# Introduction

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

Antoine Miné

year 2014–2015

course 01
17 September 2014

# Motivating program verification

# 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

- 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

Maiden flight of the Ariane 5 Launcher, 4 June 1996.

# Zoom on: Ariane 5, Flight 501

40s after launch. . .

# 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

[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 (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 (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 (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

# Proving program properties

# Invariants and programs

```
assume X in [0,1000];

I := 0;

while I < X do

    I := I + 2;


assert I in [0,?]
```

Goal: find a bound property, sufficient to express the absence of overflow

[7] R. W. Floyd. "Assigning meanings to programs". In Proc. Amer. Math. Soc. Symposia in Applied Mathematics, vol. 19, pp. 19–31, 1967.

# Invariants and programs

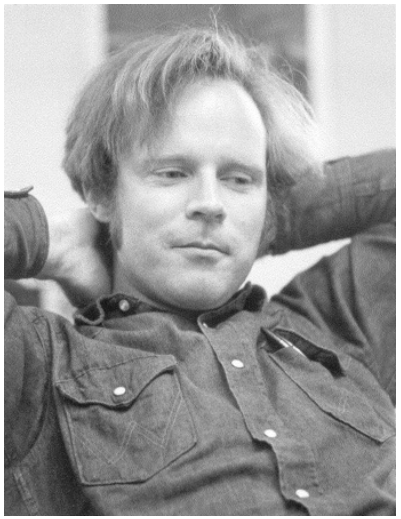
```
assume X in [0,1000];

I := 0;

while I < X do

    I := I + 2;


assert I in [0,1000]
```

Goal: find a bound property, sufficient to express the absence of overflow

[7] R. W. Floyd. "Assigning meanings to programs". In Proc. Amer. Math. Soc. Symposia in Applied Mathematics, vol. 19, pp. 19–31, 1967.

# Invariants and programs

```
assume X in [0,1000];
```
$\{X \in [0, 1000]\}$
```
I := 0;
```
$\{X \in [0, 1000], I = 0\}$
```
while I < X do
```
$\{X \in [0, 1000], I \in [0, 998]\}$
```
    I := I + 2;
```
$\{X \in [0, 1000], I \in [2, 1000]\}$
```
assert I in [0,1000]
```



Robert Floyd[7]

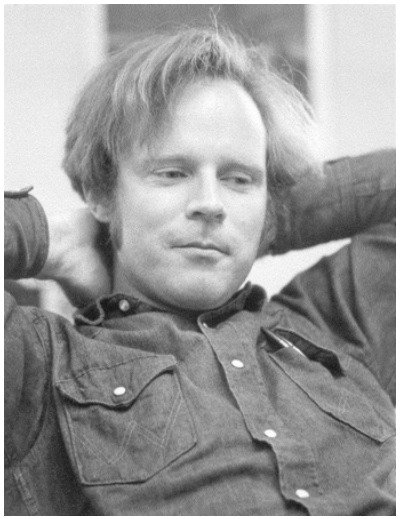
**invariant**: property true of all the executions of the program

[7] R. W. Floyd. "Assigning meanings to programs". In Proc. Amer. Math. Soc. Symposia in Applied Mathematics, vol. 19, pp. 19–31, 1967.

# Invariants and programs

```
assume X in [0,1000];
```
$\{X \in [0, 1000]\}$
```
I := 0;
```
$\{X \in [0, 1000], I = 0\}$
```
while I < X do
```
$\{X \in [0, 1000], I \in [0, 998]\}$
```
    I := I + 2;
```
$\{X \in [0, 1000], I \in [2, 1000]\}$

$\{X \in [0, 1000], I \in [0, 1000]\}$
```
assert I in [0,1000]
```



Robert Floyd[7]

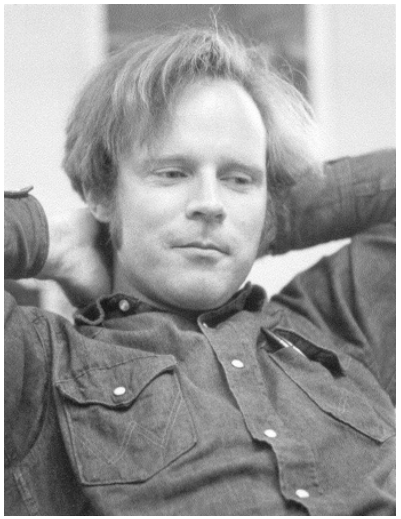
**invariant**: property true of all the executions of the program
**issue**: if $I = 997$ at a loop iteration, $I = 999$ at the next

[7] R. W. Floyd. "Assigning meanings to programs". In Proc. Amer. Math. Soc. Symposia in Applied Mathematics, vol. 19, pp. 19–31, 1967.

# Invariants and programs

```
assume X in [0,1000];
```
$\{X \in [0, 1000]\}$
```
I := 0;
```
$\{X \in [0, 1000], I = 0\}$
```
while I < X do
```
$\{X \in [0, 1000], I \in \{0, 2, \ldots, 996, 998\}\}$
```
    I := I + 2;
```
$\{X \in [0, 1000], I \in \{2, 4, \ldots, 998, 1000\}\}$

$\{X \in [0, 1000], I \in \{0, 2, \ldots, 998, 1000\}\}$
```
assert I in [0,1000]
```



Robert Floyd[7]

**inductive invariant**: invariant that can be proved to hold by induction on loop iterates

(if $I \in S$ at a loop iteration, then $I \in S$ at the next loop iteration)

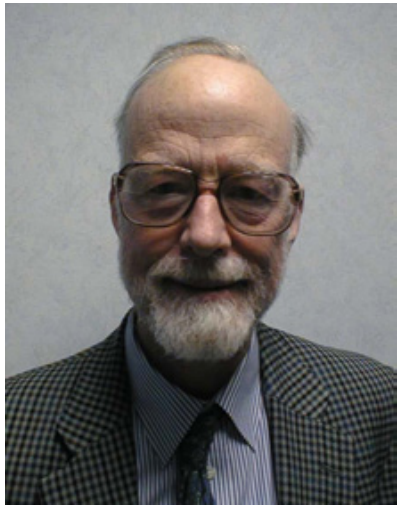
[7] R. W. Floyd. "Assigning meanings to programs". In Proc. Amer. Math. Soc. Symposia in Applied Mathematics, vol. 19, pp. 19–31, 1967.

# Logics and programs

$$\frac{}{\{P[e/X]\}\ \mathtt{X := e}\ \{P\}} \qquad \frac{\{P\}\ \mathtt{C_1}\ \{R\} \quad \{R\}\ \mathtt{C_2}\ \{Q\}}{\{P\}\ \mathtt{C_1; C_2}\ \{Q\}}$$

$$\frac{\{P\ \&\ b\}\ \mathtt{C}\ \{P\}}{\{P\}\ \mathtt{while\ b\ do\ C}\ \{P\ \&\ \neg b\}}$$

$\ldots$



Tony Hoare[8]

- sound logic to prove program properties, (rel.) complete
- proofs can be partially automated (at least proof checking)
  (e.g., using proof assistants: Coq, PVS, Isabelle, HOL, etc.)

