Introduction

MPRI 2–6: Abstract Interpretation, application to verification and static analysis

Antoine Miné

Year 2018–2019

Course 00
11 September 2018

Motivation

The cost of software failure

- Patriot MIM-104 failure, 25 February 1991
  (death of 28 soldiers[1])
- Ariane 5 failure, 4 June 1996
  (cost estimated at more than 370 000 000 US$[2])
- Toyota electronic throttle control system failure, 2005
  (at least 89 death[3])
- Heartbleed bug in OpenSSL, April 2014
- the economic cost of software bugs is tremendous[4]

---

[1] R. Skeel. "Roundoff Error and the Patriot Missile". SIAM News, volume 25, nr 4.

[2] M. Dowson. "The Ariane 5 Software Failure". Software Engineering Notes 22 (2): 84, March 1997.

[3] CBSNews. Toyota "Unintended Acceleration" Has Killed 89. 20 March 2014.

[4] NIST. Software errors cost U.S. economy $59.5 billion annually. Tech. report, NIST Planning Report, 2002.

Zoom on: Ariane 5, Flight 501

Cause: software error[5]

- arithmetic overflow in unprotected data conversion from 64-bit float to 16-bit integer types[6]

```
P_M_DERIVE(T_ALG.E_BH) :=
  UC_16S_EN_16NS (TDB.T_ENTIER_16S
    ((1.0/C_M_LSB_BH) * G_M_INFO_DERIVE(T_ALG.E_BH)));
```

- software exception not caught
  $\Longrightarrow$ computer switched off
- all backup computers run the same software
  $\Longrightarrow$ all computers switched off, no guidance
  $\Longrightarrow$ rocket self-destructs

A "simple" error. . .

---

[5] J.-L. Lions et al., Ariane 501 Inquiry Board report.

[6] J.-J. Levy. Un petit bogue, un grand boum. Séminaire du Département d'informatique de l'ENS, 2010.

How can we avoid such failures?

- Choose a safe programming language.

  C (low level) / Ada, Java, OCaml (high level)

- Carefully design the software.

  many software development methods exist

- Test the software extensively.

How can we avoid such failures?

- Choose a safe programming language.

  C (low level) / Ada, Java, OCaml (high level)

  yet, Ariane 5 software is written in Ada

- Carefully design the software.

  many software development methods exist

  yet, critical embedded software follow strict development processes

- Test the software extensively.

  yet, the erroneous code was well tested... on Ariane 4

$\Longrightarrow$ **not sufficient!**

How can we avoid such failures?

- Choose a safe programming language.

  C (low level) / Ada, Java, OCaml (high level)

  yet, Ariane 5 software is written in Ada

- Carefully design the software.

  many software development methods exist

  yet, critical embedded software follow strict development processes

- Test the software extensively.

  yet, the erroneous code was well tested... on Ariane 4

$\Longrightarrow$ **not sufficient!**

We should use **formal methods.**

provide rigorous, mathematical insurance of correctness
may not prove everything, but give a precise notion of what is proved

Verification: compromises

Undecidability: correctness properties are undecidable!

cannot build a program that automatically and precisely separates all correct programs from all incorrect ones

Possible compromises:

lose automation, or completeness, or soundness, or generality, or all

- **Test**: complete and automatic, but unsound
- **Theorem proving**
  - proof essentially manual, but checked automatically
  - powerful, but very steep learning curve
- **Deductive methods**
  - automated proofs for some logic fragments (SAT, SMT)
  - still requires program annotations (contracts, invariants)
- **Model checking**
  - check a (often hand-crafted) model of the program
  - finite or regular models, expressive properties (LTL)
  - automatic and complete (wrt. model)

Verification by static analysis

source

```
int search(int* t, int n) {
  int i;
  for (i=0; i<n; i++) {
    if (t[i]) break;
  }
  return t[i];
}
```

⟹

analysis result

```
int search(int* t, int n) {
  int i;
  for (i=0; i<n; i++) {
    // 0 ≤ i < n
    if (t[i]) break;
  }
  // 0 ≤ i ≤ n ∨ n < 0
  return t[i];
}
```

✓

✗

- work directly on the source code
- infer properties on program executions
- automatically (cost effective)
- construct dynamically a semantic abstraction of the program
- deduce program correctness or raise alarms

  (implicit specification: absence of RTE; or user-defined properties: contracts)
- with approximations (incomplete: efficient, but possible false alarms)
- soundly (no false positive)

Verification in practice: The example of avionics software

Critical avionics software is subject to certification:

- more than half the development cost
- regulated by international standards (DO-178B, DO-178C)
- mostly based on massive test campaigns & intellectual reviews

