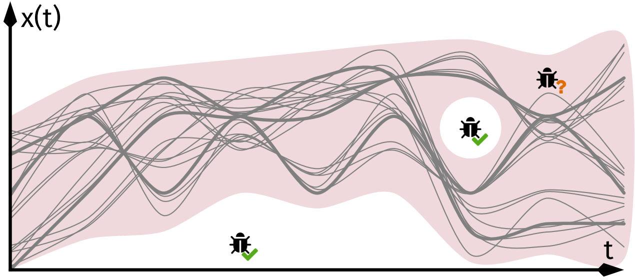


Abstract Interpretation

Software Quality Assurance - Static Code Analysis, II | Florian Sihler | December 11, 2024

Software Engineering
Programming Languages

universität ulm

Outline

1. The Why

The Why

```
public static void main(String[] args) {
    int a = 1;                              ≀ a ∈ {1} ∫
    double r = Math.random() * 10;          ≀ r ∈ [0..10) ∫
    if (r > 5) {
      a = 2;                                ≀ a ∈ {2} ∫
    }
    System.out.println(a);                  ≀ a ∈ {1,2} ∫  →  Valid? Ok? Safe?
}
```

- We want to proof, that a program satisfies certain properties

Origins

1949: First Checks
Turing [Tur49]

1953: Rice Theorem
Non-trivial Properties
are undecidable [Ric53]

Static Analysis

1967 & 69: Logical Foundation
Floyd [Flo67], Hoare [Hoa69]
But: No Automation

Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]

Origins

[Tur49; Ric53]

[Flo67; Hoa69]

Deductive Methods

- **1992**: Theorem Prover
  PVS, Owre et al. [ORS92]
- **2004**: Proof Asisstant
  Coq, Bertot et al. [BC04]

Model Checking

- **1986**: Foundations
  Clarke et al. [CES86]
- **2004**: Bounded MC
  Clarke et al. [CKL04]

Symbolic Execution

- **1974 & 75**: Foundations
  Boyer et al. [BEL75], King [Kin74]
- **2008**: Automation
  KLEE, Cadar et al. [CDE08]

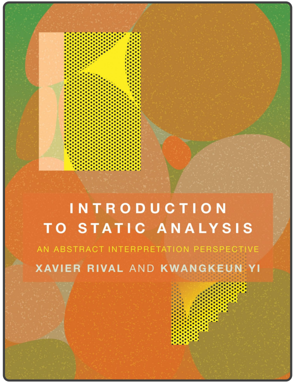
Abstract Interpretation

- **1977**: Fixpoints on Lattices
  Cousot and Cousot [CC77]
- **2004**: Automated Application
  Mauborgne [Mau04]

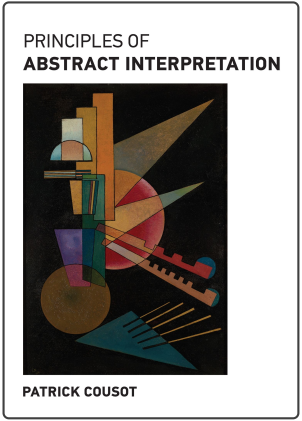
Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]

Recommended Resources

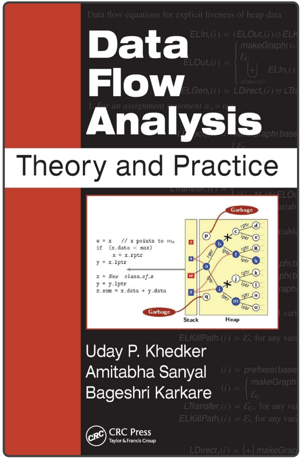
Using Analyses [RY20]



Formal Foundations [Cou21]



Dataflow Perspective [KSK09]



And for an overview: “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17]

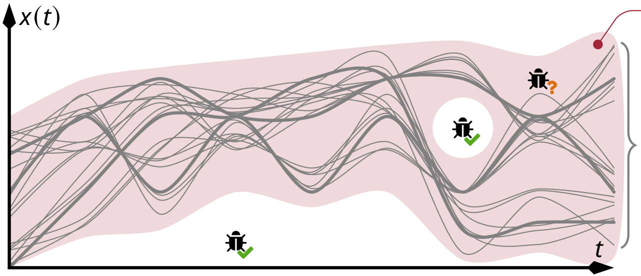
2. The How

Abstract Interpretation

- We want to proof interesting properties of programs
  - *Dataflow Properties*
    Liveness, Fainting, Reaching Definitions, …
  - *Safety Properties*
    No Null Dereference, No Division by Zero, …
  - *Numerical Properties*
    Signs, Intervals, Octagons, Polyhedra, …
  - …

