Boolean and classical assertions

structural assertions

track deallocated locations

Local axioms: write

SL

$$\frac{}{\{x \mapsto \_\}\ [x] := y\ \{x \mapsto y\}}$$

ISL

$$\frac{}{[x \mapsto v]\ [x] := y\ [\mathsf{ok} : x \mapsto y]}$$

SepSIL

$$\frac{}{\langle\!\langle x \mapsto \_ \rangle\!\rangle\ [x] := y\ \langle\!\langle x \mapsto y \rangle\!\rangle}$$

weakest pre

Local axioms: read

SL

$$\overline{\{y \mapsto v\}\ x := [y]\ \{x = v \land y \mapsto v\}}$$

ISL

$$\overline{[y \mapsto v]\ x := [y]\ [\mathrm{ok} : x = v \land y \mapsto v]}$$

SepSIL

$$\overline{\langle\!\langle y \mapsto v \land (v = x') \rangle\!\rangle\ x := [y]\ \langle\!\langle y \mapsto v \land (x = x') \rangle\!\rangle}$$

Hoare style

applicable to any post

Local axioms: allocation

SL

$$\frac{}{\{\mathsf{emp}\}\ x := \mathsf{alloc}()\ \{x \mapsto \_\}}$$

ISL

$$\frac{}{[\mathsf{emp}]\ x := \mathsf{alloc}()\ [\mathsf{ok} : x \mapsto \_]}$$

SepSIL

$$\frac{}{\langle\!\langle \mathsf{emp} \rangle\!\rangle\ x := \mathsf{alloc}()\ \langle\!\langle x \mapsto \_ \rangle\!\rangle}$$

Local axioms: dispose

SL

$$\{x \mapsto \_\}\ \mathsf{free}(x)\ \{\mathsf{emp}\}$$

ISL

$$\frac{}{[x \mapsto v]\ \mathsf{free}(x)\ [\mathsf{ok} : x \not\mapsto\ ]}$$

SepSIL

$$\frac{}{\langle\!\langle x \mapsto \_ \rangle\!\rangle\ \mathsf{free}(x)\ \langle\!\langle x \not\mapsto\ \rangle\!\rangle}$$

using cons can be strengthened to $x \mapsto v$

Different proofs of a real bug

Use-after-lifetime bug

```
void deref_after_pb(std::vector<int> *v) {
  int *x = &v->at(1);
  v->push_back(42);
  std::cout << *x << "\n"; }
```

from std::vector library, can deallocate and then reallocate v

```
push_back.cpp:7: error: VECTOR_INVALIDATION. accessing memory that was
potentially invalidated by 'std::vector::push_back()' on line 6.
  5.       int *x = &(v->at(1));
  6.       v->push_back(42);
  7. >     std::cout << *x << "\n"; }
```

if v is reallocated, x is invalidated

The C++ use-after-lifetime bug (above); the Pulse error message (below).

abstracted from real occurrences at Facebook

From C++ to regular commands

$[v \mapsto a * a \mapsto -]$ `client(v)` $[er(L_{rx})\colon \exists a'.\ v \mapsto a' * a' \mapsto - * a \not\mapsto]$

```
void push_back(int **v)
{
  if (nondet()) {
    free(*v);
    *v = malloc(sizeof(int));
  }
}
```

```
void client(v) {
  int* x = *v;
  push_back(v);
  *x = 88; }
```

C version

push_back(v) $\triangleq$
  local z, y in
    $z :=$ *;
    (assume($z \neq 0$); L_{rv}: $y := [v]$;
     L_f: free(y);
     $y :=$ malloc(); $[v] := y$)
   + (assume($z = 0$); skip)

client(v) $\triangleq$
  local x in
    $x := [v]$;
    push_back(v);
    L_{rx}: $[x] := 88$