Current trend:

use of **formal methods** now acknowledged (DO-178C, DO-333)

- at the binary level, to replace testing
- at the source level, to replace intellectual reviews
- at the source level, to replace testing
  provided the correspondence with the binary is also certified

$\Longrightarrow$ **formal methods can improve cost-effectiveness!**

Caveat: soundness is required by DO

Verification in practice: Formal verification at Airbus

Program proofs: deductive methods

- functional properties of small sequential C codes
- replace unit testing
- not fully automatic
- **Caveat** / **Frama-C** tool (CEA)

Sound static analysis:

- fully automated on large applications, non functional properties
- worst-case execution time and stack usage, on binary
  aiT, **StackAnalyzer** (AbsInt)
- absence of run-time error, on sequential C code
  Astrée analyzer (AbsInt)

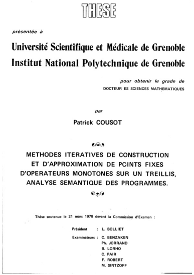
Certified compilation:

- allows source-level analysis to certify sequential binary code
- **CompCert** C compiler, certified in **Coq** (INRIA)

Overview of abstract interpretation

Abstract interpretation

Patrick Cousot[7]



THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES MATHEMATIQUES

par
Patrick COUSOT

⁂

METHODES ITERATIVES DE CONSTRUCTION
ET D'APPROXIMATION DE POINTS FIXES
D'OPERATEURS MONOTONES SUR UN TREILLIS,
ANALYSE SEMANTIQUE DES PROGRAMMES.

⁂

Thèse soutenue le 21 mars 1978 devant la Commission d'Examen :

Président : L. BOLLIET
Examinateurs : C. BENZAKEN
Ph. JORRAND
B. LORHO
C. PAIR
F. ROBERT
M. SINTZOFF

General theory of the approximation and comparison of program semantics:

- unifies existing semantics (proposed independently)
- guides the design of static analyses that are correct by construction

---

[7] P. Cousot. "Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes." Thèse És Sciences Mathématiques, 1978.

Concrete semantics

```
($\mathcal{S}_0$)
assume X in [0,1000];
($\mathcal{S}_1$)
I := 0;
($\mathcal{S}_2$)
while ($\mathcal{S}_3$) I < X do
    ($\mathcal{S}_4$)
    I := I + 2;
    ($\mathcal{S}_5$)
($\mathcal{S}_6$)
```

program

Concrete semantics

```
($\mathcal{S}_0$)
assume X in [0,1000];
($\mathcal{S}_1$)
I := 0;
($\mathcal{S}_2$)
while ($\mathcal{S}_3$) I < X do
    ($\mathcal{S}_4$)
    I := I + 2;
    ($\mathcal{S}_5$)
($\mathcal{S}_6$)
```

program

$$\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I}, \mathtt{X}\} \to \mathbb{Z})$$

$$\begin{array}{lll}
\mathcal{S}_0 = \{\, (i,x) \mid i,x \in \mathbb{Z} \,\} & = \top \\
\mathcal{S}_1 = \{\, (i,x) \in \mathcal{S}_0 \mid x \in [0,1000] \,\} & = F_1(\mathcal{S}_0) \\
\mathcal{S}_2 = \{\, (0,x) \mid \exists i,\ (i,x) \in \mathcal{S}_1 \,\} & = F_2(\mathcal{S}_1) \\
\mathcal{S}_3 = \mathcal{S}_2 \cup \mathcal{S}_5 & \\
\mathcal{S}_4 = \{\, (i,x) \in \mathcal{S}_3 \mid i < x \,\} & = F_4(\mathcal{S}_3) \\
\mathcal{S}_5 = \{\, (i+2,x) \mid (i,x) \in \mathcal{S}_4 \,\} & = F_5(\mathcal{S}_4) \\
\mathcal{S}_6 = \{\, (i,x) \in \mathcal{S}_3 \mid i \geq x \,\} & = F_6(\mathcal{S}_3)
\end{array}$$

semantics

Concrete semantics $\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I}, \mathtt{X}\} \to \mathbb{Z})$:

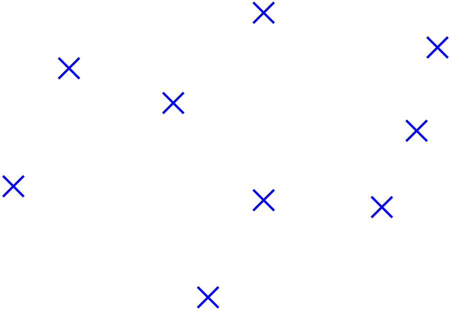
- strongest program properties (inductive invariants)
- smallest solution of a system of equations, on sets
- well-defined solution, but not computable in general

