Introduction to Abstract Interpretation

Pierre Talbot
`pierre.talbot@uni.lu`

23rd April 2024

University of Luxembourg

UNIVERSITÉ DU LUXEMBOURG

Who am I?

- 2014–2018: Ph.D., Sorbonne University, Paris (supervised by Prof. Carlos Agon).
    - *Spacetime Programming: A Synchronous Language for Constraint Search*
- 2018–2019: Postdoctoral researcher, University of Nantes (Prof. Monfroy & Truchet).
    - *Abstract Domains for Constraint Programming*.
- 2020–2023: Postdoctoral researcher, University of Luxembourg (Prof. Bouvry).
    - *A Lattice-Based Approach for GPU Programming.*
- 2024–: Research scientist, University of Luxembourg.
    - *Abstract Constraint Reasoning.*

What am I going to talk about?

- **Abstract Interpretation**: A formal program verification method.
- **Research Plan**: Research directions among combinatorial optimization, parallel programming and abstract interpretation.

Abstract Interpretation

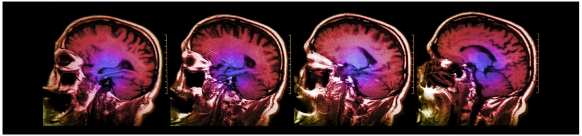
Costly Software Accidents

In 1996, the explosion of Ariane 501, which took ten years and $7 billion to build.

Costly Software Accidents

Bug in fMRI software calls 15 years of research into question

Popular pieces of software for fMRI were found to have false positive rates up to 70%



TRENDING NOW

Three of the most popular pieces of software for fMRI – SPM, FSL and AFNI – were all found to have false positive rates of up to 70 per cent. These findings could invalidate "up to 40,000 papers", researchers claim.

Costly Software Accidents

PRWeek

home

How did you survive the Great Twitter Outage of 2016?

Well, that's awkward. #twitterdown was the top trending topic in the US late Tuesday morning.

What Can We Do?

Nothing but it would be irresponsable.

COMMUNICATIONS OF THE ACM

OPINION

Computing Applications

Responsible Programming

By Vinton G. Cerf

Posted Jul 1 2014

"People who write software should have a clear sense of responsibility for its reliable operation and resistance to compromise and error."[1]

[1] `https://cacm.acm.org/opinion/responsible-programming/`

Know Your Limits...

OK, we should verify software but we should also know our limits...

Undecidability

By Rice's theorem, a static analyzer cannot have all the following properties:

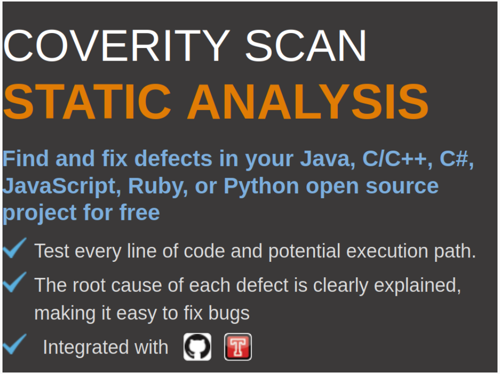
- **General**: works on Turing-complete program.
- **Automated**: does not require human intervention.
- **Sound**: report all bugs.
- **Complete**: all bugs reported are true bugs.

General, semi-automated, complete but unsound (e.g., unit testing).

"Program testing can be used to show the presence of bugs, but never to show their absence!" (Edsger Dijkstra).

Bug Finding

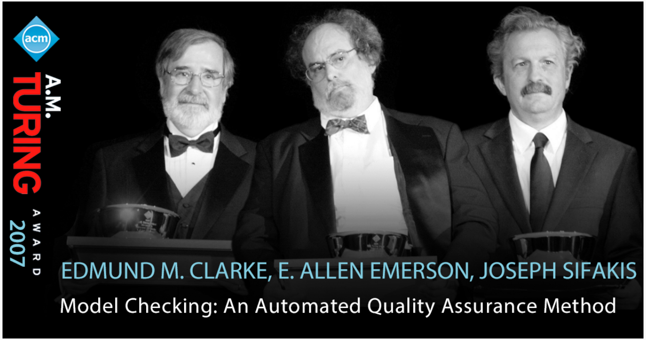
General, automated, **incomplete and unsound** (e.g. Coverity, CodeSonar).



> Additionally, Synopsys's implementation of static analysis can follow all the possible paths of execution through source code (including interprocedurally) and find defects and vulnerabilities caused by the conjunction of statements that are not errors independent of each other.

Model-Checking

Non-general (finite state model), semi-automated, complete and sound.