ISL version

// client, inlining proc call
$x := [v]$;
(// push_back
  $y := [v]$;
  free(y);
  $y :=$ alloc( );
  $[v] := y$
+
  skip
)
$[x] := 88$

SepSIL version

more succint post

stronger guarantee: any state in pre can lead to error

$\langle\langle v \mapsto a * a \mapsto \_ * \text{true} \rangle\rangle$ rclient $\langle\langle x \not\mapsto \_ * \text{true} \rangle\rangle$

ISL derivation

$[v \mapsto a * a \mapsto -]$

`local` y, z `in`

$z := *$; // Havoc

$[ok: z{=}1 * v \mapsto a * a \mapsto -]$

(`assume`$(z \neq 0)$; // Assume

$[ok: z{=}1 * z{\neq}0 * v \mapsto a * a \mapsto -]$

L_{rv}: $y := [v]$; // Load

$[ok: z{=}1 * y{=}a * v \mapsto a * a \mapsto -]$

L_f: `free`(y); // Free

$[ok: z{=}1 * y{=}a * v \mapsto a * a \not\mapsto]$

$y :=$ `malloc()`; // Alloc1, Choice

$[ok: z{=}1 * v \mapsto a * a \not\mapsto * y \mapsto -]$

$[v] := y$; // Store

$[ok: z{=}1 * v \mapsto y * a \not\mapsto * y \mapsto -]$

) + $(\ldots)$ // Choice

$[ok: z{=}1 * v \mapsto y * a \not\mapsto * y \mapsto -]$

// Local

$[ok: \exists a'.\ v \mapsto a' * a' \mapsto - * a \not\mapsto]$

$[v \mapsto a * a \mapsto -]$

`local` x `in`

$x := [v]$; // Load

$[ok: x{=}a * v \mapsto a * a \mapsto -]$

`push_back`(v); // PB-Ok

$[ok: \exists a'.\ x{=}a * v \mapsto a' * a' \mapsto - * a \not\mapsto]$ // Cons

$[ok: \exists a'.\ x{=}a * v \mapsto a' * a' \mapsto - * x \not\mapsto]$

L_{rx}: $[x] := 88$; // StoreEr

$[er(L_{rx}): \exists a'.\ x{=}a * v \mapsto a' * a' \mapsto - * x \not\mapsto]$

// Local

$[er(L_{rx}): \exists a'.\ v \mapsto a' * a' \mapsto - * a \not\mapsto]$

SepSIL derivation

$\langle\!\langle v \mapsto a * a \mapsto \_ * \mathsf{true} \rangle\!\rangle \Rightarrow \langle\!\langle v \mapsto a * a \mapsto \_ * (a = a \lor a \not\mapsto\ ) * \mathsf{true} \rangle\!\rangle$

$x := [v];$ // Load + Frame

$\langle\!\langle v \mapsto a * a \mapsto \_ * (x = a \lor x \not\mapsto\ ) * \mathsf{true} \rangle\!\rangle \Rightarrow \langle\!\langle (v \mapsto a * a \mapsto \_ * (x = a \lor x \not\mapsto\ ) * \mathsf{true}) \lor (x \not\mapsto\ * \mathsf{true}) \rangle\!\rangle$

(// push_back: Choice

$\langle\!\langle v \mapsto a * a \mapsto \_ * (x = a \lor x \not\mapsto\ ) * \mathsf{true} \rangle\!\rangle$

$y := [v];$ // Load + Frame

$\langle\!\langle v \mapsto a * y \mapsto \_ * (x = y \lor x \not\mapsto\ ) * \mathsf{true} \rangle\!\rangle \Rightarrow \langle\!\langle v \mapsto \_ * y \mapsto \_ * (x = y \lor x \not\mapsto\ ) * \mathsf{true} \rangle\!\rangle$

