

Abstract Interpretation

Software Quality Assurance — Static Code Analysis, II | Florian Sihler | December 10, 2025

Outline

1. The Why

2. The How

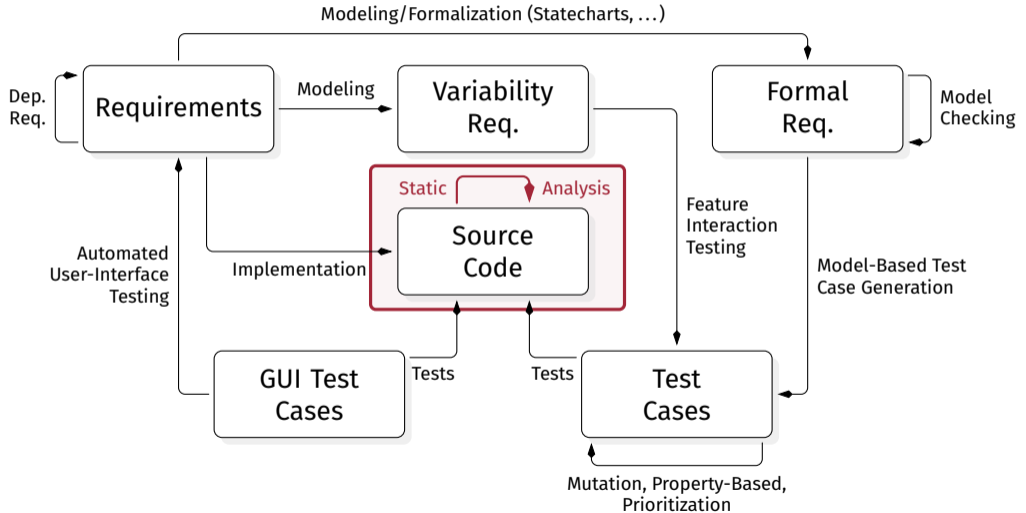
3. Semantics

4. Outlook and Comments

1. The Why

Why am I even here?

Embedding a Landscape





What is static analysis?

Discover *syntactic/semantic properties* of programs
without running them.^[RY20]

Today, we learn how to...

- describe semantic properties
- compare, refine, and combine properties
- describe and map language semantics to properties
- deal with the cost of abstraction (and the fun)



The Why

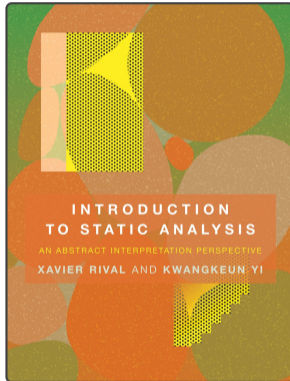
```
public static void main(String[] args) {  
    int a = 1;           { a ∈ {1} }  
    double r = Math.random() * 10; { r ∈ [0..10) }  
    if (r > 5) {  
        a = 2;           { a ∈ {2} }  
    }  
    System.out.println(1 / a); { a ∈ {1,2} } → Valid? Ok? Safe?  
}
```

java

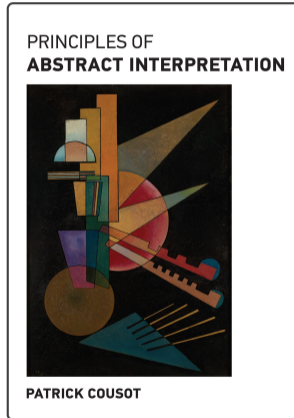
- We want to proof, that a program satisfies certain properties
- Abstract Interpretation is one (/the) technique to do so

Recommended Resources

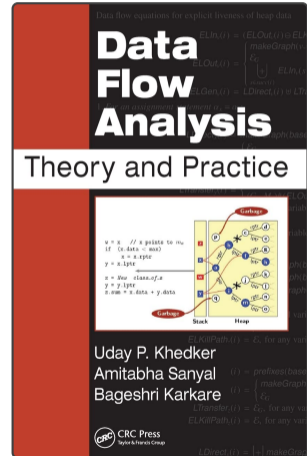
Using Analyses [RY20]



Formal Foundations [Cou21]



Dataflow Perspective [KSK09]



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

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

Gimme properties, gimme abstractions!