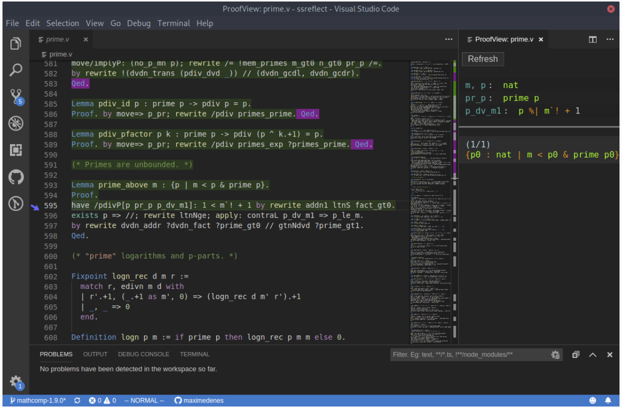


Theorem Proving

General, non-automated, complete and sound (e.g., Lean, Coq).

But require human intervention to provide invariants (time consuming and require expertise).

Success story: Compcert, certified C compiler.

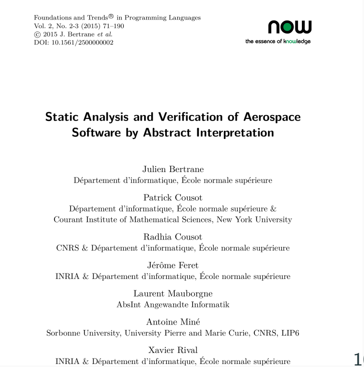


Abstract Interpretation

General, automated, incomplete and sound.

Success story: Astrée, prove absence of bugs in synchronous control/command aerospace software (Airbus).

Invented by Patrick Cousot in the seventies [CC77], and developed with his wife Radhia Cousot.



Foundations and Trends® in Programming Languages
Vol. 2, No. 2-3 (2015) 71–190
© 2015 J. Bertrane *et al.*
DOI: 10.1561/2500000002

now
the essence of knowledge

Static Analysis and Verification of Aerospace Software by Abstract Interpretation

Julien Bertrane
Département d'informatique, École normale supérieure

Patrick Cousot
Département d'informatique, École normale supérieure &
Courant Institute of Mathematical Sciences, New York University

Radhia Cousot
CNRS & Département d'informatique, École normale supérieure

Jérôme Feret
INRIA & Département d'informatique, École normale supérieure

Laurent Mauborgne
AbsInt Angewandte Informatik

Antoine Miné
Sorbonne University, University Pierre and Marie Curie, CNRS, LIP6

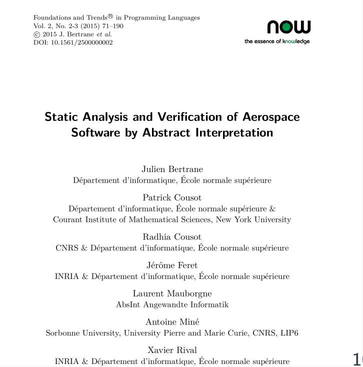
Xavier Rival
INRIA & Département d'informatique, École normale supérieure

Abstract Interpretation

General, automated, incomplete and sound.

Success story: Astrée, prove absence of bugs in synchronous control/command aerospace software (Airbus).

Invented by Patrick Cousot in the seventies [CC77], and developed with his wife Radhia Cousot.



Foundations and Trends® in Programming Languages
Vol. 2, No. 2-3 (2015) 71–190
© 2015 J. Bertrane *et al.*
DOI: 10.1561/2500000002

now
the essence of knowledge

Static Analysis and Verification of Aerospace Software by Abstract Interpretation

Julien Bertrane
Département d'informatique, École normale supérieure

Patrick Cousot
Département d'informatique, École normale supérieure &
Courant Institute of Mathematical Sciences, New York University

Radhia Cousot
CNRS & Département d'informatique, École normale supérieure

Jérôme Feret
INRIA & Département d'informatique, École normale supérieure

Laurent Mauborgne
AbsInt Angewandte Informatik

Antoine Miné
Sorbonne University, University Pierre and Marie Curie, CNRS, LIP6

Xavier Rival
INRIA & Département d'informatique, École normale supérieure

Simple Example: Pop Front

```
int pop_front(int* a, size_t& n) {
  int front = a[0];
  for(int i = 0; i < n; ++i) {
    a[i - 1] = a[i];
  }
  n--;
  return front;
}
```

This program has (at least) three bugs.

Simple Example: Pop Front

```
int pop_front(int* a, size_t& n) {
  int front = a[0];
  for(int i = 0; i < n; ++i) {
    a[i - 1] = a[i];
  }
  n--;
  return front;
}
```

This program has (at least) three bugs.

- *Invalid memory access*: `a[0]` when $n = 0$.
- *Invalid memory access*: `a[i - 1]` when $i = 0$.
- *Overflow*: `++i` can overflow since we can have $n >$ `INT_MAX`.