“Principles of Abstract Interpretation” [Cou21, p. 722],“History of Abstract Interpretation” [GR22, pp. 37]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many

Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- … or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See “A casual introduction to Abstract Interpretation” [Cou12]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, 4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$

  universe ($\mathbb{U}$)

  $$\varnothing \subseteq \mathrm{P}_1 \subseteq \mathrm{P}_2 \subseteq \mathbb{U}$$

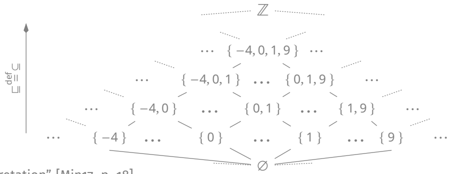
  strongest stronger weaker weakest

  $\forall x, y, z \in X : x \sqsubseteq y \land y \sqsubseteq z \implies x \sqsubseteq z$

  $\forall x \in X : x \sqsubseteq x$

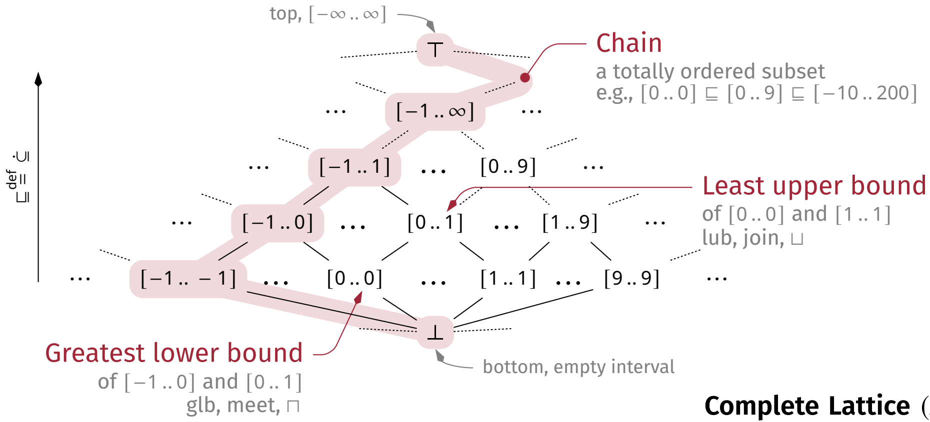
  $\forall x, y \in X : x \sqsubseteq y \land y \sqsubseteq x \implies x = y$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set
  $(\mathbb{Z}, \leq), \quad (\mathcal{P}(\mathbb{Z}), \subseteq), \quad \ldots$



“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Chains and Lattices

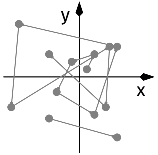


Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$

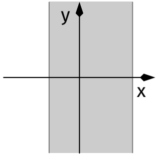
- $(X, \sqsubseteq)$ is a partial order
- $\forall A \subseteq X : \sqcup A$ and $\sqcap A$ exist
- $\perp / \top$ as smallest/largest element

