# Formal Verification by Abstract Interpretation

Patrick Cousot

cims.nyu.edu/~pcousot

di.ens.fr/~cousot

NFM 2012, 4th NASA Formal Methods Symposium
Norfolk, Virginia — April 3–5, 2012

# Formal Verification by Abstract Interpretation

Patrick Cousot

cims.nyu.edu/~pcousot

di.ens.fr/~cousot

joint work with Radhia Cousot

di.ens.fr/rcousot

NFM 2012, 4th NASA Formal Methods Symposium
Norfolk, Virginia — April 3–5, 2012

## Abstract

Abstract interpretation is a theory of abstraction and constructive approximation of the mathematical structures used in the formal description of programming languages and the inference or verification of undecidable program properties.

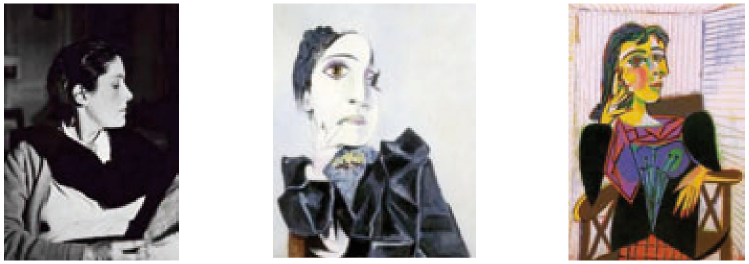
Developed in the late seventies with Radhia Cousot, it has since then been considerably applied to many aspects of programming, from syntax, to semantics, and proof methods where abstractions are sound and complete but incomputable to fully automatic, sound but incomplete approximate abstractions to solve undecidable problems such as static analysis of infinite state software systems, contract inference, type inference, termination inference, model-checking, abstraction refinement, program transformation (including watermarking), combination of decision procedures, security, malware detection, etc.

This last decade, abstract interpretation has been very successful in program verification for mission- and safety-critical systems. An example is Astrée (www.astree.ens.fr) which is a static analyzer to verify the absence of runtime errors in structured, very large C programs with complex memory usages, and involving complex boolean as well as floating-point computations (which are handled precisely and safely by taking all possible rounding errors into account), but without recursion or dynamic memory allocation. Astrée targets embedded applications as found in earth transportation, nuclear energy, medical instrumentation, aeronautics and space flight, in particular synchronous control/command such as electric flight control or more recently asynchronous systems as found in the automotive industry.

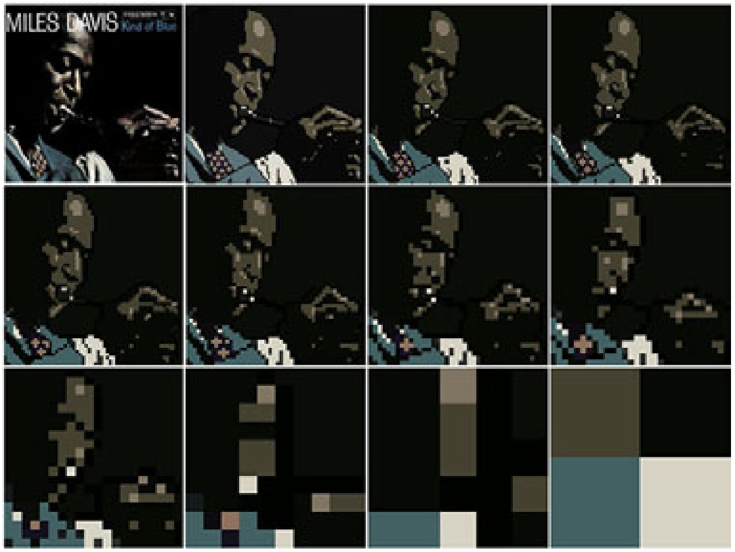
Astrée is industrialized by AbsInt (www.absint.com/astree).

# Examples of abstraction

# Abstractions of Dora Maar by Picasso



# Pixelation of a photo by Jay Maisel



/www.petapixel.com/2011/06/23/how-much-pixelation-is-needed-before-a-photo-becomes-transformed/

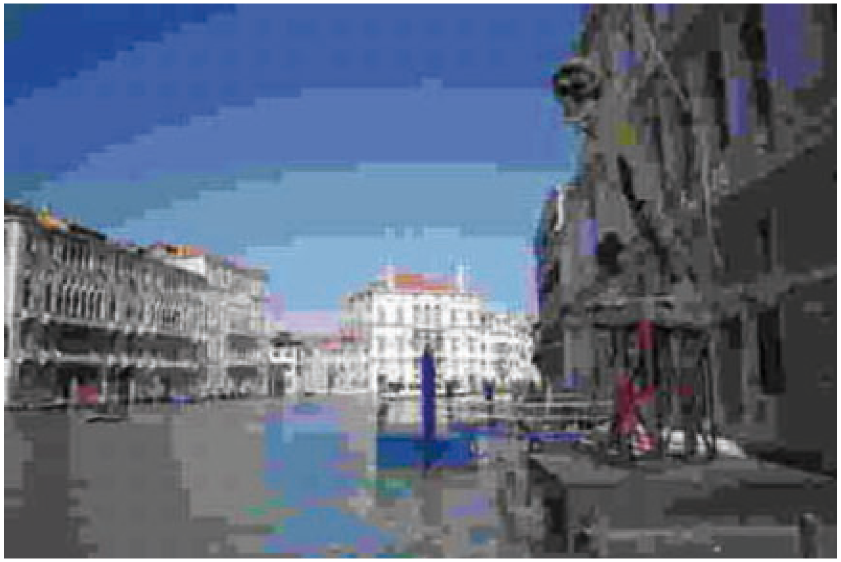
***Image credit:*** *Photograph by Jay Maisel*

# An old idea...

20 000 years old picture in a spanish cave:

The concrete is not always well-known!

# Example of picture abstraction

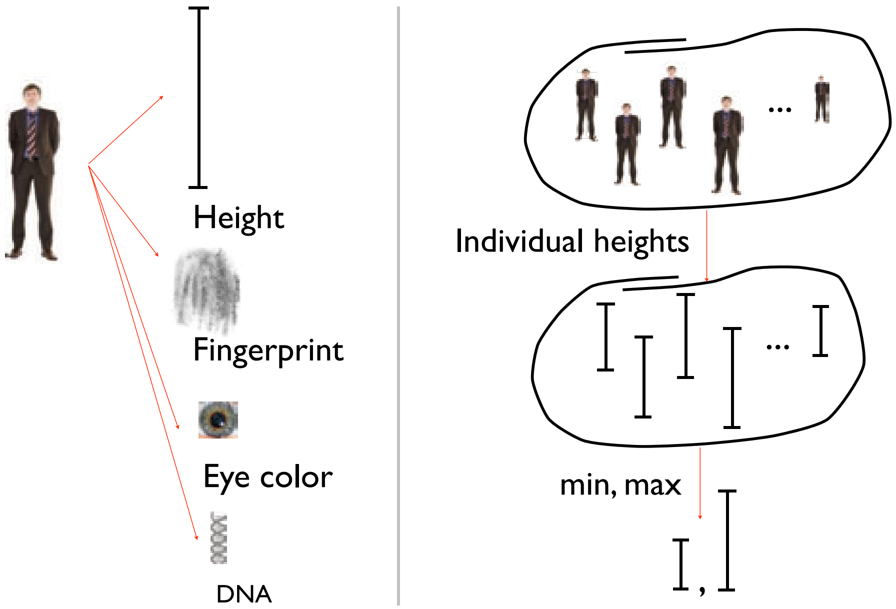


Abstraction...

