Program analysis: from proving correctness to proving incorrectness

Roberto Bruni, Roberta Gori
(University of Pisa)

Lecture #11

BISS 2024
March 11-15, 2024

Separation SIL

Ongoing work

Sufficient Incorrectness Logic: SIL and Separation SIL

FLAVIO ASCARI, Università di Pisa, Italy
ROBERTO BRUNI, Università di Pisa, Italy
ROBERTA GORI, Università di Pisa, Italy
FRANCESCO LOGOZZO, Meta Platforms, Inc., USA

Sound over-approximation methods have been proved effective for guaranteeing the absence of errors, but inevitably they produce false alarms that can hamper the programmers. Conversely, under-approximation methods are aimed at bug finding and are free from false alarms. We introduce Sufficient Incorrectness Logic (SIL), a new under-approximating, triple-based program logic to reason about program errors. SIL is designed to set apart the initial states leading to errors. We prove that SIL is correct and complete for a minimal set of rules, and we study additional rules that can facilitate program analyses. We formally compare SIL to existing triple-based program logics. Incorrectness Logic and SIL both perform under-approximations, but while the former exposes only true errors, the latter locates the set of initial states that lead to such errors. Hoare Logic performs over-approximations and as such cannot capture the set of initial states leading to errors in *nondeterministic* programs – for deterministic and terminating programs, Hoare Logic and SIL coincide. Finally, we instantiate SIL with Separation Logic formulae (Separation SIL) to handle pointers and dynamic allocation and we prove its correctness and, for loop-free programs, also its completeness. We argue that in some cases Separation SIL can yield more succinct postconditions and provide stronger guarantees than Incorrectness Separation Logic and can support effective backward reasoning.

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, Necessary Conditions, Outcome Logic

1 INTRODUCTION

Formal methods aim to provide tools for reasoning and establishing program guarantees. Historically, research in formal reasoning progressed from manual correctness proofs to effective, automatic methods that improve program reliability and security. In the late 60s, Floyd [1967] and Hoare [1969] independently introduced formal systems to reason about programs. In the 70s/early 80s, the focus was on mechanization, with the introduction of numerous techniques such as predicate transformers [Dijkstra 1975], Abstract Interpretation [Cousot and Cousot 1977], model checking [Clarke and Emerson 1981], type inference [Damas and Milner 1982] and mechanized program proofs [Coquand and Huet 1985]. Those seminal works, in conjunction with the development of automatic and semi-automatic theorem provers (e.g., [de Moura 2007]) brought impressive wins in proving program correctness of real-world applications. For instance, the Astrée abstract interpreter automatically proves the absence of runtime errors in millions of lines of safety-critical C [Blanchet et al. 2003], the SLAM model checker was used to check Windows drivers [Ball and Rajamani 2001], CompCert is a certified C compiler developed in Coq [Leroy 2009], and VCC uses the calculus of weakest precondition to verify safety properties of annotated Concurrent C programs [Cohen et al. 2009].

Despite the aforementioned successes, effective program correctness methods struggle to reach mainstream adoption. As program correctness is undecidable, all those methods *over-approximate* programs behaviours. Over-approximation guarantees soundness (if the program is proved to be

Authors' addresses: Flavio Ascari, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy, flavio.ascari@phd.unipi.it; Roberto Bruni, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy, roberto.bruni@unipi.it; Roberta Gori, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy, roberta.gori@unipi.it; Francesco LogozzoMeta Platforms, Inc., USA, logozzo@meta.com.