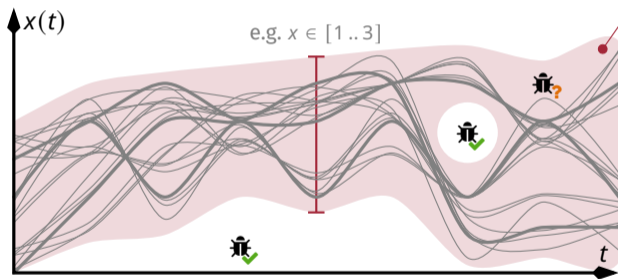
Abstract Interpretation



Radhia Cousot (1947–2014)
Patrick Cousot



Patrick Cousot (1948)
Personal Website



(Trace) Abstraction^[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

Collecting Semantics^[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, ...)
- ... or very expensive (► *dynamic*)
- Abstract Interpretation to the rescue

Terminology

- **Property** — Set of states/traces that satisfy that property

Even integers: $P = \{z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k\} = \{0, 2, -2, 4, -4, 6, \dots\} \subseteq \mathcal{P}(\mathbb{Z})$



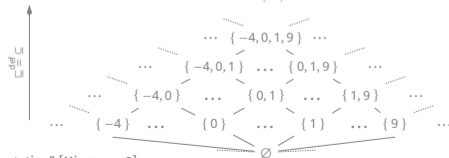
$$\forall x, y, z \in X : x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$$

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

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

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

Hasse Diagram of
 $(\mathcal{P}(\mathbb{Z}), \subseteq)$



A Task In-Between



Helmut Hasse (1898–1979)

© 2.0 Oberwolfach
Photo Collection

Task:

Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.

Draw its Hasse diagram.

Indicate the greatest and smallest element.

Hasse Diagram:

A directed graph, with edges from a to b ($a \neq b$) indicating that $a \subseteq b$ without a c with $a \subseteq c \subseteq b$.

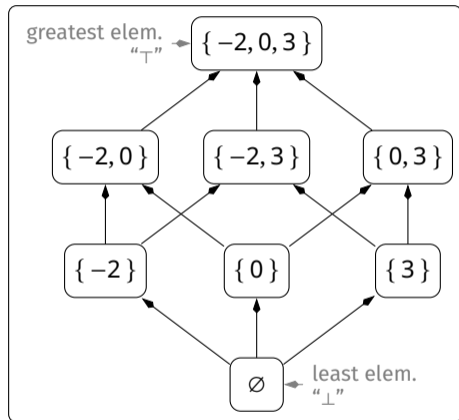
A Task In-Between

Task:

Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:

A directed graph, with edges from a to b ($a \neq b$) indicating that $a \subseteq b$ without a c with $a \subseteq c \subseteq b$.

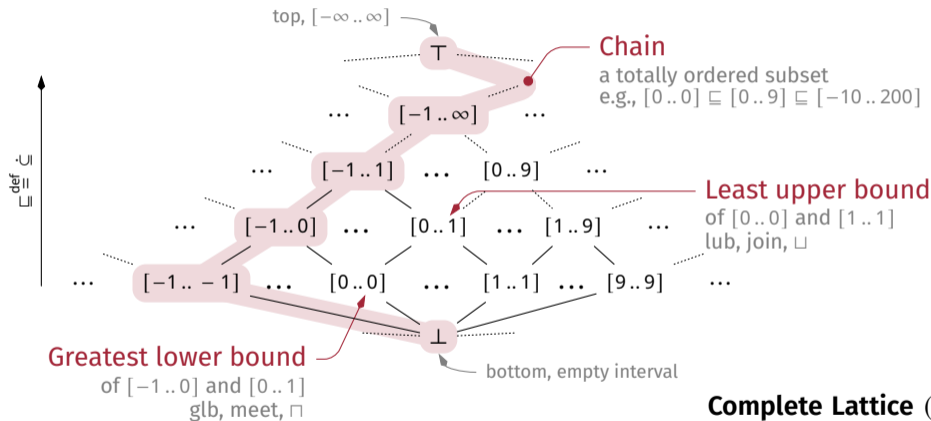


Hasse Diagram of $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$

Posets, Lattices, and Chains



Garrett Birkhoff (1911–96)
Paul Halmos



“Lattice elements (e.g. $[0 .. 1]$) define — per variable — the abstract state (the abstraction) at a given time or program point!”