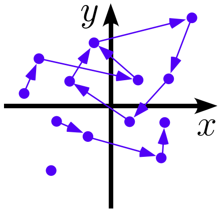
# and concretization

Grand Canal (Venice) Abstraction... which concretization is an abstract sculpture in front of the Palazzo Grassi

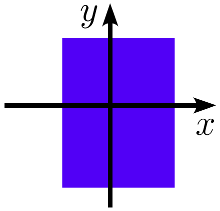
# Abstractions of a man / crowd



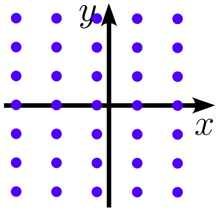
# Numerical abstractions in Astrée



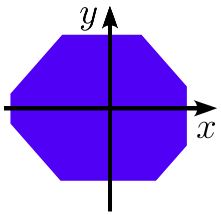
Collecting semantics: partial traces



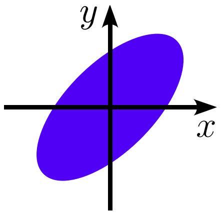
Intervals: $\mathtt{x} \in [a, b]$



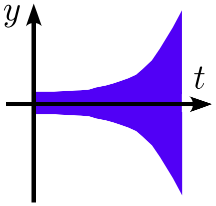
Simple congruences: $\mathtt{x} \equiv a[b]$



Octagons: $\pm\mathtt{x} \pm \mathtt{y} \leqslant a$



Ellipses: $\mathtt{x}^2 + b\mathtt{y}^2 - a\mathtt{x}\mathtt{y} \leqslant d$



Exponentials: $-a^{bt} \leqslant \mathtt{y}(t) \leqslant a^{bt}$

# Content

- Fundamental and applied motivations
- An informal introduction to abstract interpretation
- A touch of theory of abstract interpretation
- A short overview of a few applications and on-going work on software verification

For a rather complete basic introduction to abstract interpretation and applications to cyber-physical systems, see:

Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, & Xavier Rival. Static Analysis and Verification of Aerospace Software by Abstract Interpretation. In *AIAA Infotech@@Aerospace 2010*, Atlanta, Georgia. American Institute of Aeronautics and Astronautics, 20—22 April 2010. © AIAA.

# Fundamental motivations

# Scientific research

- in Mathematics/Physics:

  works towards unification and synthesis

  it is science of structure and change aiming at universal principles

- in Computer science

  works towards dispersion and parcelization

  it is a collection of local techniques for computational structures aiming at specific applications

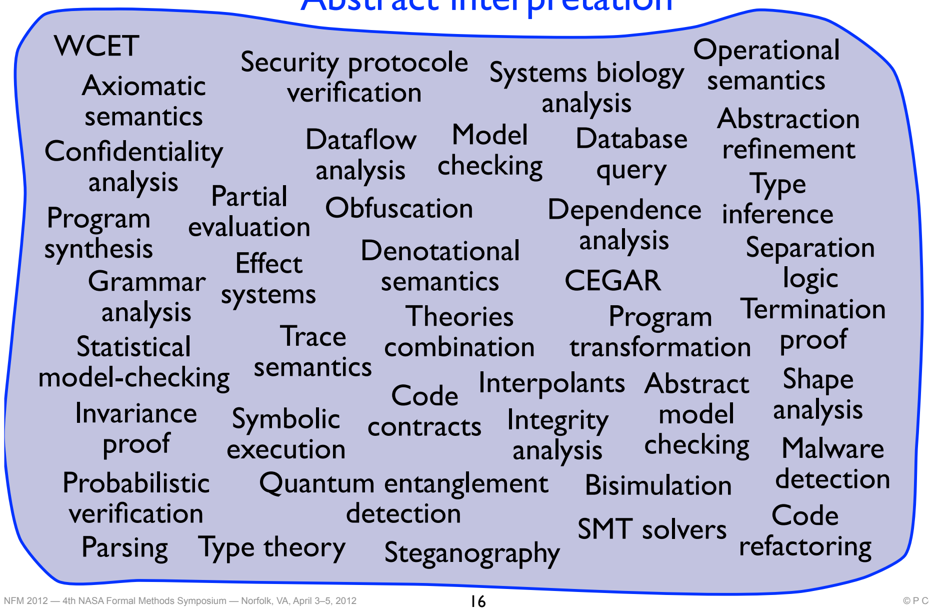
An exponential process, will stop!

# Example: reasoning on computational structures

WCET
Security protocole verification
Systems biology analysis
Operational semantics
Axiomatic semantics
Abstraction refinement
Dataflow analysis
Model checking
Database query
Confidentiality analysis
Type inference
Partial evaluation
Obfuscation
Dependence analysis
Program synthesis
Separation logic
Effect systems
Denotational semantics
CEGAR
Grammar analysis
Termination proof
Theories combination
Program transformation
Trace semantics
Statistical model-checking
Code contracts
Interpolants
Abstract model checking
Shape analysis
Invariance proof
Symbolic execution
Integrity analysis
Malware detection
Probabilistic verification
Quantum entanglement detection
Bisimulation
Code refactoring
SMT solvers
Parsing
Type theory
Steganography

# Example: reasoning on computational structures

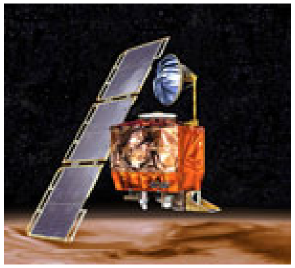
Abstract interpretation



WCET
Security protocole verification
Systems biology analysis
Operational semantics
Axiomatic semantics
Abstraction refinement
Dataflow analysis
Model checking
Database query
Confidentiality analysis
Type inference
Partial evaluation
Obfuscation
Dependence analysis
Program synthesis
Separation logic
Effect systems
Denotational semantics
CEGAR
Grammar analysis
Termination proof
Theories combination
Program transformation
Trace semantics
Statistical model-checking
Code contracts
Interpolants
Abstract model checking
Shape analysis
Invariance proof
Symbolic execution
Integrity analysis
Malware detection
Probabilistic verification
Quantum entanglement detection
Bisimulation
Code refactoring
SMT solvers
Parsing
Type theory
Steganography

# Applied motivations

## All computer scientists have experienced bugs



- Checking the presence of bugs is great
- Proving their absence is even better!