Simple Example: Pop Front

```cpp
int pop_front(int* a, size_t& n) {
  int front = a[0];
  for(int i = 0; i < n; ++i) {
    a[i - 1] = a[i];
  }
  n--;
  return front;
}
```

Let's run MOPSA, a static analyzer, on this program:

`mopsa-c pop_front.c`

```
pop_front.c: In function 'main':
pop_front.c:12.14-18: warning: Invalid memory access

 12:    int first = a[0];
                    ^^^^
```

```
pop_front.c: In function 'main':
pop_front.c:14.4-12: error: Invalid memory access

 14:      a[i - 1] = a[i];
          ^^^^^^^^
```

```
pop_front.c: In function 'main':
pop_front.c:13.24-27: warning: Integer overflow

 13:    for(int i = 1; i < n; ++i) {
                              ^^^
```

Simple Example: Push Front

Corrected version:

```
int pop_front(int* a, size_t& n) {
  if(n == 0) return -1;
  int front = a[0];
  for(size_t i = 1; i < n; ++i) {
    a[i - 1] = a[i];
  }
  n--;
  return front;
}
```

```
Analysis terminated successfully
✔ No alarm
Analysis time: 0.353s
Checks summary: 132 total, ✔ 132 safe
  Stub condition: 9 total, ✔ 9 safe
  Invalid memory access: 59 total, ✔ 59 safe
  Integer overflow: 63 total, ✔ 63 safe
  Negative array size: 1 total, ✔ 1 safe
```

Abstract Interpretation

Abstract interpretation answers precisely elementary questions:

- What is a program?
- What is a property of a program?
- What is the verification problem?

We now formally introduce abstract interpretation:

- **Concrete semantics**: answer the questions above.
- **Abstract semantics**: design effective verification algorithm.

Concrete Semantics

Syntax

$$
\begin{array}{rll}
\langle S \rangle ::= & X \leftarrow E & \textit{assignment} \\
\mid & \textbf{if } E \circ E \textbf{ then } S \textbf{ else } S \textbf{ fi} & \textit{conditional} \\
\mid & \textbf{while } E \circ E \textbf{ do } S \textbf{ done} & \textit{loop} \\
\mid & S \,;\, S & \textit{sequence} \\
\\
\langle E \rangle ::= & X & \textit{variable} \\
\mid & -E & \textit{negation} \\
\mid & E \diamond E & \textit{arithmetic operation} \\
\mid & c & \textit{constant } c \in \mathbb{Z} \\
\mid & [a, b] & \textit{random input } a, b \in \mathbb{Z} \cup \{\pm\infty\}, a \leq [a, b] \leq b
\end{array}
$$

where $\circ \in \{=, \neq, \leq, <, >, \geq, \ldots\}$ and $\diamond \in \{+, -, /, *, \%, \ldots\}$.

What is a Program?

Let's define:

- $\mathcal{X} : Var \to \mathbb{Z}$ the set of environments.
- $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ the set of control points.

At each control point, we look for the set of all possible values of i:

	set of values of i
$^{\ell_1}$	$\mathbb{Z}_{\ell_1}$
$i \leftarrow 1;^{\ell_2}$	$\{1\}_{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**	$\{1\}_{\ell_3}$
$\quad ^{\ell_4}$	$\{1\}_{\ell_4}$
$\quad i \leftarrow i + 2;^{\ell_5}$	$\{3\}_{\ell_5}$
done$^{\ell_6}$	

What is a Program?

Let's define:

- $\mathcal{X} : Var \to \mathbb{Z}$ the set of environments.
- $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ the set of control points.

At each control point, we look for the set of all possible values of i:

	set of values of i
$^{\ell_1}$	$\mathbb{Z}_{\ell_1}$
$i \leftarrow 1;\ ^{\ell_2}$	$\{1\}_{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**	$\{1, 3\}_{\ell_3}$
$\quad ^{\ell_4}$	$\{1, 3\}_{\ell_4}$
$\quad i \leftarrow i + 2;\ ^{\ell_5}$	$\{3, 5\}_{\ell_5}$
done$^{\ell_6}$	

What is a Program?

Let's define:

- $\mathcal{X} : Var \to \mathbb{Z}$ the set of environments.
- $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ the set of control points.

At each control point, we look for the set of all possible values of i:

	set of values of i
$^{\ell_1}$	$\mathbb{Z}_{\ell_1}$
$i \leftarrow 1;\ ^{\ell_2}$	$\{1\}_{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**	$\{1, 3, 5, 7, 9, 11\}_{\ell_3}$
$^{\ell_4}$	$\{1, 3, 5, 7, 9\}_{\ell_4}$
$i \leftarrow i + 2;\ ^{\ell_5}$	$\{3, 5, 7, 9, 11\}_{\ell_5}$
done$^{\ell_6}$	$\{11\}_{\ell_6}$

Property of Programs

	set of values of i
$^{\ell_1}$	$\mathbb{Z}_{\ell_1}$
$i \leftarrow 1;^{\ell_2}$	$\{1\}_{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**	$\{1,3,5,7,9,11\}_{\ell_3}$
$\quad ^{\ell_4}$	$\{1,3,5,7,9\}_{\ell_4}$
$\quad i \leftarrow i+2;^{\ell_5}$	$\{3,5,7,9,11\}_{\ell_5}$
done$^{\ell_6}$	$\{11\}_{\ell_6}$

- The sets S_{ℓ_i} are called *invariants*.
- They are the strongest possible, there is no set S'_{ℓ_i} such that $S_{\ell_i} \subset S'_{\ell_i}$.
- A property has the same domain than an invariant, for instance: `assert(i >= 11)` after ℓ_6 is the property $\{11, 12, 13, 14, 15, \dots\}$.
- Clearly this property is validated since $\{11\}_{\ell_6} \subseteq \{11, 12, 13, 14, 15, \dots\}$ (the program is even more restrictive than the property checked).