“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

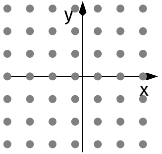
Abstract Domains Numerical



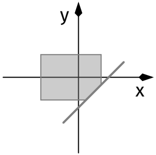
Collecting Semantics



Intervals $x \in [a \mathinner{..} b]$



Simple Congruences



Pentagons

- Octagons
- Ellipses
- Exponentials
- Signs

"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

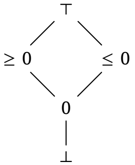
Sign Analysis

- We still have no program semantics, but we can try…

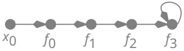
```
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

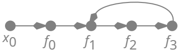
$\{\!\{\, a = 0 \,\}\!\}$
$\{\!\{\, b \geq 0 \,\}\!\}$
$\{\!\{\, c \geq 0 \quad (= 0 \ + \ \geq 0) \,\}\!\}$
$\{\!\{\, d = \top \quad (\geq 0 \ - \ \geq 0) \,\}\!\}$

- But how to handle control flow? Loops? Recursion?



Fixpoints

- For operators $f : X \to X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

reach a fixpoint, $f^p = f(f^p)$
  - 

reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$
  - 

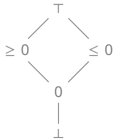
iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$ $\quad f : \mathbb{N} \to \mathbb{N}, f(x) = x + 1$
- If our function is monotonic, we can always find a fixpoint[Tar55] — for complete, nonempty lattices Tarski's Theorem
- Analyzing, e.g. loops, we "go up" the lattice until we reach a least fixpoint

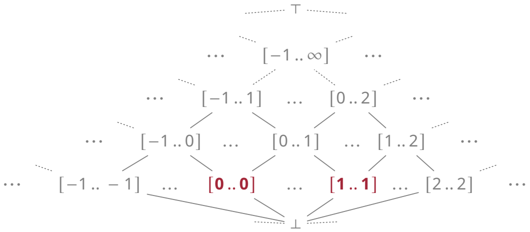
"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Interval Analysis, I

```
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

$\{\!\{\, x_0 \in [0\,..\,0] \,\}\!\}$
$\{\!\{\, \text{[pre]}\; x_1 \in [0\,..\,0] \,\}\!\}$
$\{\!\{\, \text{[in]}\; x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1]) \,\}\!\}$
$\{\!\{\, x_3 \in [1\,..\,1] \quad ([0\,..\,0] \oplus [1\,..\,1]) \,\}\!\}$





Interval Analysis, I

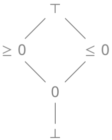
```
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

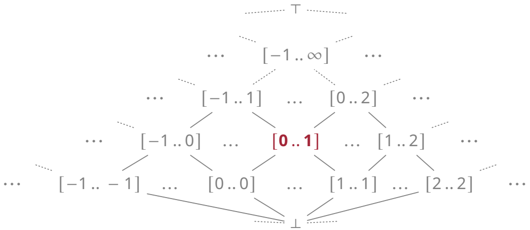
$\{\!\{\, x_0 \in [0\,..\,0] \,\}\!\}$

$\{\!\{$ [pre] $\boldsymbol{x_1 \in [0\,..\,1] \quad ([0\,..\,0] \cup [1\,..\,1])} \,\}\!\}$

$\{\!\{$ [in] $x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1]) \,\}\!\}$

$\{\!\{\, x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1]) \,\}\!\}$





Interval Analysis, I

```
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

Intervals

$\{\!\{\ x_0 \in [0 \mathinner{..} 0]\ \}\!\}$

$\{\!\{\ [\text{pre}]\ \boldsymbol{x_1 \in [0 \mathinner{..} 2]} \quad \boldsymbol{([0 \mathinner{..} 1] \cup [1 \mathinner{..} 2])}\ \}\!\}$

$\{\!\{\ [\text{in}]\ x_2 \in\ \}\!\}$

$\{\!\{\ x_3 \in\ \}\!\}$

$\{\!\{\ [\text{post}]\ x_4 \in [2 \mathinner{..} 2] \quad ([0 \mathinner{..} 2] \cap [2 \mathinner{..} \infty))\ \}\!\}$

Signs

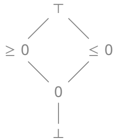
$\{\!\{\ x_0 = 0\ \}\!\}$

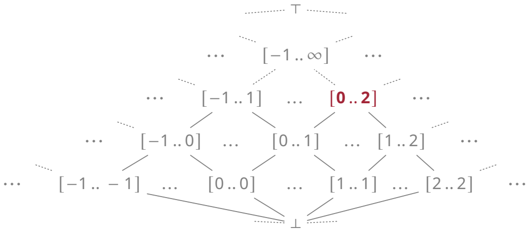
$\{\!\{\ [\text{pre}]\ x_1 \geq 0\ \}\!\}$