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

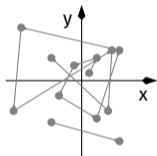
Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$

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

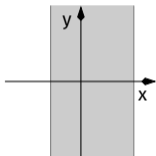
Abstract Domains

Pick Your Favorite Lattice!

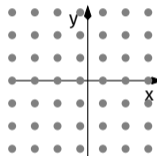
Numerical



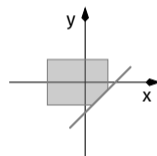
Collecting Semantics



Intervals $x \in [a..b]$
 $y \in [-\infty.. \infty]$



Simple Congruences



Pentagons

- Octagons

- Ellipses

- Exponentials

- Signs

Sign Analysis

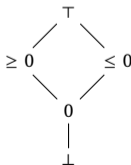
Simple Sign Domain

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

<code>int a = 0;</code>	$\{ a = 0 \}$
<code>int b = 12;</code>	$\{ b \geq 0 \}$
<code>int c = a + b;</code>	$\{ c \geq 0 \quad (= 0 + \geq 0) \}$
<code>int d = c - b;</code>	$\{ d = \top \quad (\geq 0 - \geq 0) \}$

java

- But, how do we know that `int a = 0;` implies $\{ a = 0 \}$?
 - Language Semantics (we get to those later)
 - Galois Connections



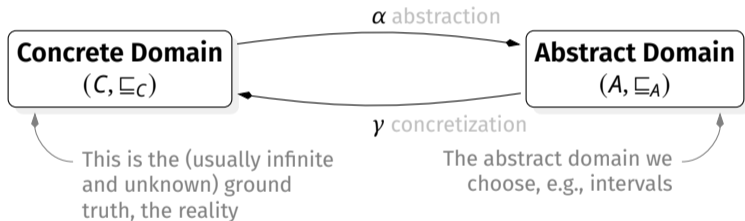
Galois Connection — Linking Posets [Cou21, p. 110] (small detour)



Évariste Galois (1811–32)
Public Domain
died in a duel for love

How do we know that `int a = 0;` implies $\{a = 0\}$?

A tale of two lattices...



Beware, beware, conversions may be lossy!

- $\forall x \forall y : \alpha(x) \sqsubseteq_A y \iff x \sqsubseteq_C \gamma(y)$

- $\forall x \forall y : x \sqsubseteq_C \gamma(\alpha(x)) \text{ and } \alpha(\gamma(y)) \sqsubseteq_A y$

"We can abstract $x \in \{1, 2\}$ with $\alpha(x) = [1 .. 2]$ or $\alpha(x) = [-5 .. 42]$, but *not* (e.g.) with $\alpha(x) = [1 .. 1]!$ "

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
 - $\alpha(u) =$
 - $\alpha(u + 1) =$
 - $\alpha(u * 0) =$
 - $\alpha(\sin(u)) =$
- What is the best concretization of $[-5..5]$? What do we loose?

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
 - $\alpha(u) = [-\infty .. \infty]$
 - $\alpha(u + 1) = [-\infty .. \infty]$
 - $\alpha(u * 0) = [0 .. 0]$
 - $\alpha(\sin(u)) = [-1 .. 1]$
- What is the best concretization of $[-5 .. 5]$? What do we loose?
 $\gamma([-5 .. 5]) = \{-5, -4, \dots, 4, 5\}$. We loose...
 - Exact values and distribution information
 - Relational information (e.g., $x = u; y = x + 1;$)
 - Potential sequences (e.g., **for**($i=0; i<n; i++$) ...)
- But, how do we handle loops? Recursion?