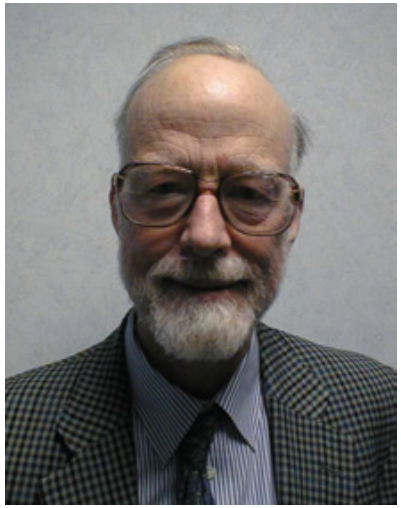
[8] C. A. R. Hoare. "An Axiomatic Basis for Computer Programming". Commun. ACM 12(10): 576-580 (1969).

# Logics and programs

$$\frac{}{\{P[e/X]\}\ \mathtt{X := e}\ \{P\}} \qquad \frac{\{P\}\ \mathtt{C_1}\ \{R\} \quad \{R\}\ \mathtt{C_2}\ \{Q\}}{\{P\}\ \mathtt{C_1; C_2}\ \{Q\}}$$

$$\frac{\{P\ \&\ b\}\ \mathtt{C}\ \{P\}}{\{P\}\ \mathtt{while\ b\ do\ C}\ \{P\ \&\ \neg b\}}$$

$\ldots$



Tony Hoare[8]

- sound logic to prove program properties, (rel.) complete
- proofs can be partially automated (at least proof checking)
  (e.g., using proof assistants: Coq, PVS, Isabelle, HOL, etc.)
- requires annotations and interaction with a prover
  even manual annotation is not practical for large programs

[8] C. A. R. Hoare. "An Axiomatic Basis for Computer Programming". Commun. ACM 12(10): 576-580 (1969).

# Limit to automation

# Computers, programs, data

$$O(P, D) \in \{yes, no, \bot\}$$

$O$


$P$


$D$

The computer $O$ runs the program $P$ on the data $D$ and answers ($yes$,$no$)... or does not answer ($\bot$).

# Computers, programs, data

$$O(P, D) \in \{yes, no, \bot\}$$

$O$


$P$


$P'$

Note that programs are also a kind of data!
They can be fed to other programs. (e.g., to compilers)

# Static analysis

Static analyzer $A$.
Given a program $P$:

- $O(A, P) = \mathit{yes} \iff \forall D,\ O(P, D)$ is safe
- $O(A, P) \neq \bot$ (the static analysis always terminates)

# Static analysis

Static analyzer $A$.
Given a program $P$:

- $O(A, P) = \mathit{yes} \iff \forall D,\ O(P, D)$ is safe
- $O(A, P) \neq \bot$ (the static analysis always terminates)

$\Longrightarrow$ $P$ is proved safe even before it is run!

# Fundamental undecidability

There cannot exist a static analyzer $A$ proving the termination of every terminating program $P$.

Alan Turing[9]

[9] A. M. Turing. "Computability and definability". The Journal of Symbolic Logic, vol. 2, pp. 153–163, (1937).

[10] H. G. Rice. "Classes of Recursively Enumerable Sets and Their Decision Problems." Trans. Amer. Math. Soc. 74, 358-366, 1953.

# Fundamental undecidability

There cannot exist a static analyzer $A$ proving the termination of every terminating program $P$.

Proof sketch:

$$A(P \cdot D) : O(A, P \cdot D) = \left| \begin{array}{l} \textit{yes} \text{ if } O(P, D) \neq \bot \\ \textit{no} \text{ otherwise} \end{array} \right.$$

$A'(X)$ : `while A(X·X) do` *nothing*`;` *no*

do we have $O(A', A') = \bot$ or $\neq \bot$? neither!
$\Longrightarrow$ $A$ cannot exist

Alan Turing[9]

All "interesting" properties are undecidable![10] 

[9] A. M. Turing. "Computability and definability". The Journal of Symbolic Logic, vol. 2, pp. 153–163, (1937).

[10] H. G. Rice. "Classes of Recursively Enumerable Sets and Their Decision Problems." Trans. Amer. Math. Soc. 74, 358-366, 1953.

# Approximation

# Approximate static analysis

An approximate static analyzer $A$ always answers in finite time ($\neq \bot$):

- either *yes*: the program $P$ is definitely safe (soundness)
- either *no*: I don't know (incompleteness)

Sufficient to prove the safety of (some) programs.
Fails on infinitely many programs...

# Approximate static analysis

An approximate static analyzer $A$ always answers in finite time ($\neq \bot$):

- either *yes*: the program $P$ is definitely safe (soundness)
- either *no*: I don't know (incompleteness)

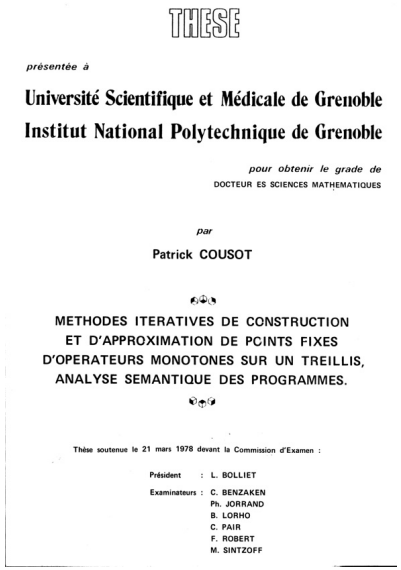
Sufficient to prove the safety of (some) programs.
Fails on infinitely many programs...

$\Longrightarrow$ We should adapt the analyzer $A$ to

- a class of programs to verify, and
- a class of safety properties to check.

# Abstract interpretation

Patrick Cousot[11]



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 many existing semantics
- allows the definition of new static analyses that are correct by construction

[11] 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.