How to automatically compute the sets S_{ℓ_i}?

Semantics of Atomic Commands

First, we compute these sets for each expression and atomic commands of the language:

- Semantics of expressions: $\mathbf{E} : \mathit{Expr} \times \mathcal{X} \to \mathcal{P}(\mathbb{Z})$.
- Semantics of commands: $\mathbf{C} : \mathit{Com} \times \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{X})$.

Examples

- Simple arithmetic: $\mathbf{E}(x * y, \{x \mapsto 4, y \mapsto 2\}) = \{8\}$.
- Assignment: $\mathbf{C}(x \leftarrow [1, 2], \{\{x \mapsto 10, y \mapsto 1\}\}) = \{\{x \mapsto 1, y \mapsto 1\}, \{x \mapsto 2, y \mapsto 1\}\}$
- Filtering: $\mathbf{C}(x \neq 2, \{\{x \mapsto 1, y \mapsto 1\}, \{x \mapsto 2, y \mapsto 1\}\}) = \{\{x \mapsto 1, y \mapsto 1\}\}$

Equational Semantic Illustrated

- At each location $\ell \in \mathcal{L}$, we compute its set of reachable environments $\mathcal{X}_\ell$.
- We create an equational system from the program such that its solution is $\{\mathcal{X}_{\ell_1}, \dots, \mathcal{X}_{\ell_n}\}$.

$^{\ell_1}$ $i \leftarrow 1;$ $^{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**
  $^{\ell_4}$ $i \leftarrow i + 2$ $^{\ell_5}$
done$^{\ell_6}$

$$
\begin{aligned}
\mathcal{X}_{\ell_1} &= \mathcal{X} \\
\mathcal{X}_{\ell_2} &= \mathbf{C}(i \leftarrow 1, \mathcal{X}_{\ell_1}) \\
\mathcal{X}_{\ell_3} &= \mathcal{X}_{\ell_2} \cup \mathcal{X}_{\ell_5} \\
\mathcal{X}_{\ell_4} &= \mathbf{C}(i \leq 10, \mathcal{X}_{\ell_3}) \\
\mathcal{X}_{\ell_5} &= \mathbf{C}(i \leftarrow i + 2, \mathcal{X}_{\ell_4}) \\
\mathcal{X}_{\ell_6} &= \mathbf{C}(i > 10, \mathcal{X}_{\ell_3})
\end{aligned}
$$

Equational Semantic Illustrated

- At each location $\ell \in \mathcal{L}$, we compute its set of reachable environments $\mathcal{X}_\ell$.
- We create an equational system from the program such that its solution is $\{\mathcal{X}_{\ell_1}, \ldots, \mathcal{X}_{\ell_n}\}$.

$^{\ell_1}$ $i \leftarrow 1;$ $^{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**
    $^{\ell_4}$ $i \leftarrow i + 2$ $^{\ell_5}$
done$^{\ell_6}$

$$
\begin{aligned}
\mathcal{X}_{\ell_1} &= \mathcal{X} \\
\mathcal{X}_{\ell_2} &= \mathbf{C}(i \leftarrow 1, \mathcal{X}_{\ell_1}) \\
\mathcal{X}_{\ell_3} &= \mathcal{X}_{\ell_2} \cup \mathcal{X}_{\ell_5} \\
\mathcal{X}_{\ell_4} &= \mathbf{C}(i \leq 10, \mathcal{X}_{\ell_3}) \\
\mathcal{X}_{\ell_5} &= \mathbf{C}(i \leftarrow i + 2, \mathcal{X}_{\ell_4}) \\
\mathcal{X}_{\ell_6} &= \mathbf{C}(i > 10, \mathcal{X}_{\ell_3})
\end{aligned}
$$

Location	0	1	2	3
$\mathcal{X}_{\ell_1}$	$\{\}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
$\mathcal{X}_{\ell_2}$	$\{\}$	$\{\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$
$\mathcal{X}_{\ell_3}$	$\{\}$	$\{\}$	$\{\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$
$\mathcal{X}_{\ell_4}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
$\mathcal{X}_{\ell_5}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
$\mathcal{X}_{\ell_6}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$

Equational Semantic Illustrated

- At each location $\ell \in \mathcal{L}$, we compute its set of reachable environments $\mathcal{X}_\ell$.
- We create an equational system from the program such that its solution is $\{\mathcal{X}_{\ell_1}, \dots, \mathcal{X}_{\ell_n}\}$.