Fixpoints

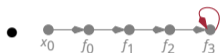


Alfred Tarski (1901–83)

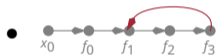
© 2.0 Oberwolfach

Photo Collection

- For operators $f : X \rightarrow X$ 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, $f^{p+\ell} = f^p, \ell > 0$

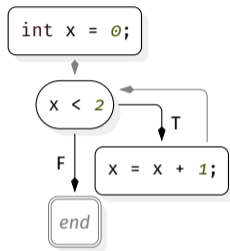


iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$

$f : \mathbb{N} \rightarrow \mathbb{N}, f(x) = x + 1$

- 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
- You may know fixpoints from tools like \LaTeX or the λ -calculus

Interval Analysis, I (the intuitive approach)



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

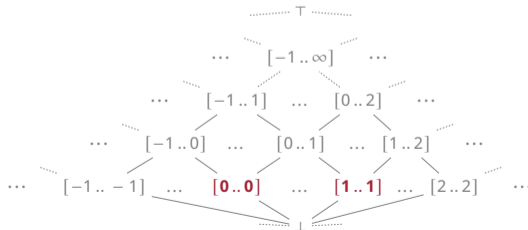
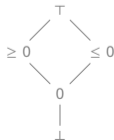
$\{ [\text{pre}] x_1 \in [0 .. 0] \}$

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

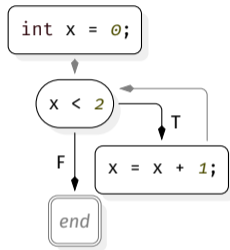
if we are inside the loop we know that $x < 2$ holds!

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

we have to know how to add intervals here



Interval Analysis, I (the intuitive approach)



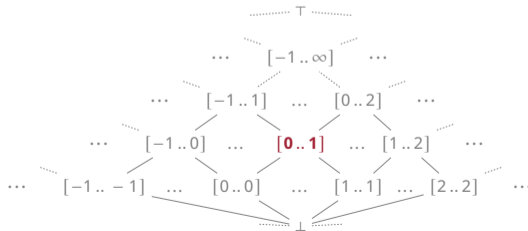
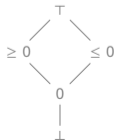
$\{ x_0 \in [0 .. 0] \}$

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

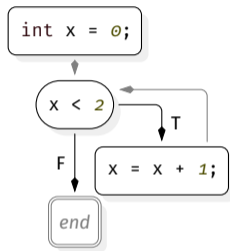
$\{ [\text{in}] x_2 \in [0 .. 1] \quad ([0 .. 1] \cap (-\infty .. 1]) \}$

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

we join on loops because, at this point x can have any of the values



Interval Analysis, I (the intuitive approach)



Intervals

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

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

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

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

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

after the loop we know that $\neg(x < 2) = x \geq 2$ holds!

Signs

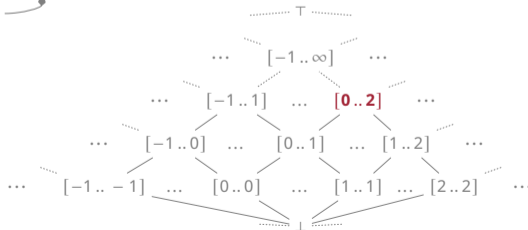
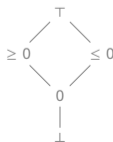
$$\{ x_0 = 0 \}$$

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

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

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

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



3. Semantics

What does my program mean?

Semantics

Program Syntax (simplified)

Variable $v \in \mathbb{V}$
Assignment
Sequence
Numeric Constant $c \in \mathbb{I}$
Loop
Comparison $\bowtie \in \{\leq, <, \dots\}$
Binary Expression

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

<i>stm</i>	$::=$	$V \leftarrow \text{expr}$	(assignment, $V \in \mathbb{V}$)	java
	$ $	$\text{stm}_1; \text{stm}_2$	(sequence)	
	$ $	while (<i>cond</i>) { <i>stm</i> }	(loop)	
<i>expr</i>	$::=$	V	(variable, $V \in \mathbb{V}$)	
	$ $	c	(constant, $c \in \mathbb{I}$)	
	$ $	$\text{expr}_1 \diamond \text{expr}_2$	(bin. expr., $\diamond \in \{+, -, \dots\}$)	
<i>cond</i>	$::=$	b	(boolean, $b \in \mathbb{B}$)	
	$ $	$\text{expr}_1 \bowtie \text{expr}_2$	(comparison, $\bowtie \in \{\leq, <, \dots\}$)	