# Abstract interpretation

$(\mathcal{S}_0)$
```
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

# Abstract interpretation

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
(S6)
```

program

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

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

semantics

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

- strongest invariant (and an inductive invariant)
- not computable in general
- smallest solution of a system of equations

# Abstract interpretation

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
     (S4)
     I := I + 2;
     (S5)
(S6)
```

program

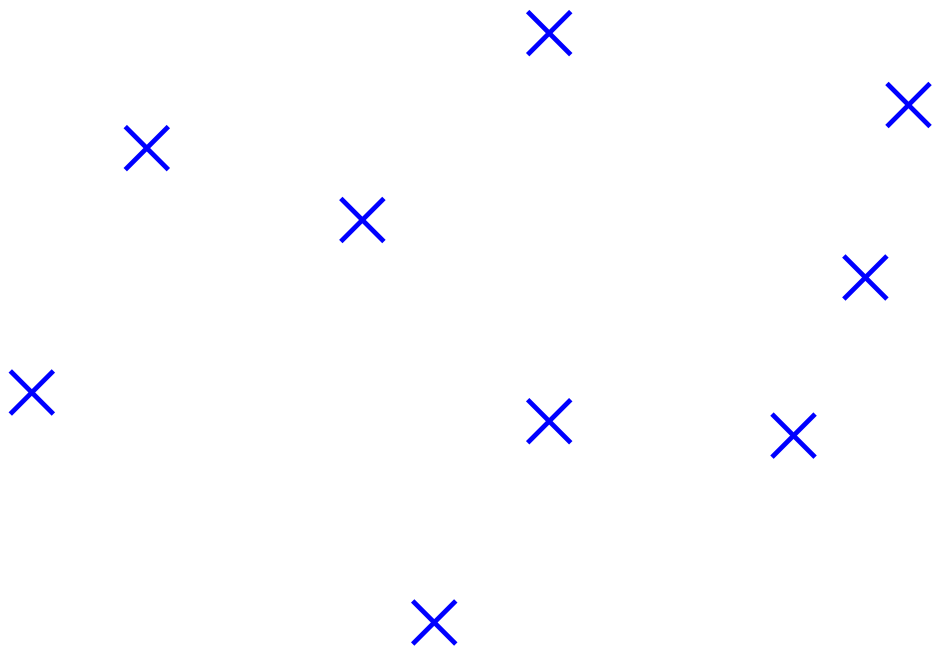
$$\mathcal{S}^\sharp_i \in \mathcal{D}^\sharp$$
$$\mathcal{S}^\sharp_0 = \top^\sharp$$
$$\mathcal{S}^\sharp_1 = F^\sharp_1(\mathcal{S}^\sharp_0)$$
$$\mathcal{S}^\sharp_2 = F^\sharp_2(\mathcal{S}^\sharp_1)$$
$$\mathcal{S}^\sharp_3 = S^\sharp_2 \cup^\sharp \mathcal{S}^\sharp_5$$
$$\mathcal{S}^\sharp_4 = F^\sharp_4(\mathcal{S}^\sharp_3)$$
$$\mathcal{S}^\sharp_5 = F^\sharp_5(\mathcal{S}^\sharp_4)$$
$$\mathcal{S}^\sharp_6 = F^\sharp_6(\mathcal{S}^\sharp_3)$$

semantics

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

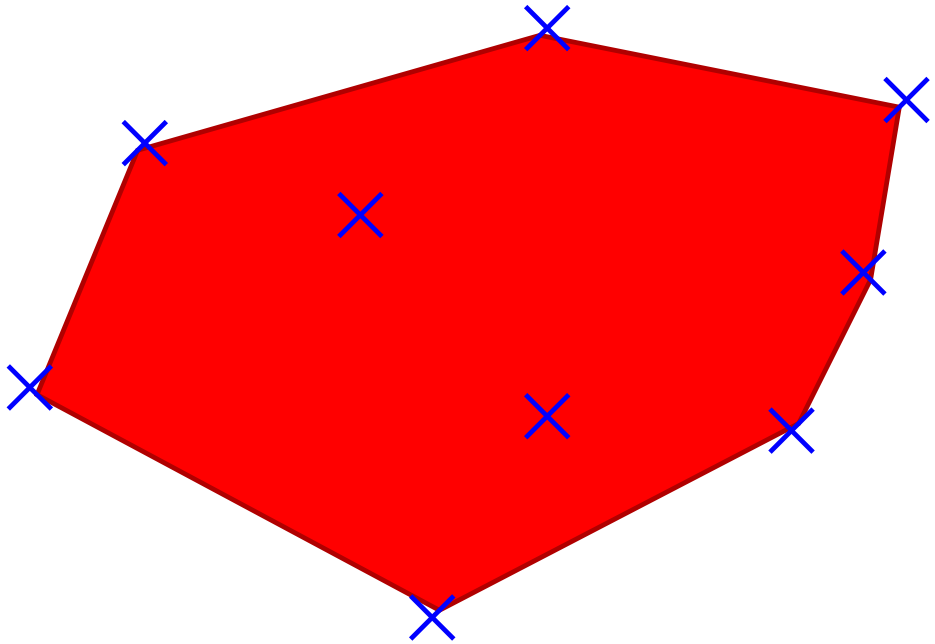
- $\mathcal{D}^\sharp$ is a subset of properties of interest with a machine representation (approximation)
- $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ over-approximates the effect of $F : \mathcal{D} \to \mathcal{D}$ in $\mathcal{D}^\sharp$ (with effective algorithms)

# Numeric abstract domain examples



concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