$^{\ell_1}\ i \leftarrow 1;\ ^{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**
    $^{\ell_4}\ i \leftarrow i + 2\ ^{\ell_5}$
done$^{\ell_6}$

$$
\begin{aligned}
\mathcal{X}_{\ell_1} &= \mathcal{X} \\
\mathcal{X}_{\ell_2} &= \mathbf{C}(i \leftarrow 1, \mathcal{X}_{\ell_1}) \\
\mathcal{X}_{\ell_3} &= \mathcal{X}_{\ell_2} \cup \mathcal{X}_{\ell_5} \\
\mathcal{X}_{\ell_4} &= \mathbf{C}(i \leq 10, \mathcal{X}_{\ell_3}) \\
\mathcal{X}_{\ell_5} &= \mathbf{C}(i \leftarrow i + 2, \mathcal{X}_{\ell_4}) \\
\mathcal{X}_{\ell_6} &= \mathbf{C}(i > 10, \mathcal{X}_{\ell_3})
\end{aligned}
$$

Location	3	4	$\dots 10$
$\mathcal{X}_{\ell_1}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
$\mathcal{X}_{\ell_2}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$
$\mathcal{X}_{\ell_3}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = v, v \in \{1, 3, \dots, 11\}\}$
$\mathcal{X}_{\ell_4}$	$\{\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 1\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = v, v \in \{1, 3, \dots, 9\}\}$
$\mathcal{X}_{\ell_5}$	$\{\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 3\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = v, v \in \{3, \dots, 11\}\}$
$\mathcal{X}_{\ell_6}$	$\{\}$	$\{\}$	$\{\rho \in \mathcal{X} \mid \rho(i) = 11\}$

Fixpoint reached after 10 iterations. This way of computing the fixpoint is called *Jacobi iterations*.

Properties of Equational Semantics

- From $\mathcal{X}_{\ell_2} = \mathbf{C}(i \leftarrow 1, \mathcal{X}_{\ell_1})$ to:

$$F_2(\{\mathcal{X}_{\ell_1}, \ldots, \mathcal{X}_{\ell_6}\}) = \{\mathcal{X}_{\ell_1}, \mathbf{C}(i \leftarrow 1, \mathcal{X}_{\ell_1}), \ldots, \mathcal{X}_{\ell_6}\}$$

$F_i(\{\mathcal{X}_{\ell_1}, \ldots, \mathcal{X}_{\ell_n}\}) = \{\mathcal{X}'_{\ell_1}, \ldots, \mathcal{X}'_{\ell_n}\}$

- Then, the fixpoint of $F_n \circ F_{n-1} \circ \ldots \circ F_1$ starting at $\{\{\}_{\ell_1}, \ldots, \{\}_{\ell_n}\}$ is

the unique least fixpoint.

(by Kleene theorem and continuity of all F_i).

- Hence, the equational semantics capture all possible executions and nothing more!

Summary

Abstract interpretation answers precisely the questions we raised at the beginning:

- What is a program? The least fixpoint point of $\mathbf{eq}(S)$.
- What is a property? A subset of the environment $P \in \mathcal{P}(\mathcal{X})$.
  Example: $i < 12$ is the property $\{\rho \in \mathcal{X} \mid \rho(i) = v, v \in \{1, 2, \ldots, 11\}\}$.
- What is the verification problem? An inclusion check: $(\mathbf{lfp}\ \mathbf{eq}(S))_{\ell_i} \subseteq P$.
  Example: $\mathcal{X}_{\ell_6} = \{\rho \in \mathcal{X} \mid \rho(i) = 11\} \subseteq \{\rho \in \mathcal{X} \mid \rho(i) = v, v \in \{1, 2, \ldots, 11\}\}$

Small Issues...

- **lfp eq**(S) might only exists after an infinite number of iterations.
- Even if finite, the sets $\mathcal{X}_{\ell_i}$ can grow exponentially, and the number of iterations can be very big.

Abstract Semantics

Intuitions of Abstract Semantics

Abstract Semantics

Let's over-approximate the least fixpoint:

$$\textbf{lfp eq}(S) \subseteq \textbf{lfp eq}^\sharp(S)$$

such that $\textbf{lfp eq}^\sharp(S)$ is computable in a finite number of steps.

Soundness

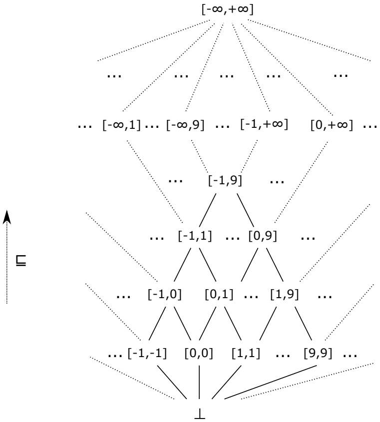
If a property can be proved in the abstract semantics, it is true to hold in the concrete semantics:

$$(\textbf{lfp eq}^\sharp(S))_{\ell_i} \subseteq P \Rightarrow \textbf{lfp eq}(S) \subseteq P$$

The interval lattice

Introduced by [Cous76].

$$\mathcal{B}^\sharp \stackrel{\text{def}}{=} \{\, [a,b] \mid a \in \mathbb{I} \cup \{-\infty\},\ b \in \mathbb{I} \cup \{+\infty\},\ a \leq b \,\} \cup \{\, \perp_b^\sharp \,\}$$



Note: intervals are open at infinite bounds $+\infty$, $-\infty$.