Atomic Expression Semantics

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

java

$expr ::= V$ (variable, $V \in \mathbb{V}$)
 $| c$ (constant, $c \in \mathbb{I}$)
 $| expr_1 \diamond expr_2$ (bin. expr., $\diamond \in \{+, -, \dots\}$)

Variable

Integer Values

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

Usually written as $\mathbb{E} \llbracket expr \rrbracket \rho$

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

$\text{evalExpr}(V, env) \stackrel{\text{def}}{=} env(V)$ ← Value of $V \in \mathbb{V}$ in Environment env
 $\text{evalExpr}(c, env) \stackrel{\text{def}}{=} c$
 $\text{evalExpr}(expr_1 + expr_2, env) \stackrel{\text{def}}{=} \text{evalExpr}(expr_1, env) + \text{evalExpr}(expr_2, env)$
 \vdots

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

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

$\mathbb{C} \llbracket cond \rrbracket \mathcal{D}$

$\mathbb{S} \llbracket stm \rrbracket \mathcal{D}$

Denotational Semantics

while loops

Suppose we start the loop with states *Start*

$$\text{while}(\textcolor{red}{cond}) \{ \textcolor{red}{stm} \} \quad \begin{array}{l} \downarrow \\ F(X) \stackrel{\text{def}}{=} \text{Start} \cup \text{evalStm}(\textcolor{red}{stm}, \text{evalCond}(\textcolor{red}{cond}, X)) \\ \downarrow \end{array} \quad \begin{array}{l} \text{iterate to find the least fixpoint [Min17, p. 52]} \\ \text{Try to map this to the Interval Analysis, I example!} \end{array}$$

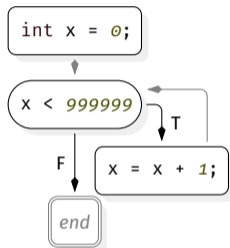
Keep only states *S* with $\text{evalCond}(\neg \textcolor{red}{cond}, S)$

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 $S^\#, C^\#, E^\#, \dots$

Interval Analysis, II



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

$$\{ [\text{pre}] x_1 \in [0..2] \quad ([0..1] \cup [1..2]) \} \nabla \implies x_1 \in [0..\infty)$$

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

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

$$\{ [\text{post}] x_4 \in [999999..\infty) \quad ([0..\infty) \cap [999999..\infty)) \}$$

- Fixpoint iteration can be *very* expensive, and may not stabilize
- *Widening* (∇) is crucial, computing an upper bound
(We only need widening if the lattice has an infinite ascending chain!)

Let's Bring it All Together

Sign Analysis



I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
drawing the hasse diagram is enough for us here

2. Define the abstract semantics on the following language!

$expr ::= c$ (constant, $c \in \mathbb{R}$)

| $expr_1 + expr_2$ (addition)

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

4. Do we need widening? If so, define ∇ !

Let's Bring it All Together

Sign Analysis



I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
drawing the hasse diagram is enough for us here

$$\mathcal{L} = (x = \{\perp, 0, \leq 0, \geq 0, \top\}, \leq = \{(\perp, 0), (\perp, \leq 0), \dots\}, \sqcup = \{(\perp, x) \mapsto x, (0, \leq 0) \mapsto \leq 0, \dots\}, \\ \sqcap = \{(\top, x) \mapsto x, (0, \geq 0) \mapsto 0, \dots\}, \perp, \top)$$

$$\top \left\langle \begin{array}{l} \geq 0 \\ \leq 0 \end{array} \right\rangle 0 \text{ --- } \perp$$

2. Define the abstract semantics on the following language!

expr ::= *c* (constant, $c \in \mathbb{R}$)

short for $\text{evalExpr}(\text{expr}, \rho = \text{env})$

| *expr*₁ + *expr*₂ (addition)

$$\llbracket c \rrbracket \rho \stackrel{\text{def}}{=} \begin{cases} \geq 0 & \text{if } c > 0 \\ 0 & \text{if } c = 0 \\ \leq 0 & \text{if } c < 0 \end{cases}$$

$$\llbracket a + b \rrbracket \rho \stackrel{\text{def}}{=} \llbracket a \rrbracket \rho \oplus \llbracket b \rrbracket \rho$$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

$$\gamma(0) = \{0\}, \quad \gamma(\leq 0) = \{n \in \mathbb{Z} \mid n \leq 0\}$$

4. Do we need widening? If so, define ∇ !

We do not need widening here as our lattice is finite

and our semantics do not introduce new elements. (We will concretize this next week.)