$\{\!\{\ [\text{in}]\ x_2 \geq 0\ \}\!\}$

$\{\!\{\ x_3 \geq 0\ \}\!\}$

$\{\!\{\ [\text{post}]\ x_4 \geq 0\ \}\!\}$





3. Semantics

Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

```
int x = 0;
```

Sequence

Numeric Constant $c \in \mathbb{I}$

Loop

```
while(x < 2) {
```

Comparison $\bowtie \in \{\leq, <, \ldots\}$

```
  x = x + 1;
```

Binary Expression

```
}
```

$$
\begin{array}{llll}
\mathit{stm} & ::= & V \leftarrow \mathit{expr} & (\text{assignment}, V \in \mathbb{V}) \\
 & | & \mathit{stm}_1;\ \mathit{stm}_2 & (\text{sequence}) \\
 & | & \textbf{while}(\mathit{cond})\ \{\ \mathit{stm}\ \} & (\text{loop}) \\
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & | & c & (\text{constant}, c \in \mathbb{I}) \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\}) \\
\mathit{cond} & ::= & b & (\text{boolean}, b \in \mathbb{B}) \\
 & | & \mathit{expr}_1 \bowtie \mathit{expr}_2 & (\text{comparison}, \bowtie \in \{\leq, <, \ldots\})
\end{array}
$$

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```
int x = 0;
while(x < 2) {
   x = x + 1;
}
```

$$
\begin{array}{llll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & \mid & c & (\text{constant}, c \in \mathbb{I}) \\
 & \mid & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

Variable | Integer Values

- We use an environment $\mathcal{E} \overset{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, \mathit{expr} \,]\!]\rho$

- Now we can define $\text{evalExpr}(\mathit{expr}, \mathit{env})$ for an environment $\mathit{env} \in \mathcal{E}$

$$
\begin{aligned}
\text{evalExpr}(V,\ \mathit{env}) &\overset{\text{def}}{=} \mathit{env}(V) \\
\text{evalExpr}(c,\ \mathit{env}) &\overset{\text{def}}{=} c \\
\text{evalExpr}(\mathit{expr}_1 + \mathit{expr}_2,\ \mathit{env}) &\overset{\text{def}}{=} \text{evalExpr}(\mathit{expr}_1,\ \mathit{env}) + \text{evalExpr}(\mathit{expr}_2,\ \mathit{env}) \\
&\ \ \vdots
\end{aligned}
$$

Value of $V \in \mathbb{V}$ in Environment env

$\mathbb{C}[\![\, \mathit{cond} \,]\!]\mathcal{D}$ $\mathbb{S}[\![\, \mathit{stm} \,]\!]\mathcal{D}$

- Additionally we can define $\text{evalCond}(\mathit{cond}, \mathit{envs})$ and $\text{evalStm}(\mathit{stm}, \mathit{envs})$

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$$\textbf{while}(\mathit{cond})\ \{\ \mathit{stm}\ \} \qquad F(X) \stackrel{\text{def}}{=} \mathit{Start} \cup \mathit{evalStm}(\mathit{stm},\ \mathit{evalCond}(\mathit{cond},\ X))$$

iterate to find the least fixpoint [Min17, p. 52]

Keep only states S with $\mathit{evalCond}(\neg \mathit{cond},\ S)$

There are alternatives (e.g., equation systems, [Cou21, part 7])

We achieve their abstract counterpart using
the same principles but for abstract domains!

Usually written as
$\mathbb{S}^\#, \mathbb{C}^\#, \mathbb{E}^\#, \ldots$

Interval Analysis, II