Abstracting

Principle: be tractable by reasoning at an abstract level

Abstracting

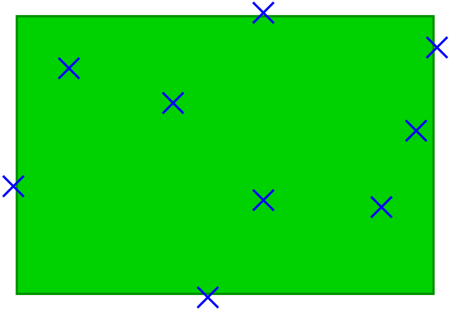
Principle: be tractable by reasoning at an abstract level



concrete executions : $\{(0,3),(5.5,0),(12,7),\ldots\}$ (not computable)

Abstracting

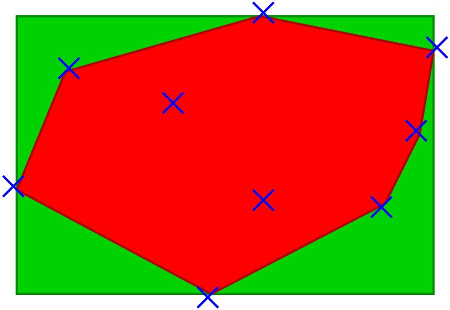
Principle: be tractable by reasoning at an abstract level



concrete executions :	$\{(0,3),(5.5,0),(12,7),\ldots\}$	(not computable)
box domain :	$X \in [0,12] \wedge Y \in [0,8]$	(linear cost)

Abstracting

Principle: be tractable by reasoning at an abstract level



concrete executions :	$\{(0,3),(5.5,0),(12,7),\ldots\}$	(not computable)
box domain :	$X \in [0,12] \wedge Y \in [0,8]$	(linear cost)
polyhedra domain :	$6X + 11Y \geq 33 \wedge \cdots$	(exponential cost)

many abstractions: trade-off cost vs. precision and expressiveness

From concrete to abstract semantics

```
$(\mathcal{S}_0)$
assume X in [0,1000];
$(\mathcal{S}_1)$
I := 0;
$(\mathcal{S}_2)$
while $(\mathcal{S}_3)$ I < X do
    $(\mathcal{S}_4)$
    I := I + 2;
    $(\mathcal{S}_5)$
$(\mathcal{S}_6)$
```

program

$$\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I},\mathtt{X}\} \to \mathbb{Z})$$

$$\begin{array}{lll}
\mathcal{S}_0 & = \{\, (i,x) \mid i,x \in \mathbb{Z} \,\} & = \top \\
\mathcal{S}_1 & = \{\, (i,x) \in \mathcal{S}_0 \mid x \in [0,1000] \,\} & = F_1(\mathcal{S}_0) \\
\mathcal{S}_2 & = \{\, (0,x) \mid \exists i,\ (i,x) \in \mathcal{S}_1 \,\} & = F_2(\mathcal{S}_1) \\
\mathcal{S}_3 & = \mathcal{S}_2 \cup \mathcal{S}_5 & \\
\mathcal{S}_4 & = \{\, (i,x) \in \mathcal{S}_3 \mid i < x \,\} & = F_4(\mathcal{S}_3) \\
\mathcal{S}_5 & = \{\, (i+2,x) \mid (i,x) \in \mathcal{S}_4 \,\} & = F_5(\mathcal{S}_4) \\
\mathcal{S}_6 & = \{\, (i,x) \in \mathcal{S}_3 \mid i \geq x \,\} & = F_6(\mathcal{S}_3)
\end{array}$$

concrete semantics

Concrete semantics $\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I},\mathtt{X}\} \to \mathbb{Z})$:

- strongest program properties (inductive invariants)
- smallest solution of a system of equations
- not computable in general