$\mathsf{free}(y);$ // Free + Frame

$\langle\!\langle v \mapsto \_ * y \not\mapsto\ * (x = y \lor x \not\mapsto\ ) * \mathsf{true} \rangle\!\rangle \Rightarrow \langle\!\langle x \not\mapsto\ * v \mapsto \_ * \mathsf{emp} * \mathsf{true} \rangle\!\rangle$

$y := \mathsf{alloc}(\,);$ // Alloc + Frame

$\langle\!\langle x \not\mapsto\ * v \mapsto \_ * y \mapsto \_ * \mathsf{true} \rangle\!\rangle \Rightarrow \langle\!\langle x \not\mapsto\ * v \mapsto \_ * \mathsf{true} \rangle\!\rangle$

$[v] := y$ // Write + Frame

$\langle\!\langle x \not\mapsto\ * v \mapsto y * \mathsf{true} \rangle\!\rangle \Rightarrow \langle\!\langle x \not\mapsto\ * \mathsf{true} \rangle\!\rangle$

+

$\langle\!\langle x \not\mapsto\ * \mathsf{true} \rangle\!\rangle$ skip $\langle\!\langle x \not\mapsto\ * \mathsf{true} \rangle\!\rangle$ // Skip + Frame

)

$\langle\!\langle x \not\mapsto\ * \mathsf{true} \rangle\!\rangle$

$[x] := 88$

Correctness and completeness

Relational semantics

$$[\![\mathsf{skip}]\!] \triangleq \{(\sigma, \sigma)\}$$
$$[\![b?]\!] \triangleq \{(\sigma, \sigma) \mid \sigma = \langle s, h\rangle \wedge s \vDash b\}$$
$$[\![x := a]\!] \triangleq \{(\langle s, h\rangle, \langle s[x \mapsto [\![a]\!]s], h\rangle)\}$$

$$[\![x := [y]]\!] \triangleq \{(\langle s, h\rangle, \langle s[x \mapsto v], h\rangle) \mid v = h(s(y)) \in \mathbb{Z}\}$$

$$[\![[x] := y]\!] \triangleq \{(\langle s, h\rangle, \langle s, h[s(x) \mapsto s(y)]\rangle) \mid h(s(x)) \in \mathbb{Z}\}$$

$$[\![x := \mathsf{alloc}(\,)]\!] \triangleq \{(\langle s, h\rangle, \langle s[x \mapsto n], h[n \mapsto v]\rangle) \mid v \in \mathbb{Z} \wedge (n \notin \mathsf{dom}(h) \vee h(n) = \bot\,)\}$$

$$[\![\mathsf{free}(x)]\!] \triangleq \{(\langle s, h\rangle, \langle s, h[s(x) \mapsto \bot\,]\rangle) \mid h(s(x)) \in \mathbb{Z}\}$$

Actual rules of SepSIL

$$\frac{}{\langle\!\langle \mathbf{emp} \rangle\!\rangle\ \texttt{skip}\ \langle\!\langle \mathbf{emp} \rangle\!\rangle}\ \langle\!\langle \text{skip} \rangle\!\rangle$$

$$\frac{}{\langle\!\langle q[a/x] \rangle\!\rangle\ \texttt{x := a}\ \langle\!\langle q \rangle\!\rangle}\ \langle\!\langle \text{assign} \rangle\!\rangle$$

$$\frac{}{\langle\!\langle \mathbf{emp} \rangle\!\rangle\ \texttt{x := alloc()}\ \langle\!\langle x \mapsto v \rangle\!\rangle}\ \langle\!\langle \text{alloc1} \rangle\!\rangle$$

$$\frac{}{\langle\!\langle q \wedge b \rangle\!\rangle\ \texttt{b?}\ \langle\!\langle q \rangle\!\rangle}\ \langle\!\langle \text{assume} \rangle\!\rangle$$

$$\frac{}{\langle\!\langle \beta \not\mapsto \rangle\!\rangle\ \texttt{x := alloc()}\ \langle\!\langle x = \beta \wedge x \mapsto v \rangle\!\rangle}\ \langle\!\langle \text{alloc2} \rangle\!\rangle$$