Abstract Fixpoint

Instead of working on the set of concrete values, we work on intervals.

$\Rightarrow$ Each operator must be have a computable and effective abstract counterpart (annotated with $^\sharp$).

$^{\ell_1}$ $i \leftarrow 1;$ $^{\ell_2}$
while $^{\ell_3} i \leq 10$ **do**
$^{\ell_4}$ $i \leftarrow i + 2$ $^{\ell_5}$
done$^{\ell_6}$

$$
\begin{aligned}
\mathcal{X}^\sharp_{\ell_1} &= \mathcal{X}^\sharp \\
\mathcal{X}^\sharp_{\ell_2} &= \mathbf{C}^\sharp(i \leftarrow 1, \mathcal{X}^\sharp_{\ell_1}) \\
\mathcal{X}^\sharp_{\ell_3} &= \mathcal{X}^\sharp_{\ell_2} \cup^\sharp \mathcal{X}^\sharp_{\ell_5} \\
\mathcal{X}^\sharp_{\ell_4} &= \mathbf{C}^\sharp(i \leq 10, \mathcal{X}^\sharp_{\ell_3}) \\
\mathcal{X}^\sharp_{\ell_5} &= \mathbf{C}^\sharp(i \leftarrow i + 2, \mathcal{X}^\sharp_{\ell_4}) \\
\mathcal{X}^\sharp_{\ell_6} &= \mathbf{C}^\sharp(i > 10, \mathcal{X}^\sharp_{\ell_3})
\end{aligned}
$$

Loss of Precision

$$
\begin{array}{l}
{}^{\ell_1}\ i \leftarrow 1;\,{}^{\ell_2}\\
\textbf{while}\ {}^{\ell_3} i \leq 10\ \textbf{do}\\
\quad {}^{\ell_4}\ i \leftarrow i + 2\ {}^{\ell_5}\\
\textbf{done}^{\ell_6}
\end{array}
$$

$$
\begin{aligned}
\mathcal{X}^\sharp_{\ell_1} &= \mathcal{X}^\sharp\\
\mathcal{X}^\sharp_{\ell_2} &= \mathbf{C}^\sharp(i \leftarrow 1, \mathcal{X}^\sharp_{\ell_1})\\
\mathcal{X}^\sharp_{\ell_3} &= \mathcal{X}^\sharp_{\ell_2} \cup^\sharp \mathcal{X}^\sharp_{\ell_5}\\
\mathcal{X}^\sharp_{\ell_4} &= \mathbf{C}^\sharp(i \leq 10, \mathcal{X}^\sharp_{\ell_3})\\
\mathcal{X}^\sharp_{\ell_5} &= \mathbf{C}^\sharp(i \leftarrow i + 2, \mathcal{X}^\sharp_{\ell_4})\\
\mathcal{X}^\sharp_{\ell_6} &= \mathbf{C}^\sharp(i > 10, \mathcal{X}^\sharp_{\ell_3})
\end{aligned}
$$

Working in the abstract can result in weaker invariants (loss of precision).

Example

- The first time we reach ℓ_5, we have $\mathcal{X}^\sharp_{\ell_5} = \{x \mapsto [1..3]\}$.
- But $2 \in [1..3]$ although it is not a possible value!
- This interval analysis would be unable to prove that $x \neq 2$ at location ℓ_5.

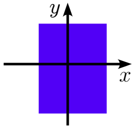
Many techniques to improve precision

- **Various abstract domains** with different precision/efficiency tradeoff (replacing intervals in the previous example).
- **Various products of abstract domains** to combine their strengths.
- More efficient fixpoint algorithms.
- ...

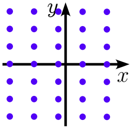
Course plan (2/8)

Bricks of abstraction: numerical domains

simple domains

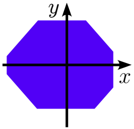


Intervals
$x \in [a, b]$

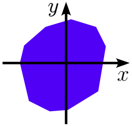


Congruences
$x \in a\mathbb{Z} + b$

relational domains

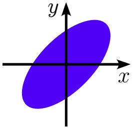


Octagons
$\pm x \pm y \leq c$

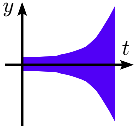


Polyhedra
$\sum_i \alpha_i x_i \leq \beta$

specific domains



Ellipsoids
digital filters



Exponentials
rounding errors

Research Plan

Research Topics

- **Abstract Constraint Reasoning**: *Can abstract interpretation be the backbone theory to unify constraint reasoning approaches?*
    - Goal I: Combine constraint solvers (by reduced products) to solve more efficiently problems.
    - Goal II: Generalize reasoning procedures (e.g., multi-objective algorithms, clause learning) to monotone functions working over any abstract domains.
- **Lattice Parallel Programming**: *Can lattice theory be the backbone of a safe model of parallel programming?*
    - Goal I: Make parallel programs correct-by-construction.
    - Goal II: Take advantage of specialized hardware (e.g., GPUs, FPGAs, quantum?).