# Abstract interpretation

Patrick Cousot & Radhia Cousot. Vérification statique de la cohérence dynamique des programmes. In Rapport du contrat IRIA SESORI No 75-035, Laboratoire IMAG, University of Grenoble, France. 125 pages. 23 September 1975.

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Patrick Cousot & Radhia Cousot. Static Determination of Dynamic Properties of Programs. In B. Robinet, editor, Proceedings of the 2nd international symposium on Programming, 106—130, 1976, Dunod, Paris.

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Patrick 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, Université Joseph Fourier, Grenoble, France, 21 March 1978

Patrick Cousot. Semantic foundations of program analysis. In S.S. Muchnick & N.D. Jones, editors, Program Flow Analysis: Theory and Applications, Ch. 10, pages 303—342, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

## Abstract interpretation

- *Started in the 70's* and widely applied since then
- Based on the idea that undecidability and complexity of automated program analysis can be fought by *sound approximations* or *complete abstractions*
- Wide-spectrum theory so applications range from *static analysis* to *verification* to *biology*
- ***Does scale up!***

## Fighting undecidability and complexity in practical program verification

- Any *automatic* program verification method will definitely fail on infinitely many programs (Gödel)
- Solutions (excluding non-termination):
  - Ask for human help (theorem-prover/proof assistant based *deductive methods*)
  - Consider finite systems (*model-checking*) which are small enough to avoid combinatorial explosion
  - Do sound approximations or complete abstractions (*abstract interpretation*) which are precise enough to avoid false alarms

# An informal introduction to abstract interpretation

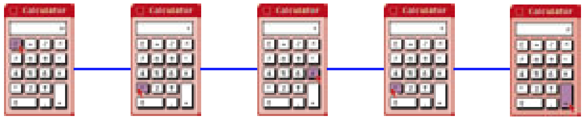
P. Cousot & R. Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In *Logics and Languages for Reliability and Security*, J. Esparza, O. Grumberg, & M. Broy (Eds), NATO Science Series III: Computer and Systems Sciences, © IOS Press, 2010, Pages 1—29.

# An informal introduction to abstract interpretation (a) Principle

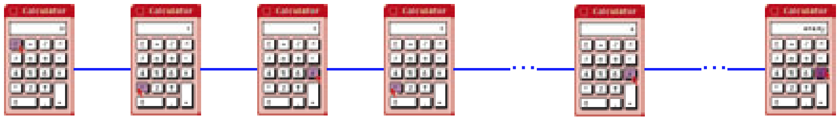
P. Cousot & R. Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In *Logics and Languages for Reliability and Security*, J. Esparza, O. Grumberg, & M. Broy (Eds), NATO Science Series III: Computer and Systems Sciences, © IOS Press, 2010, Pages 1—29.

## 1) Define the programming language semantics

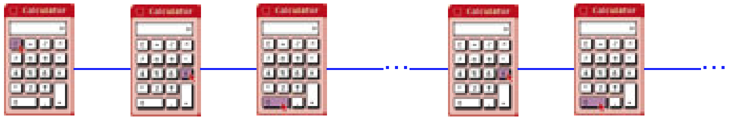
– Finite (C1+1=) :



– Erroneous (C1+1+1+1...) :

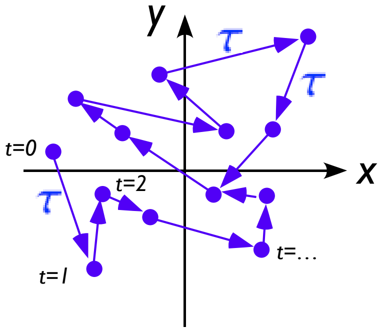


– Infinite (C0+0+0+0+...) :

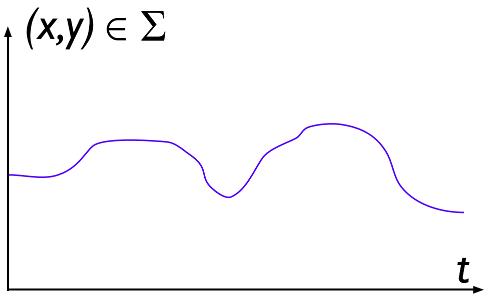


# Formal concrete semantics

*Formalize what you are interested to* ***observe*** *about concrete program behaviors (e.g. execution traces of a transition system)*