From concrete to abstract semantics

$(\mathcal{S}_0)$

`assume X in [0,1000];`

$(\mathcal{S}_1)$

`I := 0;`

$(\mathcal{S}_2)$

`while` $(\mathcal{S}_3)$ `I < X do`

    $(\mathcal{S}_4)$

    `I := I + 2;`

    $(\mathcal{S}_5)$

$(\mathcal{S}_6)$

program

$$\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$$

$$\begin{aligned}
\mathcal{S}_0^\sharp &= \top^\sharp \\
\mathcal{S}_1^\sharp &= [\![\, \mathit{assume}\ X \in [0,1000] \,]\!]^\sharp(\mathcal{S}_0^\sharp) \\
\mathcal{S}_2^\sharp &= [\![\, I \leftarrow 0 \,]\!]^\sharp(\mathcal{S}_1^\sharp) \\
\mathcal{S}_3^\sharp &= S_2^\sharp \cup^\sharp \mathcal{S}_5^\sharp \\
\mathcal{S}_4^\sharp &= [\![\, \mathit{assume}\ I < X \,]\!]^\sharp(\mathcal{S}_3^\sharp) \\
\mathcal{S}_5^\sharp &= [\![\, I \leftarrow I + 2 \,]\!]^\sharp(\mathcal{S}_4^\sharp) \\
\mathcal{S}_6^\sharp &= [\![\, \mathit{assume}\ I \geq X \,]\!]^\sharp(\mathcal{S}_3^\sharp)
\end{aligned}$$

abstract semantics

Abstract semantics $\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$:

- $\mathcal{D}^\sharp$ is a subset of properties of interest

  semantic choice + a machine representation

- $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ over-approximates the effect of $F : \mathcal{D} \to \mathcal{D}$ in $\mathcal{D}^\sharp$

  with effective algorithms

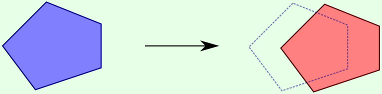
Abstract interpretation

Define an interpretation of atomic statements in the abstract,
and compose them to analyze the program

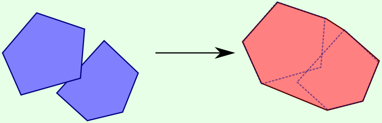
- by propagation along the edges of the control-flow graph (data-flow)
- or by induction on the syntax of programs (interpretation)

Example in the polyhedra domain

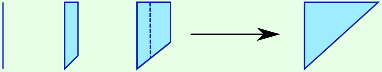
Assignment:
● `X = X + 1` ●
translation



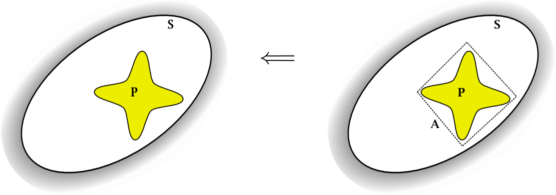
Join:
if ··· **then** ···● **else** ···● **fi** ●
convex hull



Loops or CFG cycles:
iteration with widening



Soundness and false alarms



$P \subseteq S$

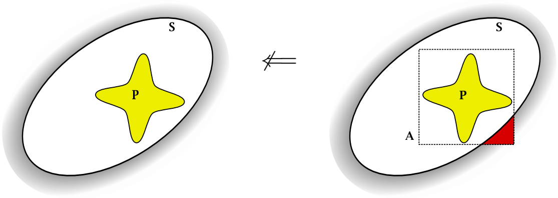
$A \subseteq S$
program proved

Goal : prove that a program P satisfies its specification S

We collect the reachable states P and compare to S

A polyhedral abstraction A can prove the correctness

Soundness and false alarms



$P \subseteq S$

$A \not\subseteq S$
false alarm

Goal : prove that a program P satisfies its specification S

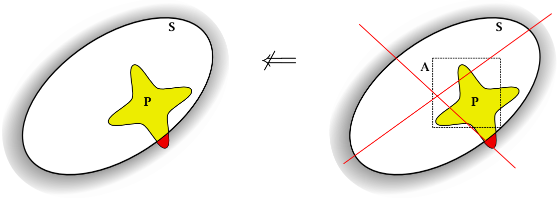
We collect the reachable states P and compare to S

A polyhedral abstraction A can prove the correctness

A box abstraction cannot prove the correctness
$\Longrightarrow$ false alarm

(especially since the analysis may not output the tightest box / polyhedron!)

Soundness and false alarms



$P \not\subseteq S$

$A \subseteq S$
false negative
cannot occur

Goal : prove that a program P satisfies its specification S

We collect the reachable states P and compare to S

A polyhedral abstraction A can prove the correctness

A box abstraction cannot prove the correctness
$\Longrightarrow$ false alarm

(especially since the analysis may not output the tightest box / polyhedron!)

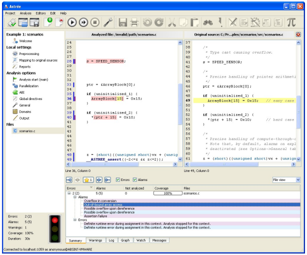
The analaysis is sound: no false negative reported!

Example static analyzer: Astrée

Astrée: developed at ENS & INRIA by P. Cousot & al.