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

SepSIL = SIL + SL

SIL

$$\langle\!\langle P \rangle\!\rangle \; r \; \langle\!\langle Q \rangle\!\rangle$$

SL

$$\frac{\{P\} \; r \; \{Q\}}{\{P * R\} \; r \; \{Q * R\}}$$

SepSIL

$$\frac{\langle\!\langle P \rangle\!\rangle \; r \; \langle\!\langle Q \rangle\!\rangle}{\langle\!\langle P * R \rangle\!\rangle \; r \; \langle\!\langle Q * R \rangle\!\rangle}$$

Regular commands

$$
\begin{aligned}
r ::= \;& e \\
\mid \;& r_1 ; r_2 \\
\mid \;& r_1 + r_2 \\
\mid \;& r^{\star}
\end{aligned}
$$

regular command

atomic command

choice

Kleene star

$$
\begin{aligned}
e ::= \;& \mathsf{skip} \\
\mid \;& b? \\
\mid \;& x := a \\
\mid \;& x := [y] \quad \text{// read} \\
\mid \;& [x] := y \quad \text{// write} \\
\mid \;& x := \mathsf{alloc}(\,) \\
\mid \;& \mathsf{free}(x)
\end{aligned}
$$

simplified

Assertion language

assertion

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

Boolean and classical assertions

structural assertions

track deallocated locations

Local axioms: write

SL

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

ISL

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

SepSIL

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

weakest pre

Local axioms: read

SL

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

ISL

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

SepSIL

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

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

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

SepSIL

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

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

$$[v \mapsto a * a \mapsto -]\ \texttt{client}(v)\ [er(\mathrm{L}_{rx})\colon \exists a'.\ v \mapsto a' * a' \mapsto - * a \not\mapsto\ ]$$

```
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

$\texttt{push\_back}(v) \triangleq$
  $\texttt{local}\ z, y\ \texttt{in}$
    $z := *;$
    $(\texttt{assume}(z \neq 0);\ \mathrm{L}_{rv}\colon y := [v];$
     $\mathrm{L}_f\colon \texttt{free}(y);$
     $y := \texttt{malloc}();\ [v] := y)$
   $+\ (\texttt{assume}(z = 0);\ \texttt{skip})$

$\texttt{client}(v) \triangleq$
  $\texttt{local}\ x\ \texttt{in}$
    $x := [v];$