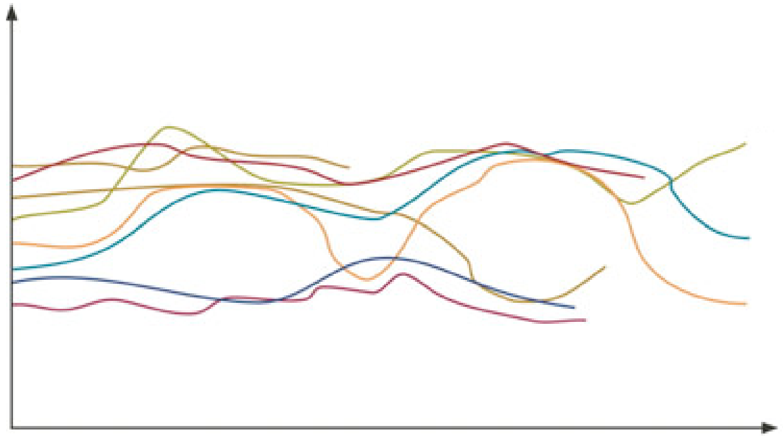


Trajectory
in state space Σ



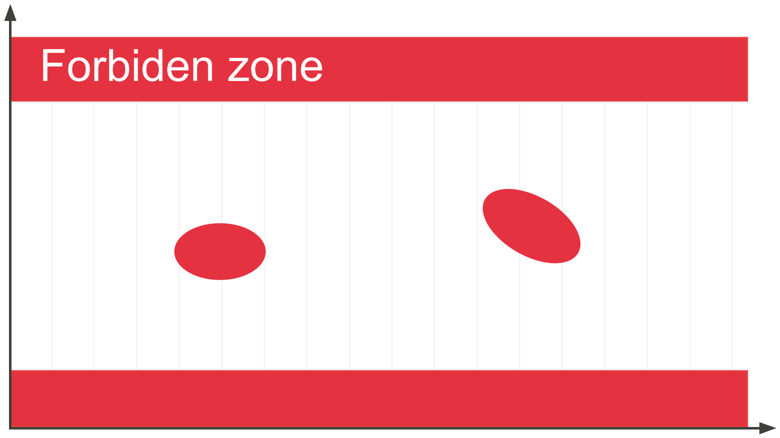
Space/time trajectory

# Formal concrete semantics (cont'd)



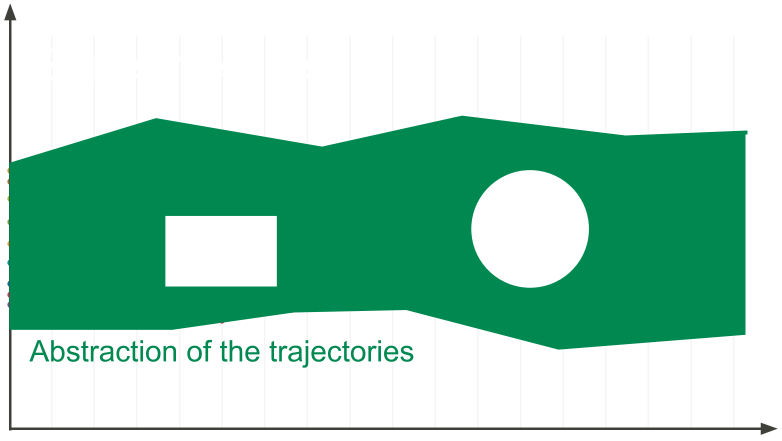
# II) Define which specification must be checked

*Formalize what you are interested to* ***prove*** *about program behaviors*



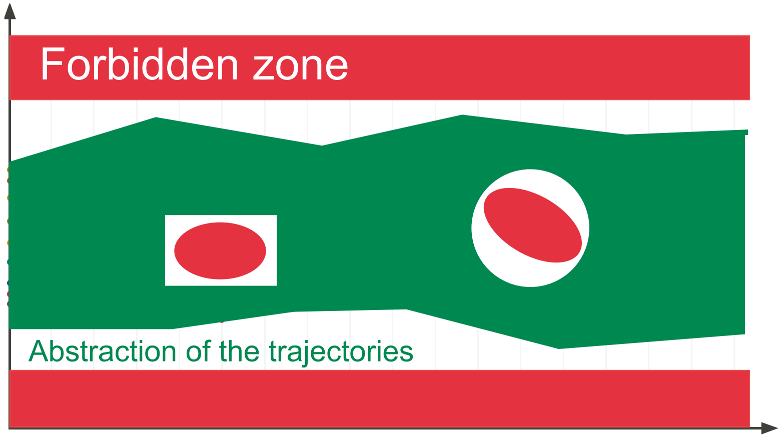
# III) Choose an appropriate abstraction

***Abstract*** *away all information on program behaviors irrelevant to the proof*



# IV) Mechanically verify in the abstract

*The proof is fully **automatic** in finite time*

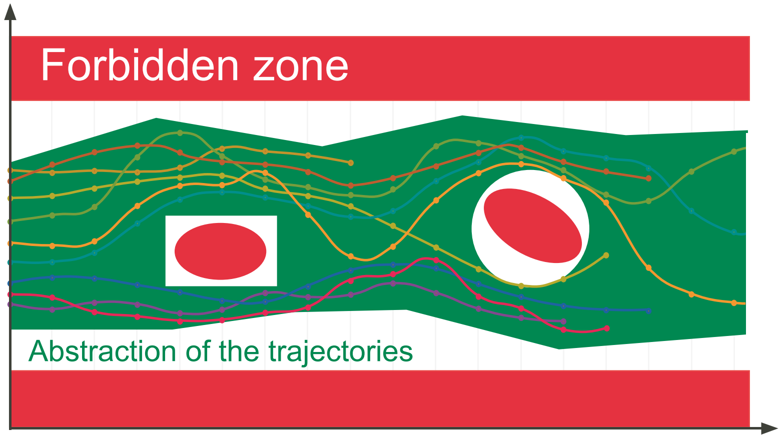


# An informal introduction to abstract interpretation

# (b) [Un]soundness

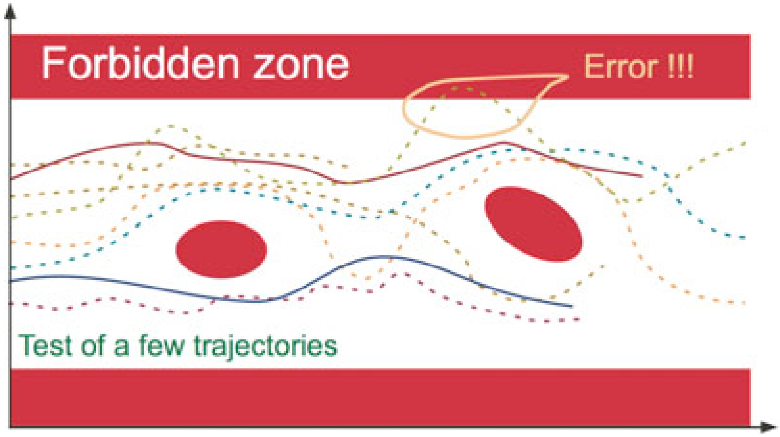
# Soundness of the abstract verification

*Never forget any possible case so the **abstract proof is correct in the concrete***



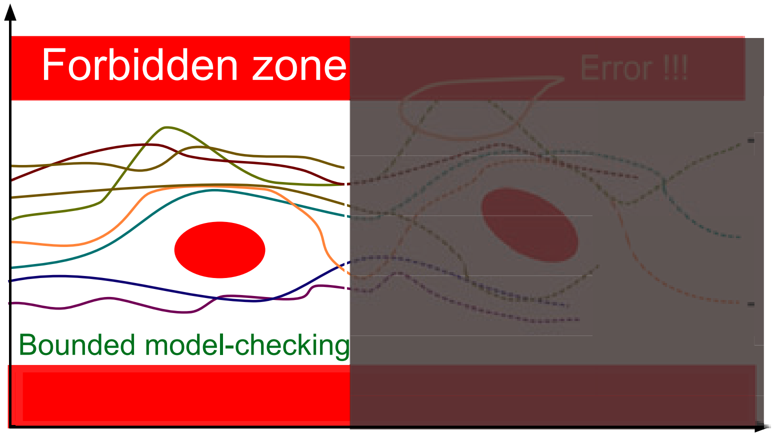
# Unsound validation: testing

*Try a few cases*



## Unsound validation: bounded abstraction

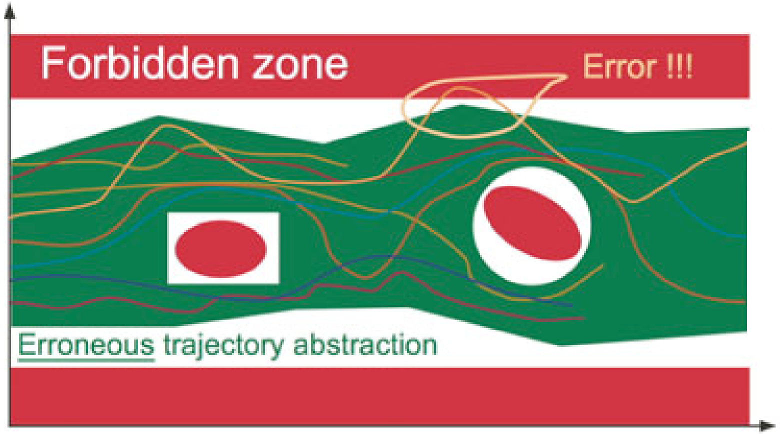
*Simulate the beginning of all executions*



Examples: bounded model-checking, symbolic execution, ...