\oplus	\perp	0	\leq	\geq	\top
\perp	\perp	\perp	\perp	\perp	\perp
0		0	\leq	\geq	\top
\leq			\leq	\top	\top
\geq				\geq	\top
\top					\top

$$x \oplus y = y \oplus x$$

4. Outlook and Comments

This is incredible, I need more!

The Domains We Built...

- We know a handful of important concepts for our domains:
 - Lattices (poset, with join \sqcup , meet \sqcap , bottom \perp , and top \top)
 - Program Semantics ($\mathbb{S}[\![stm]\!] \mathcal{D}, \dots$, e.g., $\mathbb{E}[\![a + b]\!] \rho = \mathbb{E}[\![a]\!] \rho \oplus \mathbb{E}[\![b]\!] \rho$)
 - Fixpoint Iterations (to interpret loops, recursion, ...)
 - Widening (to ensure termination with infinite chains)
 - Galois Connections (to relate concrete and abstract domains)
- With abstract interpretation we interpret programs *over* these domains!
- However, we have only looked at single variables and single domains so far!

Combining Domains: Products



Saunders Mac Lane (1909–2005)

© 2.0 Oberwolfach
Photo Collection



Samuel Eilenberg (1913–1998)

© 2.0 Oberwolfach
Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$
- For example, let's suppose we want to track the shape of tables!
We define three domains!

- $\mathcal{N} \stackrel{\text{def}}{=} \mathcal{P}(\Sigma^*)$ (set of column names)
- $\mathcal{R} = \mathcal{C} \stackrel{\text{def}}{=} \{ [a..b] \mid a, b \in \mathbb{N}_0 \cup \{\infty\}, a \leq b \}$ (# of rows/columns)
- Then we can define the domain: $\mathcal{DF} \stackrel{\text{def}}{=} \mathcal{N} \times \mathcal{R} \times \mathcal{C}$

column names

#rows	id	name	score

#columns			



[Ger25] Oliver Gerstl. Tracking the shape of data frames in R programs using abstract interpretation (Ulm University, 2025)

Category theory is amazing!

Outlook

- Domain transformers
combine abstract domains [Min17, p. 149]
- Galois connections (offer so much more)
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 [JMo9]
- There are other ways to define semantics
e.g., small-step, big-step [Ci013]

Domains can also capture relations between variables (e.g. *polyhedra*), their provenance, and much more!

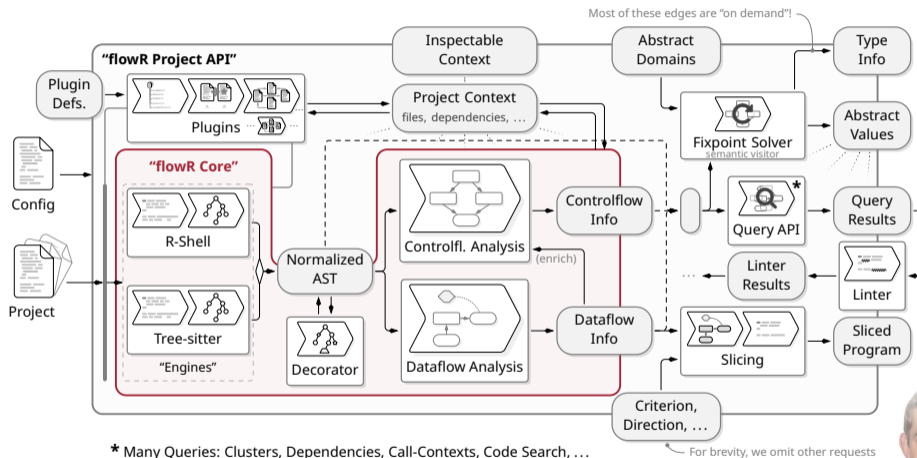
However, this implies trade-offs which we discuss next time.