$$\frac{}{\langle\!\langle x \mapsto - \rangle\!\rangle\ \texttt{free(x)}\ \langle\!\langle x \not\mapsto \rangle\!\rangle}\ \langle\!\langle \text{free} \rangle\!\rangle$$

$$\frac{x \notin \text{fv}(a)}{\langle\!\langle y \mapsto a * q[a/x] \rangle\!\rangle\ \texttt{x := [y]}\ \langle\!\langle y \mapsto a * q \rangle\!\rangle}\ \langle\!\langle \text{load} \rangle\!\rangle$$

$$\frac{}{\langle\!\langle x \mapsto - \rangle\!\rangle\ \texttt{[x] := y}\ \langle\!\langle x \mapsto y \rangle\!\rangle}\ \langle\!\langle \text{store} \rangle\!\rangle$$

$$\frac{\langle\!\langle p \rangle\!\rangle\ \mathrm{r}\ \langle\!\langle q \rangle\!\rangle \quad \text{fv}(t) \cap \text{mod}(\mathrm{r}) = \emptyset}{\langle\!\langle p * t \rangle\!\rangle\ \mathrm{r}\ \langle\!\langle q * t \rangle\!\rangle}\ \langle\!\langle \text{frame} \rangle\!\rangle$$

$$\frac{\langle\!\langle p \rangle\!\rangle\ \mathrm{r}\ \langle\!\langle q \rangle\!\rangle \quad x \notin \text{fv}(\mathrm{r})}{\langle\!\langle \exists x.p \rangle\!\rangle\ \mathrm{r}\ \langle\!\langle \exists x.q \rangle\!\rangle}\ \langle\!\langle \text{exists} \rangle\!\rangle$$

Correctness

Th. [*correctness*]
If $\langle\!\langle P \rangle\!\rangle \; r \; \langle\!\langle Q \rangle\!\rangle$ then $P \subseteq [\![\overleftarrow{r}]\!] Q$

Proof. By induction on the derivation.

(Relative) completeness

Th. [*completeness*]
Any valid triple $\langle\!\langle P \rangle\!\rangle \; r \; \langle\!\langle Q \rangle\!\rangle$ can be derived

Proof. See full paper.

Questions

Question 1

Which SepSIL triples are valid ?

$\langle\!\langle \text{emp} \rangle\!\rangle\ \text{free}(x)\ \langle\!\langle x \not\mapsto\ \rangle\!\rangle$ ✗

$\langle\!\langle x \not\mapsto\ \rangle\!\rangle\ \text{free}(x)\ \langle\!\langle x \not\mapsto\ \rangle\!\rangle$ ✗

$\langle\!\langle \text{false} \rangle\!\rangle\ \text{free}(x)\ \langle\!\langle \text{emp} \rangle\!\rangle$ ✓

$\langle\!\langle x \mapsto \_ \rangle\!\rangle\ \text{free}(x)\ \langle\!\langle \text{emp} \rangle\!\rangle$ ✗

Question 2

Transform the following C-like code in the syntax of SepSIL

$i := 0\,; q := *\,p\,;$ while $(q \neq \text{nil})$ do $\{\ q := *\,q\,; i := i + 1\ \}$

$i := 0\,;$

$q := [p]\,;$

$(\ (q \neq \text{nil?})\,; q := [q]\,; i := i + 1\ )^\star;$

$(q = \text{nil?})$

Exam question

Prove the SepSIL triple $\langle\!\langle p \mapsto \mathsf{nil} * \mathsf{true} \rangle\!\rangle \; c \; \langle\!\langle i = 0 \rangle\!\rangle$ where

$$c \triangleq i := 0 \,;\, q := *p \,;\, \text{while } (q \neq \text{nil}) \text{ do } \{\; q := *q \,;\, i := i + 1 \;\}$$