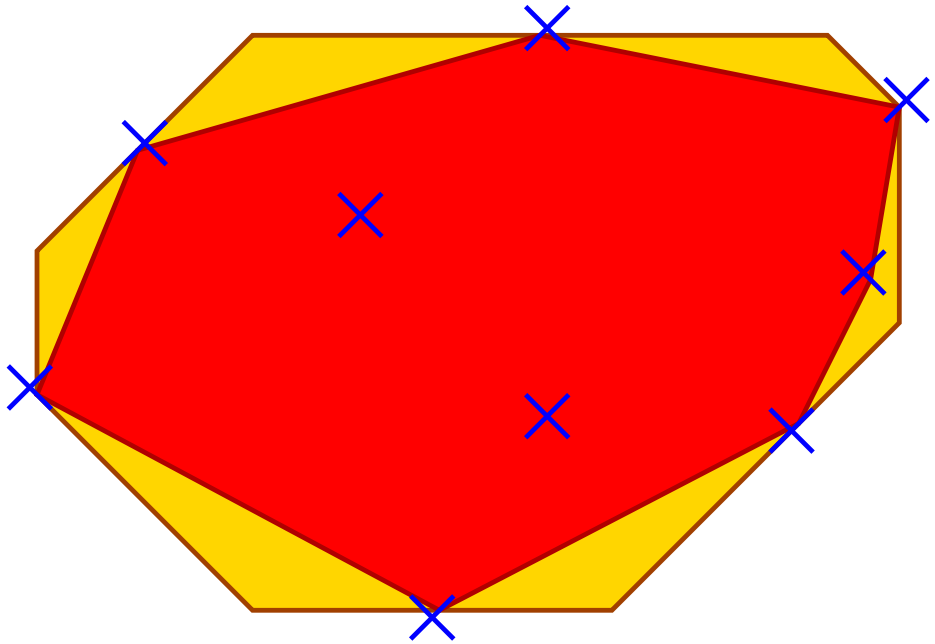
# Numeric abstract domain examples



concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}_p^\sharp$: $6X + 11Y \geq 33 \wedge \cdots$

# Numeric abstract domain examples

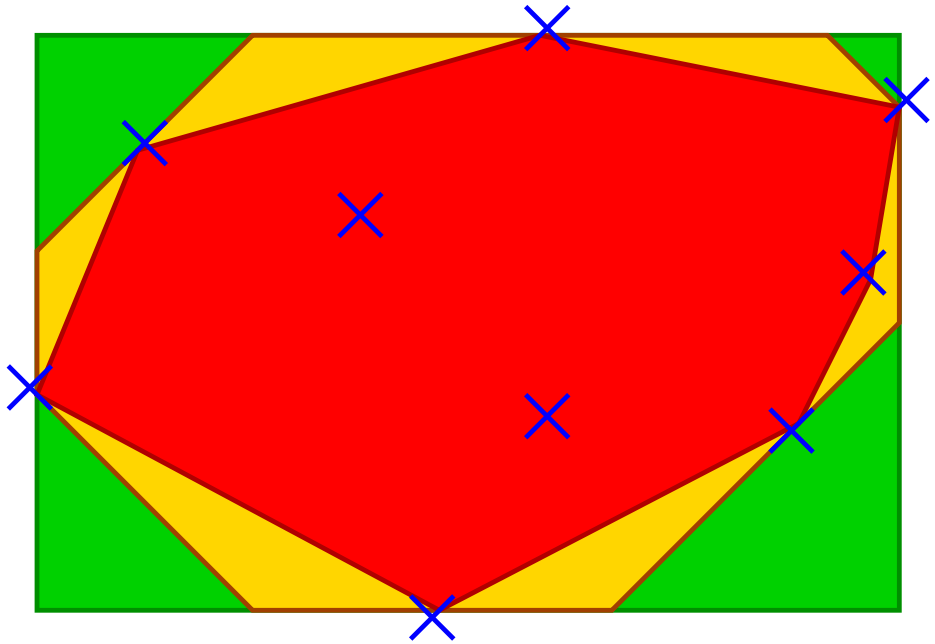


concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}_p^\sharp$: $6X + 11Y \geq 33 \wedge \cdots$

abstract octagons $\mathcal{D}_o^\sharp$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$

# Numeric abstract domain examples



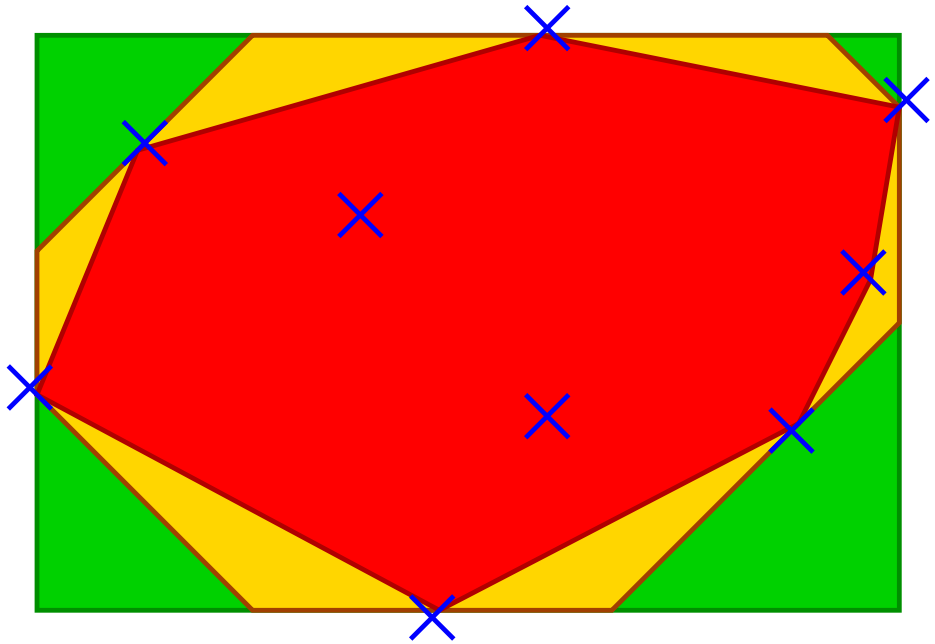
concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}^\sharp_p$: $6X + 11Y \geq 33 \wedge \cdots$

abstract octagons $\mathcal{D}^\sharp_o$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$

abstract intervals $\mathcal{D}^\sharp_i$: $X \in [0,12] \wedge Y \in [0,8]$

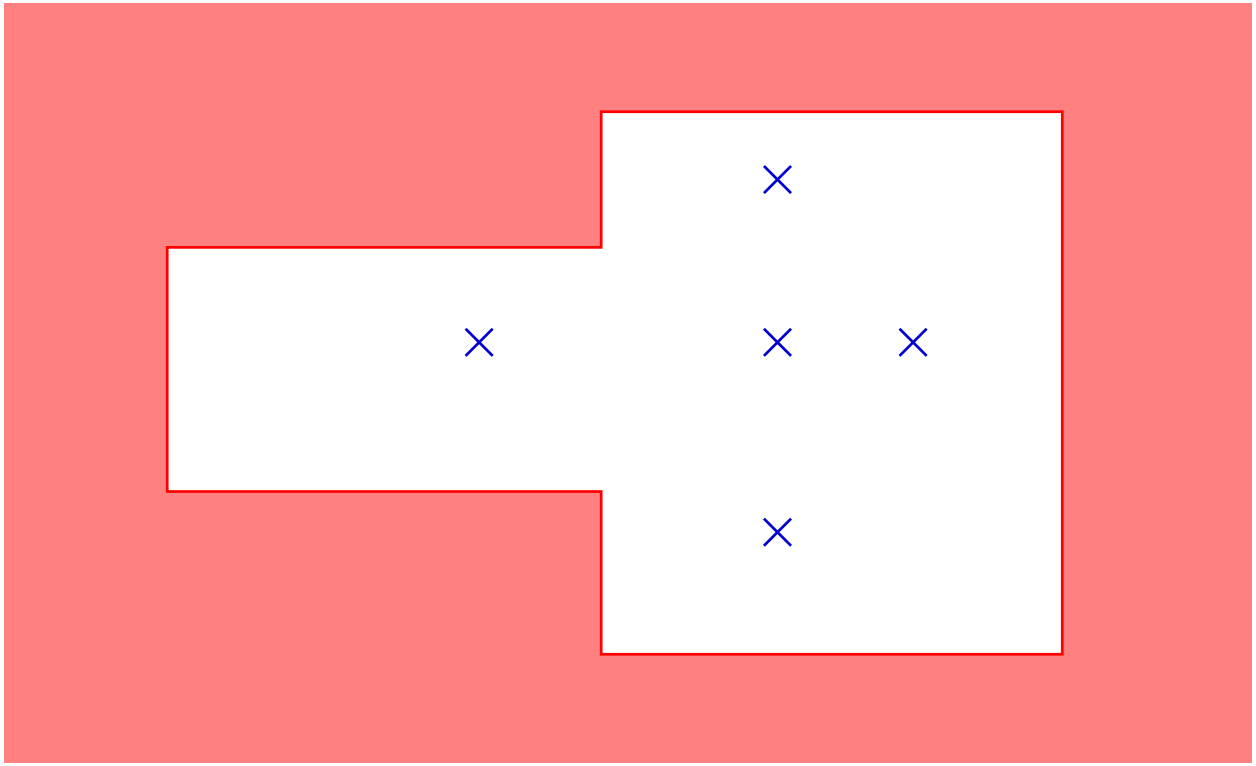
# Numeric abstract domain examples



| | | |
|---|---|---|
| concrete sets $\mathcal{D}$: | $\{(0,3),(5.5,0),(12,7),\ldots\}$ | not computable |
| abstract polyhedra $\mathcal{D}^\sharp_p$: | $6X + 11Y \geq 33 \wedge \cdots$ | exponential cost |
| abstract octagons $\mathcal{D}^\sharp_o$: | $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$ | cubic cost |
| abstract intervals $\mathcal{D}^\sharp_i$: | $X \in [0,12] \wedge Y \in [0,8]$ | linear cost |

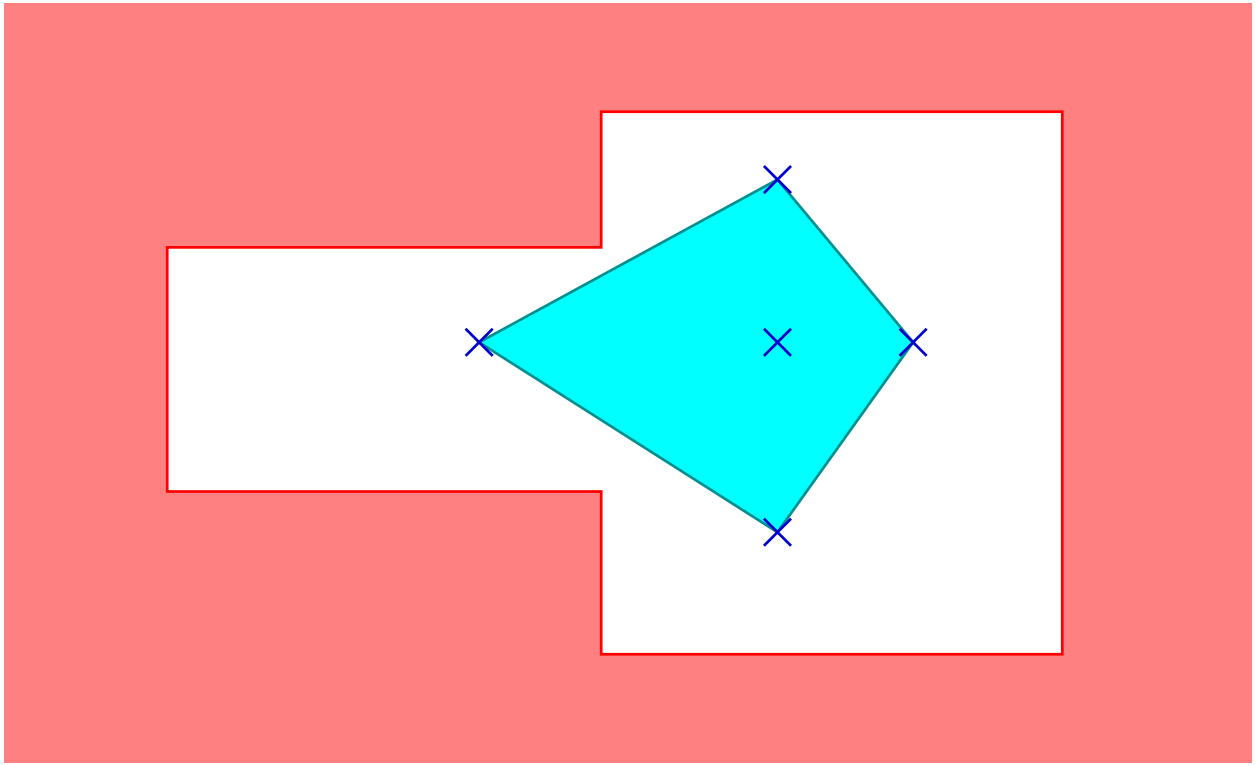
Trade-off between cost and expressiveness / precision

# Correctness proof and false alarms



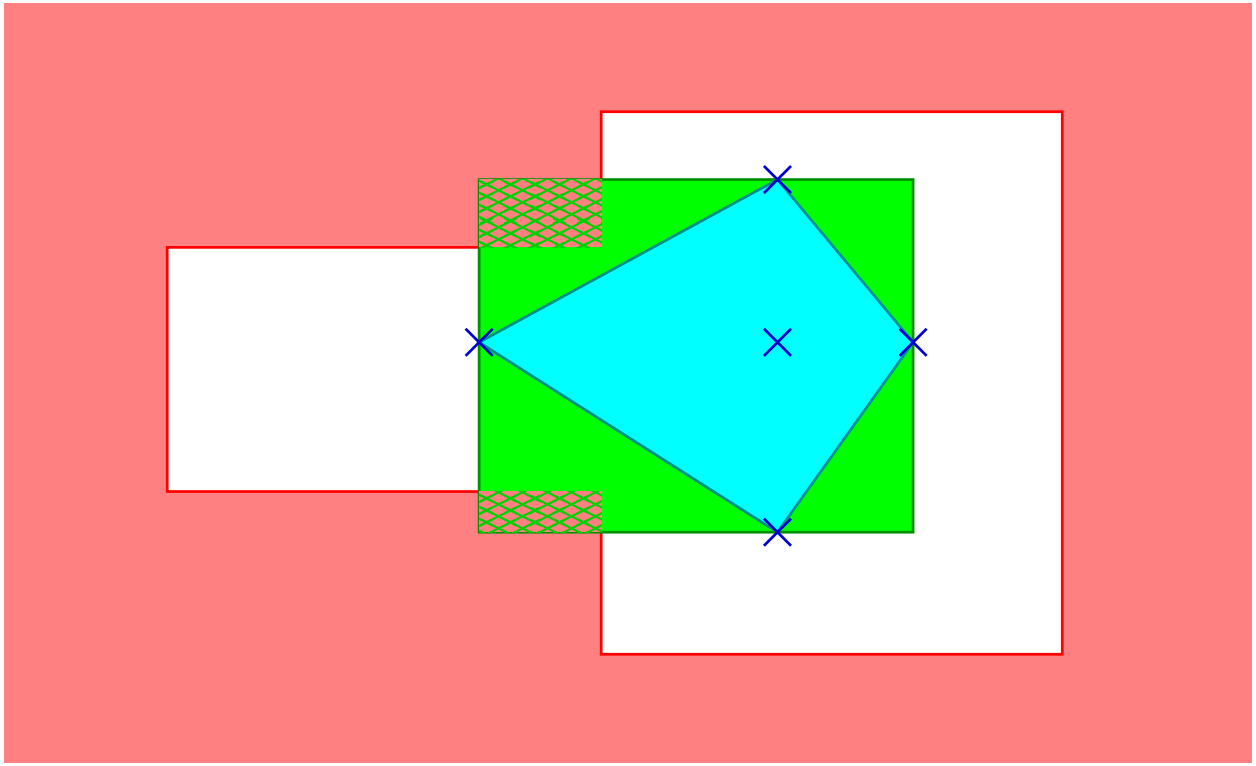
The program is correct (blue $\cap$ red $= \emptyset$).

# Correctness proof and false alarms



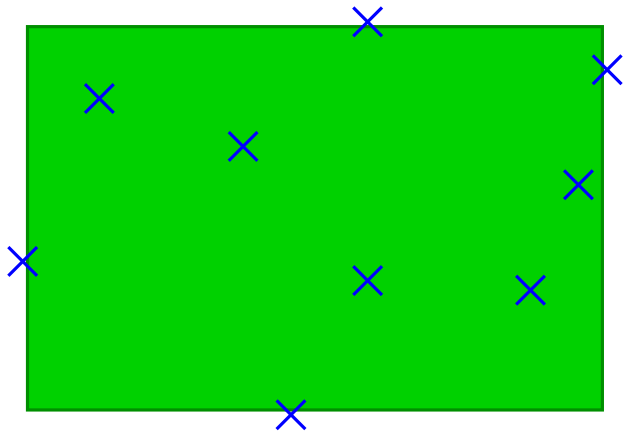