## Unsound validation: incorrect static analysis

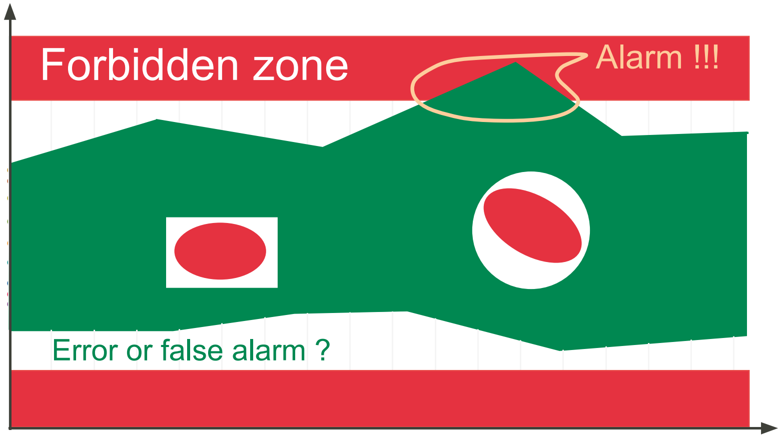
*Many static analysis tools are **unsound** (e.g. Coverity, etc.) so inconclusive*



# An informal introduction to abstract interpretation (c) Incompleteness

## Incompleteness

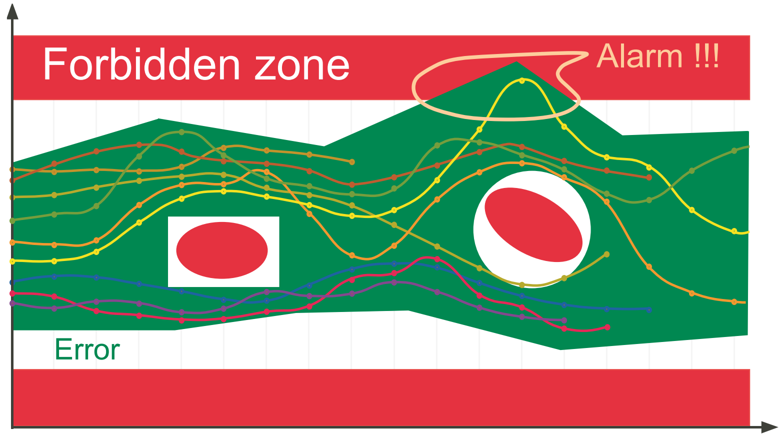
*When abstract proofs may fail while concrete proofs would succeed*



*By soundness an alarm must be raised for this overapproximation!*

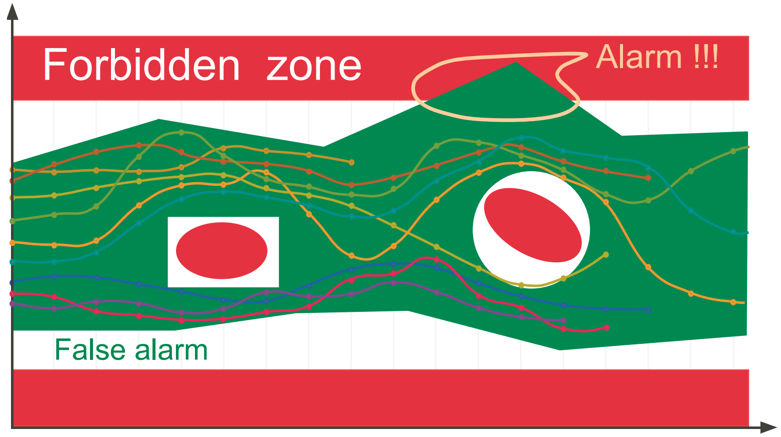
# True error

*The abstract alarm may correspond to a concrete error*



# False alarm

*The abstract alarm may correspond to no concrete error (false negative)*

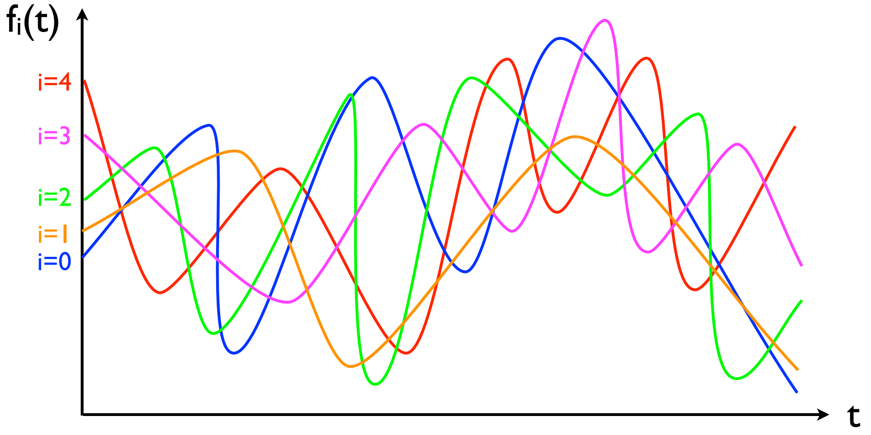


# What to do about false alarms: refinement

# What to do about false alarms? (I) Automatic refinement

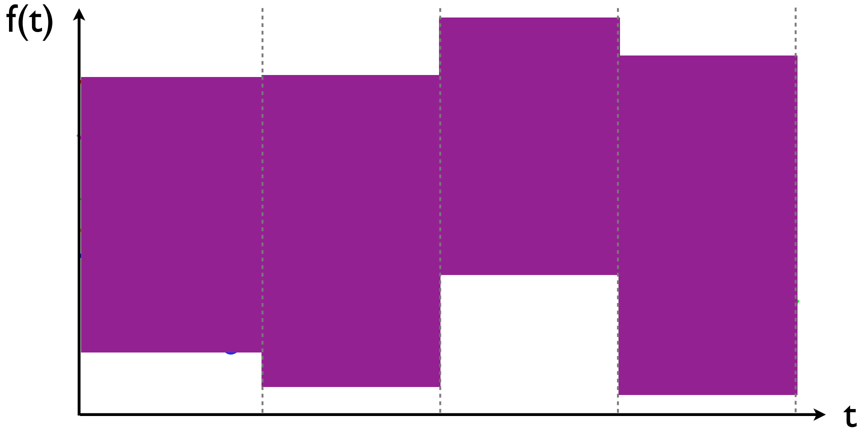
- Inefficient and may not terminate (Gödel)
- Refinement needs intelligence