```
int x = 0;
while(x < 999999) {
  x = x + 1;
}
```

$\wr\, x_0 \in [0\,..\,0] \,\int$

$\wr$ [pre] $x_1 \in [0\,..\,2] \quad ([0\,..\,1] \cup [1\,..\,2]) \,\int \quad \nabla \implies x_1 \in [0\,..\,\infty)$

$\wr$ [in] $x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1]) \,\int$

$\wr\, x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1]) \,\int$

$\wr$ [post] $x_4 \in [999999\,..\,\infty) \quad ([0\,..\,\infty) \cap [999999\,..\,\infty)) \,\int$

- Fixpoint iteration can be *very* expensive, and may not stabilize
- *Widening* (∇) is crucial, computing an upper bound

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

4. Soundness and Completeness

Rice's Theorem

- We want to prove properties of programs (e.g., no overflow, shapes, …)
- However, thanks to Rice [Ric53] we know:

  Rice's theorem states that all nontrivial semantic properties of programs are undecidable. [Cou21, p. 100]

- We can not solve the halting problem
- We have to approximate the reality

The Confusion Matrix

E.g., do we claim there is an error?

		Prediction	
		Pos.	**Neg.**
Actual	**Pos.**	**(TP) True Positive**	(FN) False Negative
	Neg.	(FP) False Positive	**(TN) True Negative**

E.g., is there really an error?

- **Precision:** $\mathbf{TP}/(\mathbf{TP} + \mathrm{FP})$ (“how many false alarms”)
- **Recall:** $\mathbf{TP}/(\mathbf{TP} + \mathrm{FN})$ (“how many errors did we find”)

Soundness and Completeness

Soundness

- All properties we derive are true (but we may miss some)
- If we report bugs for violated properties, we produce no false negative

Completeness

- We are able to infer all interesting properties in the program
- If we report bugs for violated properties, we produce no false positive

Abstract interpretation soundly over-approximates the program semantics

5. Outlook

Outlook

- Domain transformers
  combine abstract domains[Min17, p. 149]
- Galois connections
  define the relationship between concrete and abstract domains[Cou21, p. 110]
- Corresponding to widening, narrowing
  refines approximations[Cou21, p. 395]
- Function calls
  require special handling[MJ12]
- Existing libraries allow for easy implementation
  LiSA[Fer+21], MOPSA[Jou+19], Apron[JM09]

References I

[Bal+18] Roberto Baldoni et al. “A Survey of Symbolic Execution Techniques”. In: *ACM Comput. Surv.* 51.3 (2018), 50:1–50:39. DOI: 10.1145/3182657. URL: https://doi.org/10.1145/3182657.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. ISBN: 978-3-642-05880-6. DOI: 10.1007/978-3-662-07964-5. URL: https://doi.org/10.1007/978-3-662-07964-5.

[BEL75] Robert S. Boyer, Bernard Elspas, and Karl N. Levitt. “SELECT - a formal system for testing and debugging programs by symbolic execution”. In: *Proceedings of the International Conference on Reliable Software 1975, Los Angeles, California, USA, April 21-23, 1975*. Ed. by Martin L. Shooman and Raymond T. Yeh. ACM, 1975, pp. 234–245. DOI: 10.1145/800027.808445. URL: https://doi.org/10.1145/800027.808445.

[Bir67] Garrett Birkhoff. “Lattice theory”. In: *Publications of AMS* (1967).

[CC77] Patrick Cousot and Radhia Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*. Ed. by Robert M. Graham, Michael A. Harrison, and Ravi Sethi. ACM, 1977, pp. 238–252. DOI: 10.1145/512950.512973. URL: https://doi.org/10.1145/512950.512973.

[CDE08] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs”. In: *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*. Ed. by Richard Draves and Robbert van Renesse. USENIX Association, 2008, pp. 209–224. URL: http://www.usenix.org/events/osdi08/tech/full%5C_papers/cadar/cadar.pdf.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. “Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications”. In: *ACM Trans. Program. Lang. Syst.* 8.2 (1986), pp. 244–263. DOI: 10.1145/5397.5399. URL: https://doi.org/10.1145/5397.5399.

[CKL04] Edmund Clarke, Daniel Kroening, and Flavio Lerda. “A tool for checking ANSI-C programs”. In: *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29-April 2, 2004. Proceedings 10*. Springer. 2004, pp. 168–176.

[Cou12] Patrick Cousot. “A casual introduction to Abstract Interpretation”. In: *CMACS Workshop on Systems Biology and Formals Methods (SBFM'12)* (2012). URL: https://pcousot.github.io/talks/PCousot-SBFM-2012-1-1.pdf (visited on 12/09/2024).

[Cou21] Patrick Cousot. “Principles of Abstract Interpretation”. In: (2021).

References II