The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).

# Correctness proof and false alarms



The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).
The interval domain cannot (green $\cap$ red $\neq \emptyset$, false alarm).

# Numeric abstract domain examples (cont.)



abstract semantics $F^\sharp$ in the interval domain $\mathcal{D}_i^\sharp$

- $I \in \mathcal{D}_i^\sharp$ is a pair of bounds $(\ell, h) \in \mathbb{Z}^2$ (for each variable) representing an interval $[\ell, h] \subseteq \mathbb{Z}$
- `I:=I+2`: $(\ell, h) \mapsto (\ell+2, h+2)$
- $\cup^\sharp$: $(\ell_1, h_1) \cup^\sharp (\ell_2, h_2) = (\min(\ell_1, \ell_2), \max(h_1, h_2))$
- ...

# Resolution by iteration and extrapolation

Challenge: the equation system is recursive: $\vec{\mathcal{S}}^\sharp = \vec{F}^\sharp(\vec{\mathcal{S}}^\sharp)$.

Solution: resolution by iteration: $\vec{\mathcal{S}}^{\sharp\,0} = \emptyset^\sharp$, $\vec{\mathcal{S}}^{\sharp\,i+1} = \vec{F}^\sharp(\vec{\mathcal{S}}^{\sharp\,i})$.

e.g., $\mathcal{S}_3^\sharp$ : $\mathtt{I} \in \emptyset$, $\mathtt{I} = 0$, $\mathtt{I} \in [0, 2]$, $\mathtt{I} \in [0, 4]$, $\ldots$, $\mathtt{I} \in [0, 1000]$

# Resolution by iteration and extrapolation

Challenge: the equation system is recursive: $\vec{\mathcal{S}}^\sharp = \vec{F}^\sharp(\vec{\mathcal{S}}^\sharp)$.

Solution: resolution by iteration: $\vec{\mathcal{S}}^{\sharp\,0} = \emptyset^\sharp$, $\vec{\mathcal{S}}^{\sharp\,i+1} = \vec{F}^\sharp(\vec{\mathcal{S}}^{\sharp\,i})$.

e.g., $\mathcal{S}^\sharp_3$ : $\mathtt{I} \in \emptyset$, $\mathtt{I} = 0$, $\mathtt{I} \in [0, 2]$, $\mathtt{I} \in [0, 4]$, $\ldots$, $\mathtt{I} \in [0, 1000]$