# Set of functions

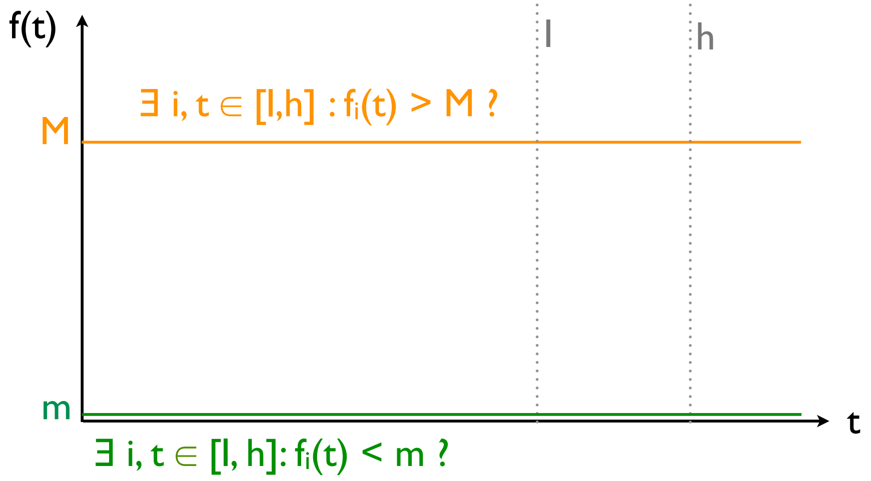


How to approximate { $f_1$, $f_2$, $f_3$, $f_4$ } ?

# Set of functions abstraction



# Concrete questionsthe $f_i$

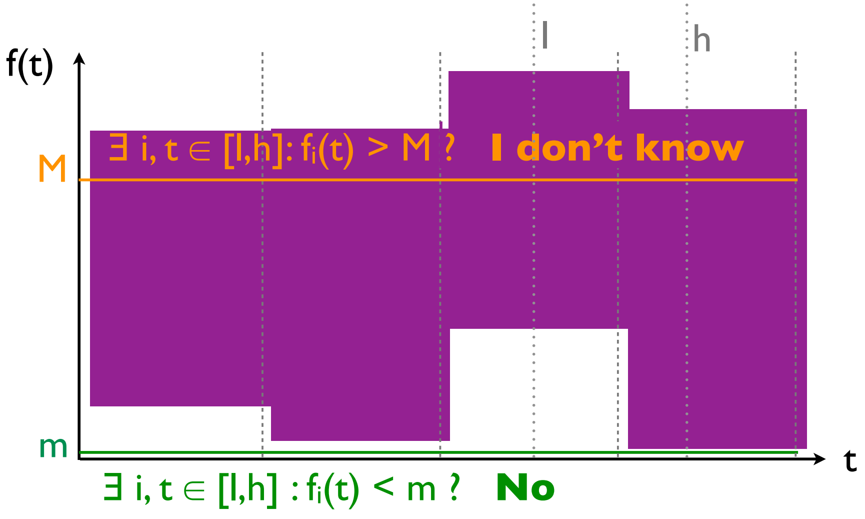


$\exists$ i, t $\in$ [l,h] : $f_i(t) > M$ ?

$\exists$ i, t $\in$ [l, h]: $f_i(t) < m$ ?

Min/max questions on the $f_i$

# Concrete questions answered in the

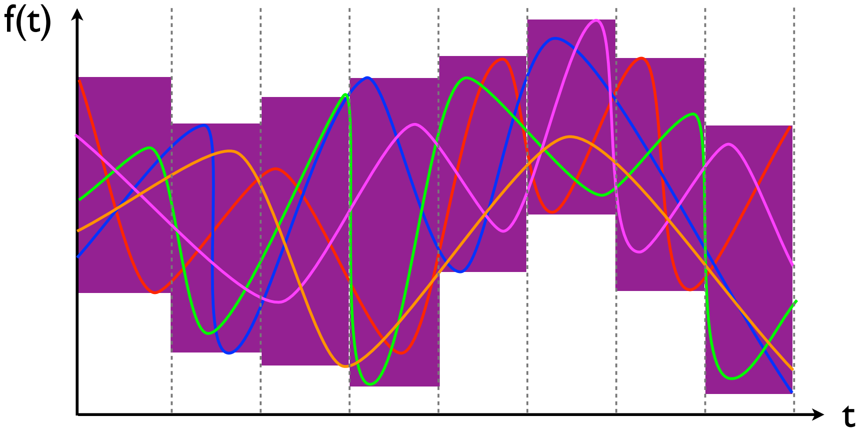


$\exists$ i, t $\in$ [l,h]: $f_i(t) > M$ ? **I don't know**

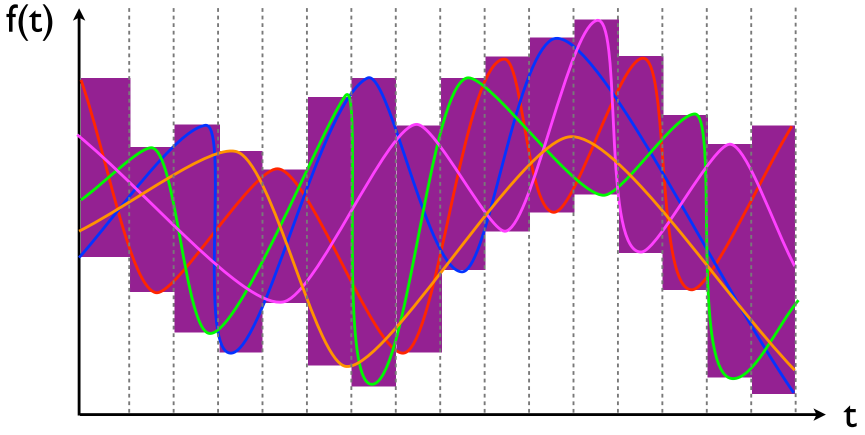
$\exists$ i, t $\in$ [l,h] : $f_i(t) < m$ ? **No**