- analyzes embedded critical C software
  subset of C, no memory allocation, no recursivity → simpler semantics
- checks for run-time errors
  arithmetic overflows, array overflows, divisions by 0, pointer errors, etc. → non-functional
- specialized for control / command software
  with zero false alarm goal
  application domain specific abstractions

Airbus A380



Analyzed file: /invalid/path/scenarios.c

```
    s = SPEED_SENSOR;




    ptr = &ArrayBlock[0];

    if (uninitialized_1) {
      ArrayBlock[15] = 0x15;
    }

    if (uninitialized_2) {
      *(ptr + 15) = 0x10;
    }







    z = (short)((unsigned short)vx + (unsig
    __ASTREE_assert((-2<=z && z<=2));
```

Line 36, Column 0

Original source: C:/Pr...ples/scenarios/src/scenarios.c

```
    /*
     * Type cast causing overflow.
     */
    s = SPEED_SENSOR;

    /*
     * Precise handling of pointer arithmeti
     */
    ptr = &ArrayBlock[0];

    if (uninitialized_1) {
      ArrayBlock[15] = 0x15;    // easy case
    }

    if (uninitialized_2) {
      *(ptr + 15) = 0x10;       // hard case
    }

    /*
     * Precise handling of compute-through-c
     * Note that, by default, alarms on expl
     * deactivated (see Options->General tab
     */
    z = (short)((unsigned short)vx + (unsign
```

Line 49, Column 0

Errors	Alarms	Not analyzed	Coverage	Files
2 (2)	5 (5)	0	100%	scenarios.c

- Alarms
  - Overflow in conversion
  - Out-of-bound array access
  - Possible overflow upon dereference
  - Possible overflow upon dereference
  - Assertion failure
- Errors
  - Definite runtime error during assignment in this context. Analysis stopped for this context.
  - Definite runtime error during assignment in this context. Analysis stopped for this context.

Errors: 2 (2)
Alarms: 5 (5)
Warnings: 1
Coverage: 100%
Duration: 30s

Connected to localhost:1059 as anonymous@ABSINT-VMWARE

2001–2004: academic success
proof of absence of RTE
on flight command

2009: industrialization

AbsInt

Example static analyzer: Infer.AI at Facebook

Infer: `http://fbinfer.com/`

- developed at Facebook (team formerly at Monoidics)
- Infer.AI is an analysis framework based on abstract interpretation
- open-source since 2015
- analyzes Java, C, C++, and Objective-C
- checks ThreadSafety (Java), Initalisation Order (C++), etc.
- modular, bottom-up interprocedural analysis
- targets the analysis of merge requests (small bits at a time)
- favors speed over soundness

  pragmatic choices, based on “what engineers want”
  no requirements for certification, unlike the avionics industry
- used in production

Course organisation

Course plan

- **foundation** of abstract interpretation (2 courses)
  - fixpoint program semantics
  - order and approximation theory
  - hierarchy of semantics

- **bricks** of static analyzers (5 courses)
  - numeric abstract domains
  - pointer and memory shape abstract domains
  - partitioning domains
  - domain combiners (reduced products, partitioning)

- domain-specific **static analyses** (9 courses)
  - analysis of control-command embedded programs
  - analysis of concurrent programs
  - analysis of program transformation
  - analysis of distributed systems
  - analysis of mobile systems
  - analysis of biological systems

Teaching team

Cezara Drăgoi

Jérôme Feret

Antoine Miné

Xavier Rival

Syllabus and exams

`https://www-apr.lip6.fr/~mine/enseignement/mpri/2018-2019`

Visit regularly for:

- latest information on course dates
- course material
- course assignments
- M2 internship proposals, updated regularly

Exams:

- 50%: written mid-term exam
- 50%: oral final exam
  (read a scientific article, present it, answer questions)

Course material

Links available on the web-page:

- main material: slides
- course notes

  cover mainly foundations and numeric abstract domains based on: A. Miné. *Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation*. In Foundations and Trends in Programming Languages, 4(3–4), 120–372. Now Publishers.
- recommended reading on theory and applications: J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, X. Rival. *Static analysis and verification of aerospace software by abstract interpretation*. In Foundations and Trends in Programming Languages, 2(2–3), 71–190, 2015. Now Publishers.

Course assignments

On the web page, **highly recommanded** homework

- exercises: prove a theorem, solve a former exam, etc.
- reading assignments: an article related to the course
- experiments: use a tool

Principle: self-evaluation

Not evaluated by the teacher, no credit.
The solution to the exercises is also given.

Additional material:

- previous exams, with correction
- course bibliography in the slides (reading not mandatory)
- optional programming project (not evaluated)