    $\texttt{push\_back}(v);$
    $\mathrm{L}_{rx}\colon [x] := 88$

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

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

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

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

) + (...) // 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

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

// LOCAL

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

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

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

$$\frac{}{\langle\!\langle q \wedge b \rangle\!\rangle\ \texttt{b?}\ \langle\!\langle q \rangle\!\rangle}\ \langle\!\langle\text{assume}\rangle\!\rangle \qquad \frac{}{\langle\!\langle x \mapsto - \rangle\!\rangle\ \texttt{free(x)}\ \langle\!\langle x \not\mapsto\ \rangle\!\rangle}\ \langle\!\langle\text{free}\rangle\!\rangle$$

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

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

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

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

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

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 ArXiV draft.

6.7 Relative completeness of Separation SIL

The proof system in Section 6.3 is not complete. To move towards completeness, we first limit the assertion language to the existential fragment of first-order logic:

$$\begin{aligned} \mathrm{Asl} \ni p, q, t ::= {} & \mathrm{false} \mid \mathrm{true} \mid p \wedge q \mid p \vee q \mid \exists x.p \mid a \asymp a \\ & \mid \mathbf{emp} \mid x \mapsto \mathsf{a} \mid x \not\mapsto \mid p * q \end{aligned}$$

We remove negation, so we don't include universal quantifiers and heap assertions must be positive. However, we argue that this is sufficient to find bugs: for instance, in the example in Section 6.5, we only used assertions from this subset.

With this limited assertion language, the proof system in Section 6.3 is complete for all atomic commands except `alloc`. To deal with `alloc`, we need the ability to refer to the specific memory location that was allocated. However, the naive solution to add a constraint $x = \alpha$ in the post of ⟪alloc⟫ makes the frame rule unsound: for instance, the following triple is not valid:

$$\langle\!\langle \mathbf{emp} * \alpha \mapsto - \rangle\!\rangle\ \texttt{x := alloc()}\ \langle\!\langle (x \mapsto - \wedge x = \alpha) * \alpha \mapsto - \rangle\!\rangle.$$

To recover the frame rule, just like ISL needs the deallocated assertion in the post [Raad et al. 2020, §3], we need a "will be allocated" assertion in the pre. To this end we use the $\not\mapsto$ assertion, and change the semantic model to only allocate a memory location that is explicitly $\perp$ instead of one not in the domain of the heap. We formalize this by letting $avail(l) \triangleq h(l) = \perp$ in Figure 13a, and

replacing the axiom ⟪alloc⟫ with

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

Soundness still holds for this different semantics. Moreover, we can prove relative completeness [Apt and Olderog 2019, §4.3] for loop-free programs:

Theorem 6.3 (Relative completeness for loop-free programs). *Suppose to have an oracle to prove implications between formulas in Asl. Let* $\mathsf{r} \in \mathsf{HRCmd}$ *be a regular command without* $*$ *and* $p, q \in Asl$ *such that* $[\![\overleftarrow{\mathsf{r}}]\!]\{\!|q|\!\} \supseteq \{\!|p|\!\}$. *Then the triple* $\langle\!\langle p \rangle\!\rangle\ \mathsf{r}\ \langle\!\langle q \rangle\!\rangle$ *is provable.*

The proof relies on the possibility to rewrite any q in an equivalent assertion of the form $\exists x_1. \cdots \exists x_n. \bigvee_{1 \leq i \leq k} q_i$ where all q_i are assertions involving atoms composed with $\wedge$ and $*$ only. This way, completeness is proved for such q_i first and then extended to the entire q thanks to rules ⟪disj⟫ and ⟪exists⟫. Notably, we show that the weakest (possible) precondition $[\![\overleftarrow{\mathsf{r}}]\!]\{\!|q|\!\}$ of loop-free programs is always expressible as an assertion $t \in \mathrm{Asl}$, namely $\{\!|t|\!\} = [\![\overleftarrow{\mathsf{r}}]\!]\{\!|q|\!\}$, and prove that the triple $\langle\!\langle t \rangle\!\rangle\ \mathsf{r}\ \langle\!\langle q \rangle\!\rangle$ can be derived. Then, by ⟪cons⟫, the theorem follows for any p that implies t.

Questions

Question 1

Which SepSIL triples are valid ?

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