Min/max questions on the $f_i$

# A more precise/refined abstraction

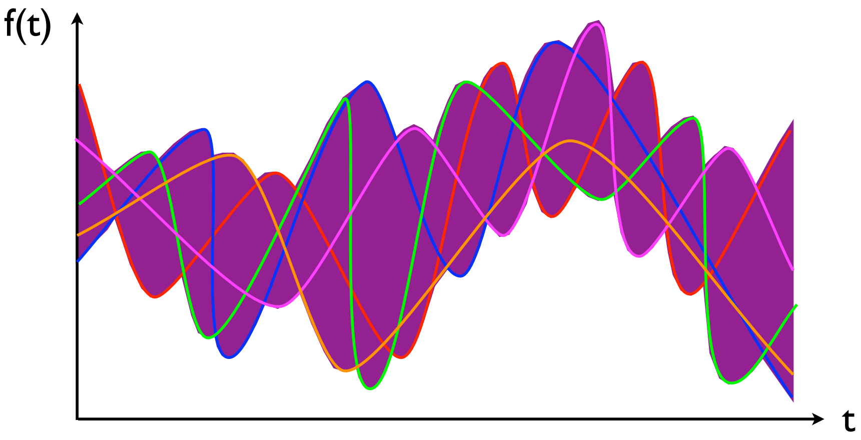


# An even more precise/refined abstraction



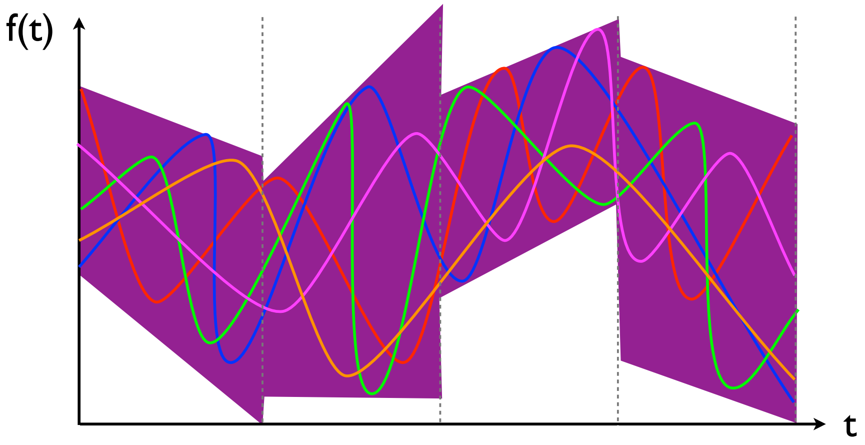
Note: this is already much more elaborate than CEGAR that goes counter-example by counter-example!

# *Intelligent* passing to the limit



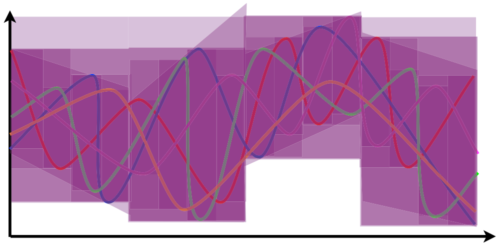
Sound and *complete* abstraction for min/max questions on the $f_i$

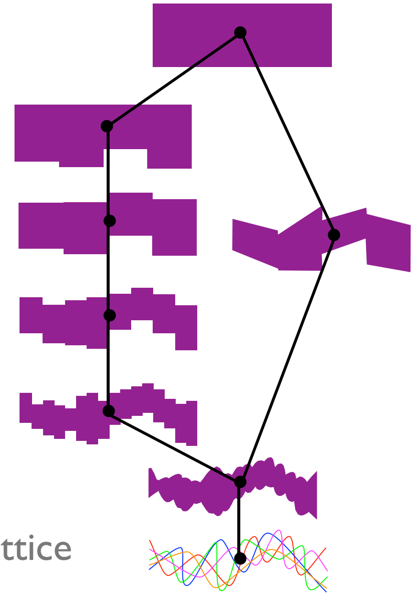
# A non-comparable abstraction



Sound and *<u>incomplete</u>* abstraction for min/max questions on the $f_i$

# The hierarchy of abstractions





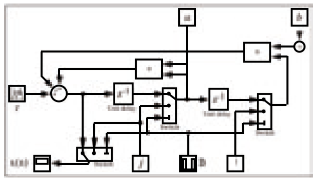
A complete lattice

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

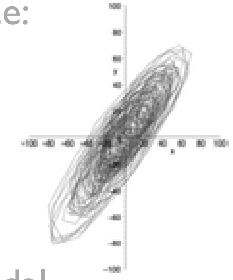
# (I) Automatic refinement: Astrée example

- Filter invariant abstraction:

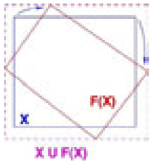
2nd order filter:



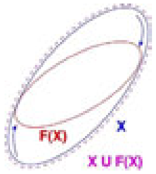
Execution trace:



Unstable polyhedral abstraction:



Stable ellipsoidal abstraction:



Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, & Xavier Rival. Static Analysis and Verification of Aerospace Software by Abstract Interpretation. In *AIAA Infotech@@Aerospace 2010*, Atlanta, Georgia. American Institute of Aeronautics and Astronautics, 20—22 April 2010. © AIAA.

# What to do about false alarms? (II) Domain specific refinement

- Adapt the abstraction to the *programming paradigms* typically used in given *domain-specific applications*
- e.g. Astrée for *synchronous control/command*: no recursion, no dynamic memory allocation, maximum execution time, etc.

# A Touch of Abstract Interpretation Theory

# Fixpoint

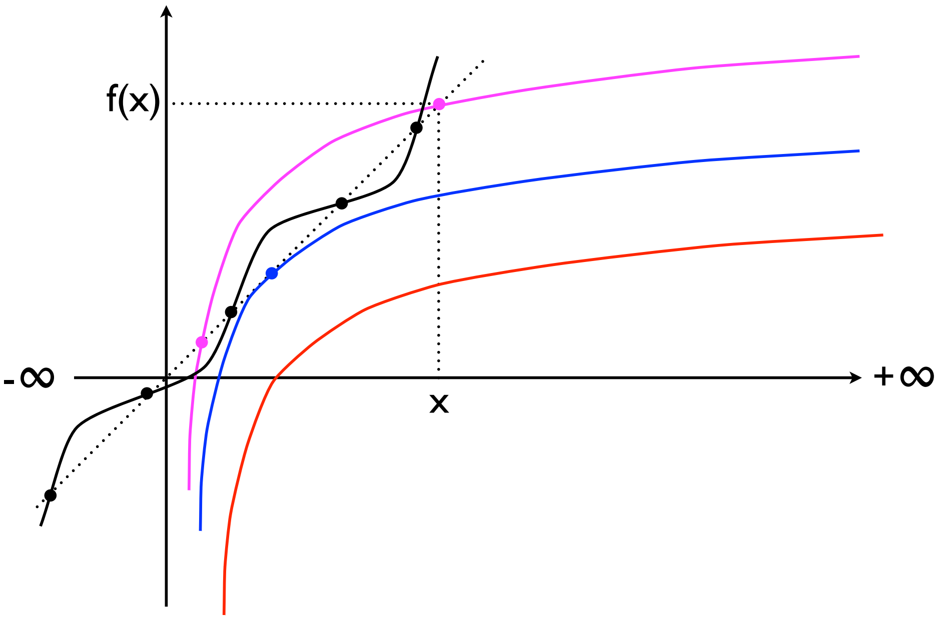
- Set $\mathcal{P}$
- Transformer $F \in \mathcal{P} \to \mathcal{P}$
- Fixpoint

$$x \in \mathcal{P} \text{ is a fixpoint of } F \iff F(x) = x$$

- Poset $\langle \mathcal{P},\ \leqslant \rangle$
- Least fixpoint

$$x \in \mathcal{P} \text{ is the least fixpoint of } F \text{ (written } x = \mathsf{lfp}^{\leqslant} F) \iff F(x) = x \wedge \forall y \in \mathcal{P} : (F(y) = y) \Rightarrow (x \leqslant y)$$