Abstract Constraint Reasoning

A framework for combining constraint solvers

SAT [DHK13]

SMT [CCM13]

Logic programming [Cou20]

Constraint programming [Pel+13]

Linear programming [CH78]

Multi-objective optimization

Multilevel programming

...

$\longrightarrow$ Abstract domains

- **Context:** In constraint programming, *propagators* are monotone functions reducing the domains of the variables.
- It is possible to design an abstract domain of propagators [TMT20] ordered by inclusion.
- Research question: From a constraint, e.g. $x + y \leq 12$, how do we automatically obtain its propagator?

Collaboration with Bruno Teheux

It seems that the algebraic essence of *propagators* comes from *residuated lattices*.

- **Context:** In constraint programming, *global constraints* are propagators with dedicated inference algorithms for subproblems, e.g., `alldifferent(`$[x_1, \ldots, x_n]$`)`.
- Research question: Which global constraints can be generalized into abstract domains?

Collaboration with Éric Monfroy

We are working on the *Table abstract domain* generalizing the well-known `table` constraint:

$$\begin{aligned} &(x \geq 4 \wedge y > 1 \wedge z < 3) \\ \vee &(x = 1 \wedge y = 2 \wedge z = 3) \\ \vee &(x > 1 \wedge y > 1 \wedge z > 3) \end{aligned}$$

Perspective: Towards automatic creation of the abstract domain

Research question: Given a set of abstract domains and reduced products, how to build the most efficient one to solve a given formula?

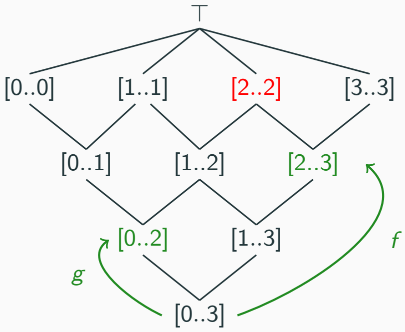
$$\begin{tikzcd}
& \varphi \arrow[dl, "\llbracket\rrbracket"'] \arrow[dr, "\llbracket\rrbracket^\flat"] & \\
A_1^\sharp \times \ldots \times A_n^\sharp \arrow[rr, shift left, "\gamma"] & & C^\flat \arrow[ll, shift left, "\alpha"]
\end{tikzcd}$$

- How to create an appropriate combination of abstract domains for a particular formula?
- “Type inference”: In which abstract domain goes each subformula $\varphi_i \in \varphi$?

Lattice Parallel Programming

Lattice Parallel Programming

Intuition: data are lattices, programs are monotone functions by construction [TPB22]:

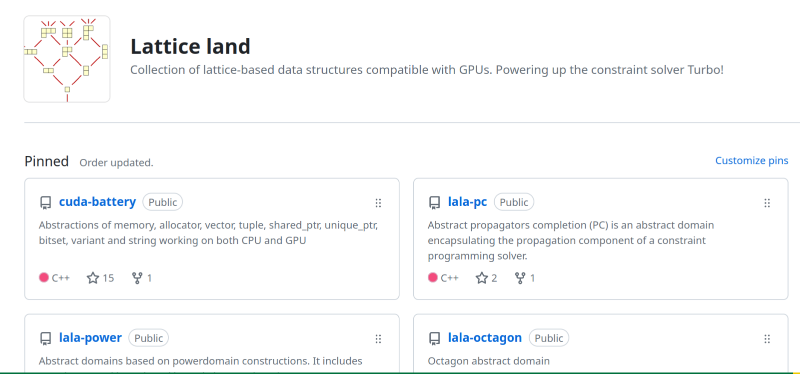


- $f(x) = x \sqcup [2..\infty]$ models the constraint $x \geq 2$.
- $g(x) = x \sqcup [-\infty..2]$ models the constraint $x \leq 2$.
- Concurrent execution: `f || g = [2..2]`

A new twist on an old idea: *asynchronous iterations* of abstract interpretation [Cou77].

In-Progress: CUDA-compatible Lattice Library

- **Various abstract domains**: interval, bitset, store, propagators completion, search tree, branch and bound, ... `https://github.com/lattice-land`
- **Octagon** soon by Thibault Falque.
- **Zonotope** soon by Yi-Nung Tsao.
- *Turbo* GPU-based constraint solver (`https://github.com/ptal/turbo/`).