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

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

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

Question 2

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

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

$i := 0\,;$

$q := [p]\,;$

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

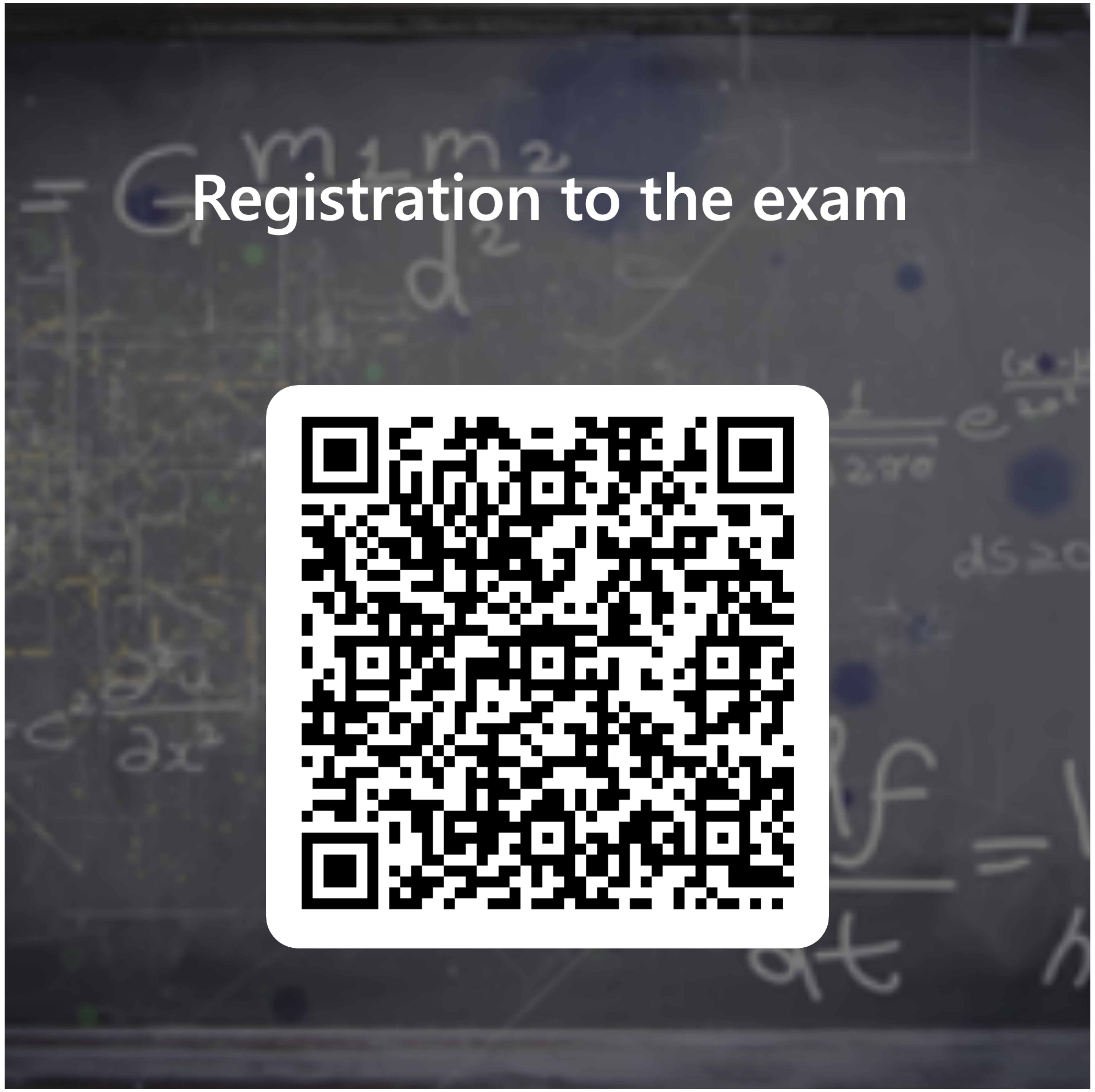
$(q = \text{nil?})$

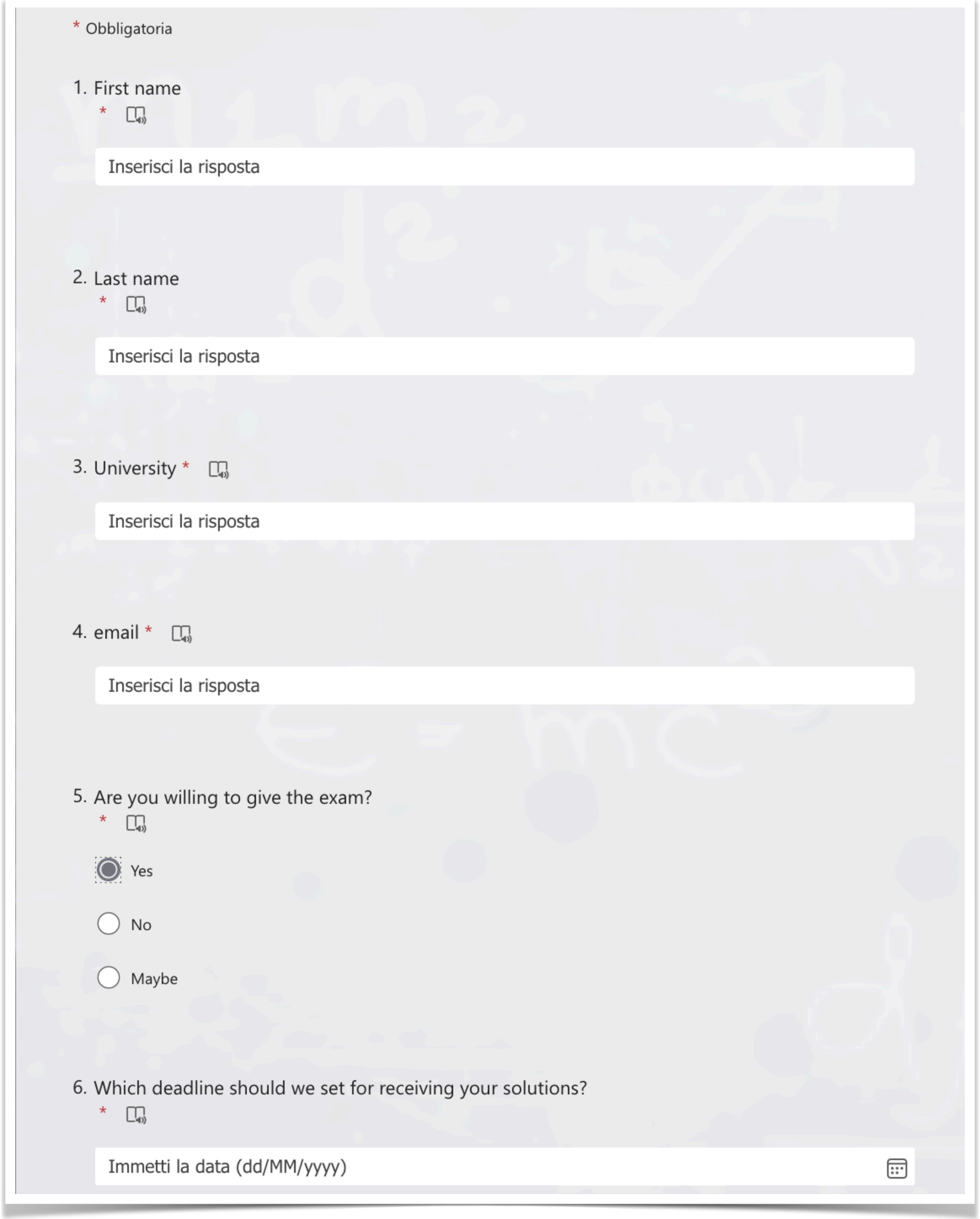
* Exam 16

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

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

Exam registration form





* Obbligatoria

1. First name *

Inserisci la risposta

2. Last name *

Inserisci la risposta

3. University *

Inserisci la risposta

4. email *

Inserisci la risposta

5. Are you willing to give the exam? *

- Yes
- No
- Maybe

6. Which deadline should we set for receiving your solutions? *

Immetti la data (dd/MM/yyyy)

Some references

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Isabel Garcia-Contreras, Dusko Pavlovic:
Abstract extensionality: on the properties of incomplete abstract interpretations.
Proc. ACM Program. Lang. 4(POPL): 28:1-28:28 (2020)

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Francesco Ranzato:
A Logic for Locally Complete Abstract Interpretations. LICS 2021: 1-13

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Francesco Ranzato:
A Correctness and Incorrectness Program Logic. J. ACM 70(2): 15:1-15:45 (2023)

Flavio Ascari, Roberto Bruni, Roberta Gori:
Limits and difficulties in the design of under-approximation abstract domains.
FoSSaCS 2022: 21-39

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Francesco Ranzato:
Abstract interpretation repair. PLDI 2022: 426-441

Roberto Bruni, Roberta Gori, Nicolas Manini:
Deciding Program Properties via Complete Abstractions on Bounded Domains.
SAS 2022: 175-200

Flavio Ascari, Roberto Bruni, Roberta Gori:
Logics for Extensional, Locally Complete Analysis via Domain Refinements.
ESOP 2023: 1-27

Flavio Ascari, Roberto Bruni, Roberta Gori, Francesco Logozzo:
Sufficient Incorrectness Logic: SIL and Separation SIL.
CoRR abs/2310.18156 (2023)

{many} (HL ; IL ; AI ; LCL ; NC ; SIL

\+

SL ; ISL ; SepSIL)* [thanks]