[CZ11] Agostino Cortesi and Matteo Zanioli. “Widening and narrowing operators for abstract interpretation”. In: *Comput. Lang. Syst. Struct.* 37.1 (2011), pp. 24–42. DOI: 10.1016/J.CL.2010.09.001. URL: https://doi.org/10.1016/j.cl.2010.09.001.

[Fer+21] Pietro Ferrara et al. “Static analysis for dummies: experiencing LiSA”. In: *SOAP@PLDI 2021: Proceedings of the 10th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, Virtual Event, Canada, 22 June, 2021*. Ed. by Lisa Nguyen Quang Do and Caterina Urban. ACM, 2021, pp. 1–6. DOI: 10.1145/3460946.3464316. URL: https://doi.org/10.1145/3460946.3464316.

[Flo67] Robert W. Floyd. “Assigning Meanings to Programs”. In: *Proc. of the American Mathematical Society Symposia on Applied Mathematics*. Vol. 19. 1967, pp. 19–32.

[GR22] Roberto Giacobazzi and Francesco Ranzato. “History of Abstract Interpretation”. In: *IEEE Ann. Hist. Comput.* 44.2 (2022), pp. 33–43. DOI: 10.1109/MAHC.2021.3133136. URL: https://doi.org/10.1109/MAHC.2021.3133136.

[Hoa69] C. A. R. Hoare. “An Axiomatic Basis for Computer Programming”. In: *Commun. ACM* 12.10 (1969), pp. 576–580. DOI: 10.1145/363235.363259. URL: https://doi.org/10.1145/363235.363259.

[JM09] Bertrand Jeannet and Antoine Miné. “Apron: A Library of Numerical Abstract Domains for Static Analysis”. In: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*. Ed. by Ahmed Bouajjani and Oded Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 661–667. DOI: 10.1007/978-3-642-02658-4_52. URL: https://doi.org/10.1007/978-3-642-02658-4%5C_52.

[Jou+19] Matthieu Journault et al. “Combinations of Reusable Abstract Domains for a Multilingual Static Analyzer”. In: *Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers*. Ed. by Supratik Chakraborty and Jorge A. Navas. Vol. 12031. Lecture Notes in Computer Science. Springer, 2019, pp. 1–18. DOI: 10.1007/978-3-030-41600-3_1. URL: https://doi.org/10.1007/978-3-030-41600-3%5C_1.

[Kin74] James C. King. “A New Approach to Program Testing”. In: *Programming Methodology, 4th Informatik Symposium, IBM Germany, Wildbad, September 25-27, 1974*. Ed. by Clemens Hackl. Vol. 23. Lecture Notes in Computer Science. Springer, 1974, pp. 278–290. DOI: 10.1007/3-540-07131-8_30. URL: https://doi.org/10.1007/3-540-07131-8%5C_30.

[Kle52] Stephen Cole Kleene. “Introduction to metamathematics”. In: (1952).

[KSK09] Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. 1st. USA: CRC Press, Inc., 2009. ISBN: 0849328802.

[LF08] Francesco Logozzo and Manuel Fähndrich. “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses”. In: *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*. Ed. by Roger L. Wainwright and Hisham Haddad. ACM, 2008, pp. 184–188. DOI: 10.1145/1363686.1363736. URL: https://doi.org/10.1145/1363686.1363736.

References III

[Mau04] Laurent Mauborgne. “Astrée: verification of absence of run-time error”. In: *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*. Ed. by René Jacquart. Vol. 156. IFIP. Kluwer/Springer, 2004, pp. 385–392. DOI: 10.1007/978-1-4020-8157-6_30. URL: https://doi.org/10.1007/978-1-4020-8157-6%5C_30.

[Min17] Antoine Miné. “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation”. In: *Found. Trends Program. Lang.* 4.3-4 (2017), pp. 120–372. DOI: 10.1561/2500000034. URL: https://doi.org/10.1561/2500000034.

[MJ12] Jan Midtgaard and Thomas P. Jensen. “Control-flow analysis of function calls and returns by abstract interpretation”. In: *Inf. Comput.* 211 (2012), pp. 49–76. DOI: 10.1016/J.IC.2011.11.005. URL: https://doi.org/10.1016/j.ic.2011.11.005.

[ORS92] Sam Owre, John M. Rushby, and Natarajan Shankar. “PVS: A Prototype Verification System”. In: *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*. Ed. by Deepak Kapur. Vol. 607. Lecture Notes in Computer Science. Springer, 1992, pp. 748–752. DOI: 10.1007/3-540-55602-8_217. URL: https://doi.org/10.1007/3-540-55602-8%5C_217.

[Ric53] Henry Gordon Rice. “Classes of recursively enumerable sets and their decision problems”. In: *Transactions of the American Mathematical society* 74.2 (1953), pp. 358–366.

[RY20] Xavier Rival and Kwangkeun Yi. “Introduction to Static Analysis: An Abstract Interpretation Perspective”. In: (2020).

[Tar55] Alfred Tarski. “A lattice-theoretical fixpoint theorem and its applications.”. In: (1955).

[Tur49] Alan Turing. “Checking a large routine”. In: *Report of a Conference on High Speed Automatic Calculating Machines*. 1949, pp. 67–69.