Challenge: infinite or very long sequence of iterates in $\mathcal{D}^\sharp$

Solution: extrapolation operator $\triangledown$

e.g., $[0, 2] \triangledown [0, 4] = [0, +\infty[$

- remove unstable bounds and constraints
- ensures the convergence in finite time
- inductive reasoning (through generalisation)

# Resolution by iteration and extrapolation

Challenge: the equation system is recursive: $\vec{\mathcal{S}}^\sharp = \vec{F}^\sharp(\vec{\mathcal{S}}^\sharp)$.

Solution: resolution by iteration: $\vec{\mathcal{S}}^{\sharp\,0} = \emptyset^\sharp$, $\vec{\mathcal{S}}^{\sharp\,i+1} = \vec{F}^\sharp(\vec{\mathcal{S}}^{\sharp\,i})$.

e.g., $\mathcal{S}_3^\sharp$ : $\mathtt{I} \in \emptyset$, $\mathtt{I} = 0$, $\mathtt{I} \in [0, 2]$, $\mathtt{I} \in [0, 4]$, $\ldots$, $\mathtt{I} \in [0, 1000]$

Challenge: infinite or very long sequence of iterates in $\mathcal{D}^\sharp$

Solution: extrapolation operator $\triangledown$

e.g., $[0, 2] \triangledown [0, 4] = [0, +\infty[$

- remove unstable bounds and constraints
- ensures the convergence in finite time
- inductive reasoning (through generalisation)

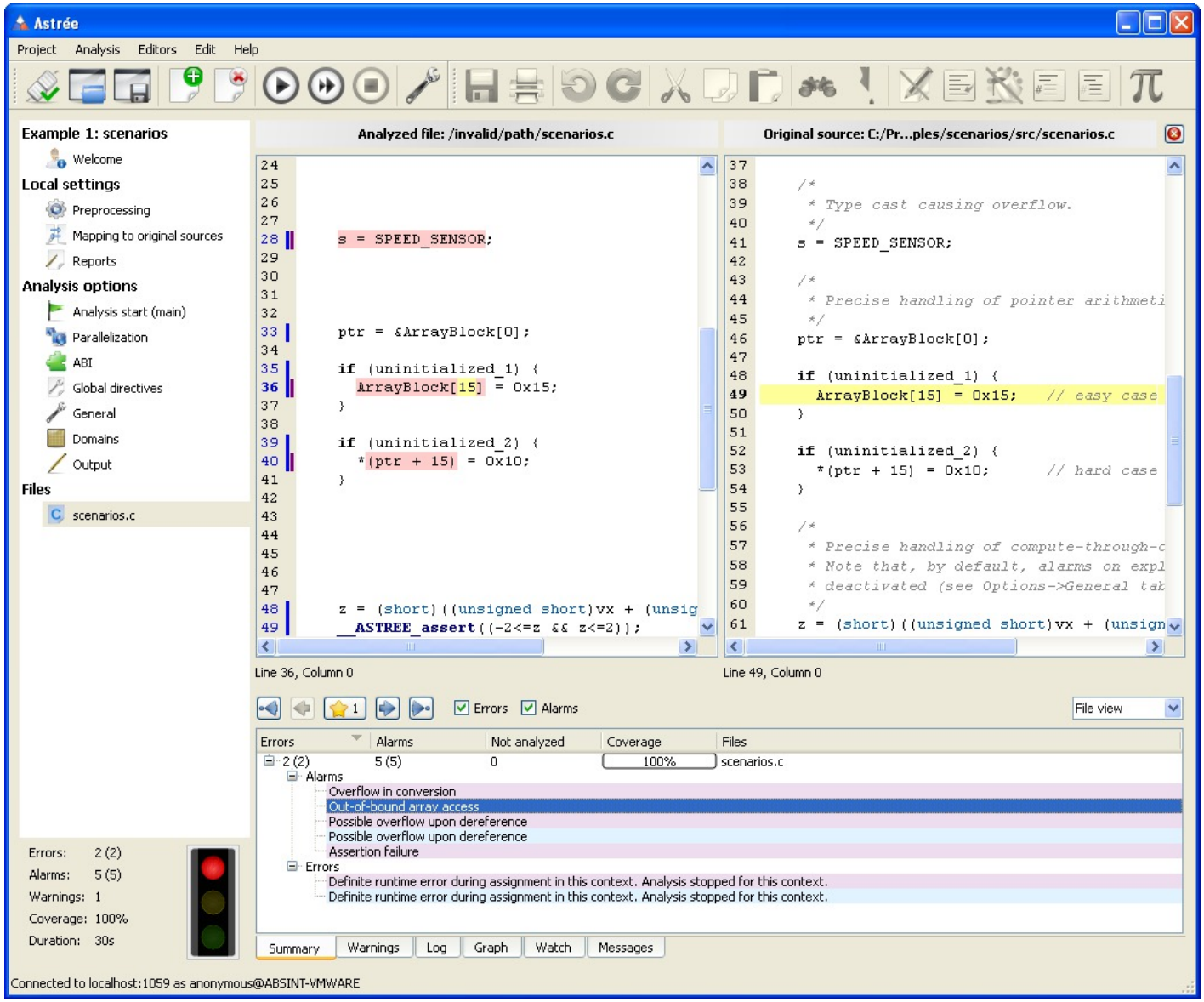
$\Longrightarrow$ effective solving method $\longrightarrow$ static analyzer!

# Other uses of abstract interpretation

- Analysis of dynamic memory data-structures (*shape analysis*).
- Analysis of parallel, distributed, and multi-thread programs.
- Analysis of probabilistic programs.
- Analysis of biological systems.
- Security analysis (*information flow*).
- Termination analysis.
- Cost analysis.