Perspective: Towards zero-cost abstraction

- Generality comes with layers of software abstractions slowing down the execution.
- Idea: View abstract domain as a specialized constraint compiler to a guarded command language [TPB22], itself compiled to a lower-level language such as PTX or other architecture (e.g. FPGA).
- Why interesting: Constraint solving is basically repeating the same thing million of times. Removing abstractions is accelerating solving, unlocking compilation-based optimization, and reducing threads divergence.

Other Collaborations

Thanks to (or because of?) my Ph.D. students, I can learn about non-lattice stuff.

- Manuel on *multi-objective constraint solving algorithms*.
- Hedieh on *hyperparameter optimization of constraint solver*.
- Yi-Nung on *formal verification of neural networks*.
- Tobias on *optimization of quantum circuits*.

Fortunately, Thibault (COMOC postdoc) still play with me with lattices :-)

Conclusion

Universality of Lattice Theory and Abstract Interpretation

Abstraction and approximation are two central concepts in computer science. Abstract interpretation captures those precisely, thus has many applications beyond program analysis:

- Constraint reasoning.
- Neural network verification.
- (Gradual) typing.
- Conflict-free replicated data types (CRDTs).
- Parallel computing.

Abstract Week

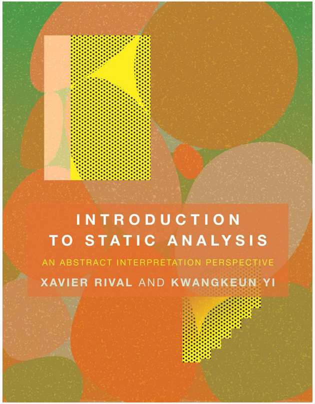
A week to learn about abstract interpretation and its applications (static analysis, constraint reasoning, neural network verification):

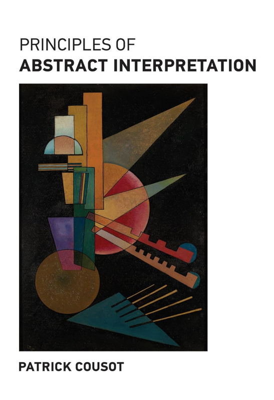
- Monday: Introduction to Lattice Theory, Bruno Teheux.
- Tuesday: Introduction to Abstract Interpretation, Pierre Talbot.
- Tuesday: The Octagon Abstract Domain, Thibault Falque.
- Wednesday: Abstract Interpretation of Neural Networks, Yi-Nung Tsao.
- Wednesday: Abstract Interpretation of Constraint Programming, Pierre Talbot.
- Thursday: Lattice Theory for Parallel Programming, Pierre Talbot.

In preparation of the MHPC course Lattice Theory for Parallel Programming with Bruno.

Resources

- MPRI class of Antoine Miné: `https://www-apr.lip6.fr/~mine/enseignement/mpri/2023-2024/` (two slides stolen from this class).
- Two recent books:





References

[CC77] Patrick Cousot and Radhia Cousot. **“Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”.** In: *POPL 77'*. ACM, 1977, pp. 238–252. DOI: 10.1145/512950.512973.

[CCM13] Patrick Cousot, Radhia Cousot, and Laurent Mauborgne. **“Theories, Solvers and Static Analysis by Abstract Interpretation”.** In: *J. ACM* 59.6 (Jan. 2013). DOI: 10.1145/2395116.2395120.

[CH78] Patrick Cousot and Nicolas Halbwachs. **“Automatic discovery of linear restraints among variables of a program”.** In: *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1978, pp. 84–96.

[Cou20] Patrick Cousot. **“The Symbolic Term Abstract Domain”.** In: *TASE* (Dec. 2020). URL: https://sei.ecnu.edu.cn/tase2020/file/video-slides-PCousot-TASE-2020.pdf.

[Cou77] Patrick Cousot. **Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice.** Research Report 88. Grenoble, France: Laboratoire IMAG, Université scientifique et médicale de Grenoble, Sept. 1977, p. 15.

[DHK13] Vijay D'Silva, Leopold Haller, and Daniel Kroening. **"Abstract Conflict Driven Learning".** In: *POPL '13*. ACM, 2013, pp. 143–154. DOI: 10.1145/2429069.2429087.

[Pel+13] Marie Pelleau et al. **"A constraint solver based on abstract domains".** In: *VMCAI 13'*. Springer, 2013, pp. 434–454. DOI: 10.1007/978-3-642-35873-9_26.

[TMT20] Pierre Talbot, Éric Monfroy, and Charlotte Truchet. **"Modular Constraint Solver Cooperation via Abstract Interpretation".** In: *Theory and Practice of Logic Programming* 20.6 (2020), pp. 848–863. DOI: 10.1017/S1471068420000162.

[TPB22] Pierre Talbot, Frédéric Pinel, and Pascal Bouvry. **"A Variant of Concurrent Constraint Programming on GPU".** In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 4. 2022, pp. 3830–3839. DOI: 10.1609/aaai.v36i4.20298.