Back to the Questions

1. How would you capture what a *property* is? ✓
2. How would you phrase that one property is “better” than another? ✓
3. For what operations would you *not* use a control-flow graph?
4. Why can't there be a fully automatic, sound, and complete static analyzer for general programs?
5. What (big) additional challenges do you see in the real-world?

And... Back to the Real World?



In case you like/are intrigued by what you see, join the horde: florian.sihler@uni-ulm.de



Bibliography

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>.
- [BCo4] 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>.
- [CDEo8] 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>.
- [Cio13] Ștefan Ciobâcă. “From Small-Step Semantics to Big-Step Semantics, Automatically”. In: *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*. Ed. by Einar Broch Johnsen and Luigia Petre. Vol. 7940. Lecture Notes in Computer Science. Springer, 2013, pp. 347–361. DOI: 10.1007/978-3-642-38613-8_24. URL: https://doi.org/10.1007/978-3-642-38613-8_24.
- [CKLo4] 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.

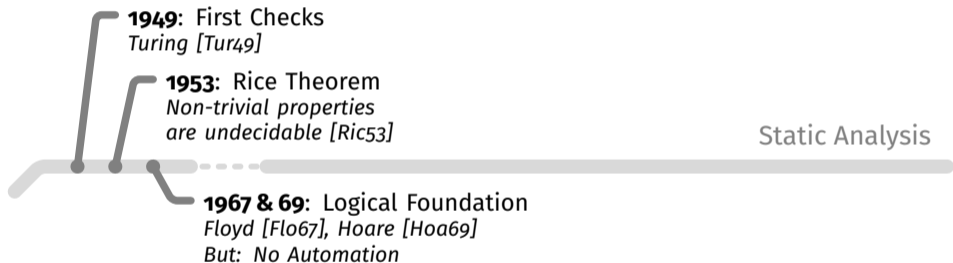
References II

- [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).
- [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.
- [Ger25] Oliver Gerstl. *Tracking the shape of data frames in R programs using abstract interpretation*. en. Master's Thesis. 2025. DOI: 10.18725/OPARU-58621. URL: <https://oparu.uni-ulm.de/handle/123456789/58696>.
- [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_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_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_30.

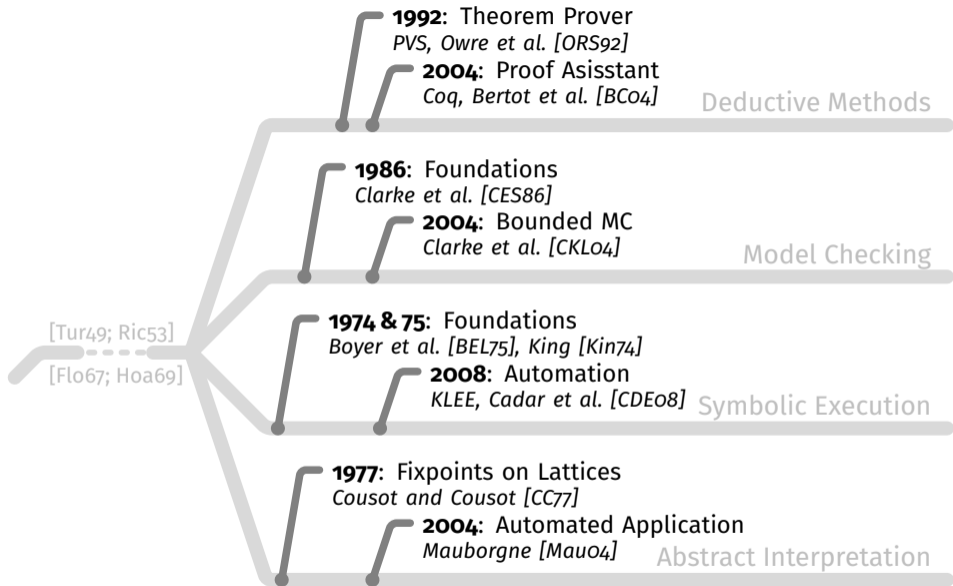
References III

- [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.
- [LFO8] 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>.
- [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_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: 10.1561/25000000034. URL: <https://doi.org/10.1561/25000000034>.
- [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_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.

A Little Bit of History



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]



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]