- Analyses to enable compiler optimisations.
- ...

# Some static analysis tools based on Abstract Interpretation

# The Astrée static analyzer



# The Astrée static analyzer

**Analyseur statique de programmes temps-réels embarqués**

(static analyzer for real-time embedded software)

- developed at ENS
  > B. Blanchet, P. Cousot, R. Cousot, J. Feret,
  > L. Mauborgne, D. Monniaux, A. Miné, X. Rival
- industrialized and made commercially available by AbsInt

Astrée

`www.astree.ens.fr`

**AbsInt**

`www.absint.com`

# The Astrée static analyzer

Specialized:

- for the analysis of run-time errors

  (arithmetic overflows, array overflows, divisions by 0, etc.)

- on embedded critical C software

  (no dynamic memory allocation, no recursivity)

- in particular on control / command software

  (reactive programs, intensive floating-point computations)

- intended for validation

  (analysis does not miss any error and tries to minimise false alarms)

# The Astrée static analyzer

Specialized:

- for the analysis of run-time errors
  (arithmetic overflows, array overflows, divisions by 0, etc.)
- on embedded critical C software
  (no dynamic memory allocation, no recursivity)
- in particular on control / command software
  (reactive programs, intensive floating-point computations)
- intended for validation
  (analysis does not miss any error and tries to minimise false alarms)

Approximately 40 abstract domains are used at the same time:

- numeric domains (intervals, octagons, ellipsoids, etc.)
- boolean domains
- domains expressing properties on the history of computations

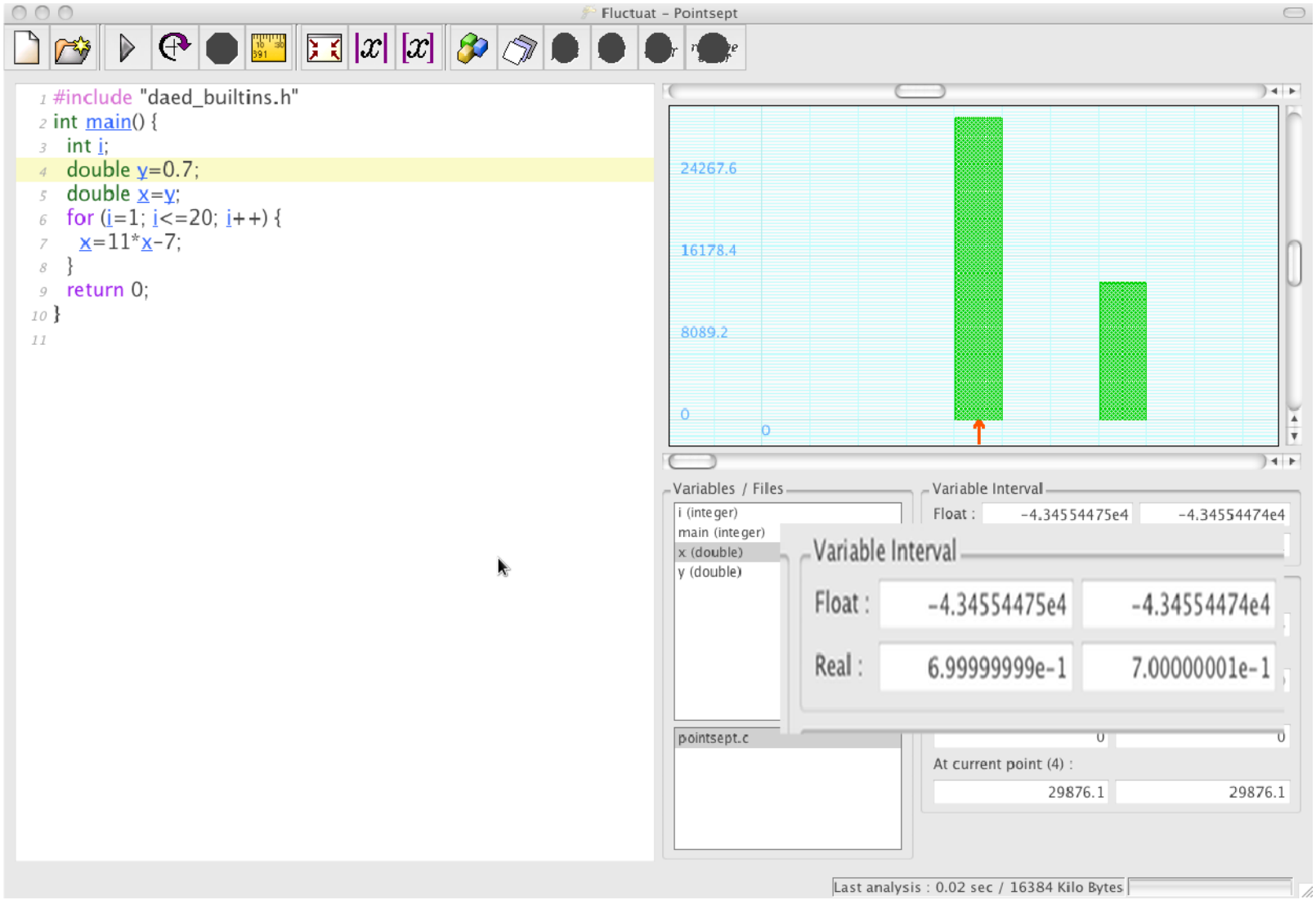
# Astrée applications


Airbus A340-300 (2003)


Airbus A380 (2004)

- size: from 70 000 to 860 000 lines of C
- analysis time: from 45mn to $\simeq$40h
- 0 alarm: proof of absence of run-time error

# Fluctuat


Fluctuat – Pointsept
#include "daed_builtins.h"
int main() {
int i;
double y=0.7;
double x=y;
for (i=1; i<=20; i++) {
x=11*x-7;
}
return 0;
}
24267.6
16178.4
8089.2
0
0
Variables / Files
i (integer)
main (integer)
x (double)
y (double)
pointsept.c
Variable Interval
Float :
-4.34554475e4
-4.34554474e4
Real :
6.99999999e-1
7.00000001e-1
At current point (4) :
29876.1
29876.1
Last analysis : 0.02 sec / 16384 Kilo Bytes

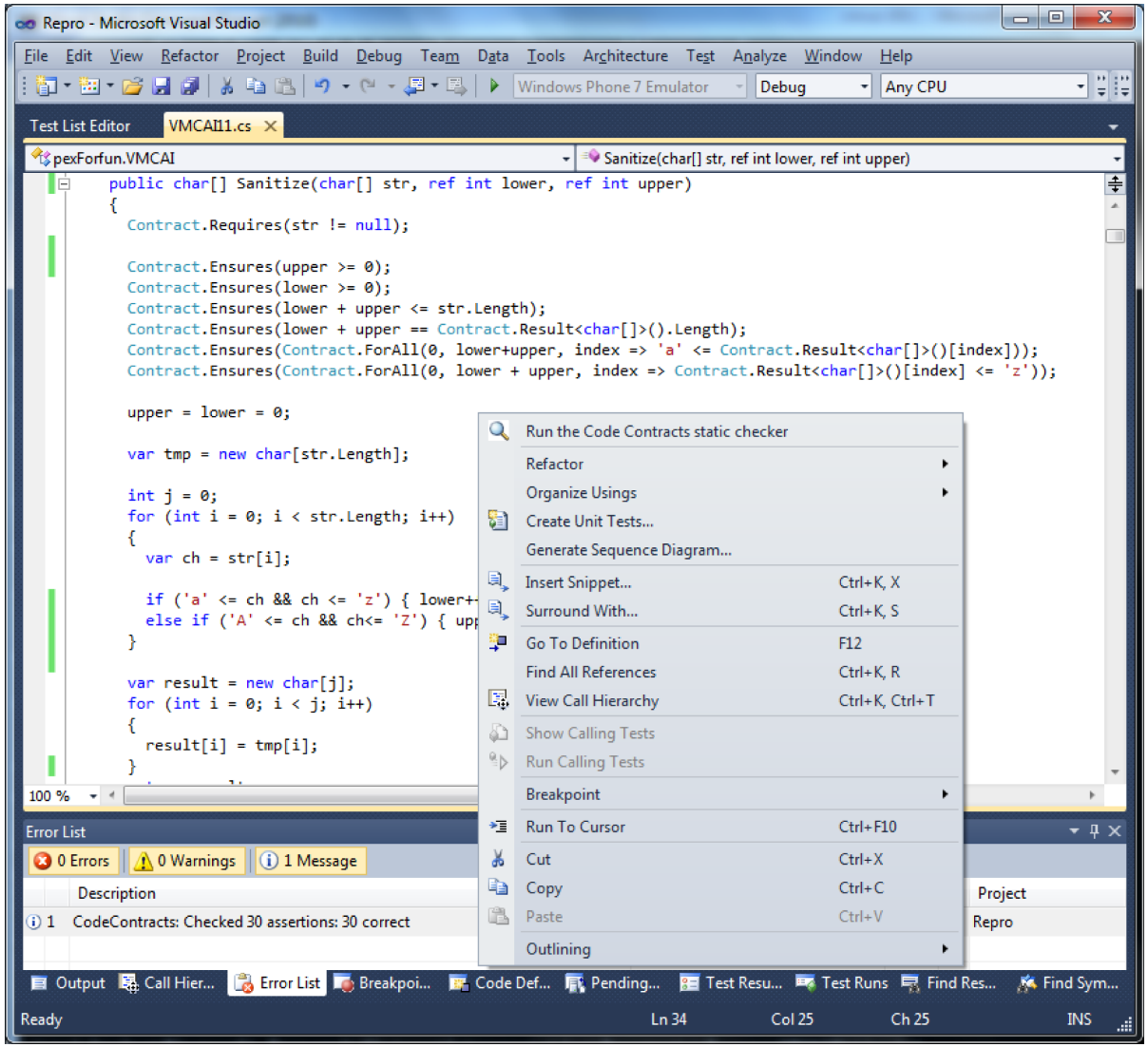
# Fluctuat

Static analysis of the **accuracy of floating-point computations**:

- bound the range of variables
- bound the rounding errors wrt. real computation
- track the origin of rounding errors
  (which operation contributes to most error,
  target for improvements)
- uses specific abstract domains
  (affine arithmetic, zonotopes)

- developed at CEA-LIST (E. Goubault, S. Putot)
- industrial use (Airbus)

# Clousot: CodeContract static checker



# Clousot: CodeContract static checker

CodeContracts:

- assertion language for .NET (C#, VB, etc.)
  (pre-conditions, post-conditions, invariants)
- dynamic checking
  (insert run-time checks)
- static checking
  (modular abstract interpretation)
- automatic inference
  (abstract interpretation to infer necessary preconditions backwards)

- developed at Microsoft Research (M. Fahndrich, F. Logozzo)
- part of .NET Framework 4.0
- integrated to Visual Studio