# Fixpoints of increasing functions (Tarski)



# Program properties as fixpoints

- Program semantics and program properties can be formalized as least/greatest fixpoints of increasing transformers on complete lattices [(I)]
  - *Complete lattice / cpo of properties*

$$\langle \mathcal{P},\ \leqslant,\ 0,\ 1,\ \vee,\ \wedge \rangle \quad / \quad \langle \mathcal{P},\ \leqslant,\ 0,\ \vee \rangle$$

  - *Properties* of program P

$$S[\![\mathrm{P}]\!] = \mathsf{lfp}^{\leqslant} F[\![\mathrm{P}]\!]$$

  - *Transformer* of program P

$$F[\![\mathrm{P}]\!] \in \mathcal{P} \to \mathcal{P}, \text{ increasing (or continuous)}$$

[(I)] Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Example: reachable states [(I,II)]

- *Transition system* (set of states $\Sigma$, initial states $\mathcal{I} \subseteq \Sigma$, transition relation $\tau$ )

$$\langle \Sigma,\ \mathcal{I},\ \tau \rangle$$

- *Reflexive transitive closure*

$$\tau^{\star} = \mathsf{lfp}^{\subseteq} \lambda X \bullet \mathbb{1} \cup X \circ \tau$$

- *Right-image* of a set of states by transitions

$$\mathsf{post}[\tau]X \triangleq \{s' \mid \exists s \in X : \tau(s, s')\}$$

$$\langle \wp(\qquad),\ \rangle \underset{\mathsf{post}[\ ]\mathcal{I}}{\overset{\longleftarrow}{\longrightarrow}} \langle \wp(\ ),\ \rangle$$

abstraction

- *Reachable states* from initial states $\mathcal{I}$

$$\mathsf{post}[\tau^{\star}]\mathcal{I} = \mathsf{lfp}^{\subseteq} \lambda X \bullet \mathcal{I} \cup \mathsf{post}[\tau]X$$

[(I)] Patrick 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*, Université Joseph Fourier, Grenoble, France, 21 March 1978

[(II)] Patrick Cousot. Semantic foundations of program analysis. In S.S. Muchnick & N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, Ch. 10, pages 303—342, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

# Proof methods

- *Proof methods* directly follow from the fixpoint definition

$$
\begin{aligned}
& S\llbracket \mathrm{P} \rrbracket \leqslant P \\
\Leftrightarrow\ & \mathsf{lfp}^{\leqslant} F\llbracket \mathrm{P} \rrbracket \leqslant P \\
\Leftrightarrow\ & \exists I : F\llbracket \mathrm{P} \rrbracket(I) \leqslant I \wedge I \leqslant P
\end{aligned}
$$

(proof by Tarski's fixpoint theorem for increasing transformers on complete lattice or Pataria for cpos)

$$\mathsf{lfp}^{\leqslant} F = \bigwedge\{x \mid F(x) \leqslant x\}$$

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Example: Turing/Floyd Invariance Proof

- Bad states

$$\mathcal{B} \subseteq \Sigma$$

- Prove that no bad state is reachable

$$\mathsf{post}[\tau^\star]\mathcal{I} \subseteq \neg\mathcal{B}$$

- Turing/Floyd proof method

$$\exists I \in \wp(\Sigma) : \mathcal{I} \subseteq I \wedge \mathsf{post}[\tau]I \subseteq I \wedge I \subseteq \neg\mathcal{B}$$

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Abstraction

- Abstract the concrete properties into *abstract properties*

$$\langle \mathcal{A},\ \sqsubseteq,\ \bot,\ \top,\ \sqcup,\ \sqcap \rangle$$

- If any concrete property $P \in \mathcal{P}$ has a best abstraction $\alpha(P) \in \mathcal{A}$, then the correspondence is given by a *Galois connection*

$$\langle \mathcal{P},\ \leqslant \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{A},\ \sqsubseteq \rangle$$

i.e.

$$\forall P \in \mathcal{P} : \forall Q \in \mathcal{A} : \alpha(P) \sqsubseteq Q \Leftrightarrow P \leqslant \gamma(Q)$$

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Example: elementwise abstraction

- Morphism

$$h \in \mathcal{P} \mapsto \mathcal{A}$$

- Abstraction

$$\alpha(X) \triangleq \{h(x) \mid x \in X\}$$

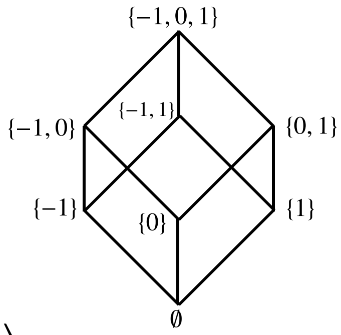
- Galois connection

$$\langle \wp(\mathcal{P}),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \wp(\mathcal{A}),\ \subseteq \rangle$$

- Example: rule of signs

$$h : \mathbb{Z} \to \{-1, 0, 1\}$$

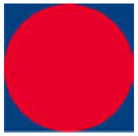
$$h(z) \triangleq z/|z|$$



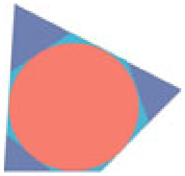
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# In absence of best abstraction

- Best abstraction of a disk by a rectangular parallelogram



- No best abstraction of a disk by a polyhedron (Euclid)



use only concretization or abstraction or widening (1)

(1) Patrick Cousot, Radhia Cousot: Abstract Interpretation Frameworks. J. Log. Comput. 2(4): 511-547 (1992)

# Example abstract transformer: rule of signs

$$\begin{tikzcd}
\{-1,-2,-7\} \arrow[dd, "\alpha"'] & * & \{0,-2,-5\} \arrow[dd, "\alpha"'] & \overset{\Delta}{=} & \{0,2,4,14,5,10,35\} \arrow[d, "\alpha"'] \\
 & & & & = \\
\{-1\} & \overline{*} & \{-1,0\} & \overset{\Delta}{=} & \{1,0\}
\end{tikzcd}$$

Negative Negative or zero Positive or zero

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Example abstract transformer: rule of signs

$$\begin{tikzcd}
\{-3,-4,-7\} \arrow[ddd, "\alpha"'] & + & \{1,2,3\} \arrow[ddd, "\alpha"'] & \overset{\Delta}{=} & \{-2,-3,-6,-1,-2,-5,0,-1,-4\} \arrow[d, "\alpha"'] \\
 & & & & \{-1,0\} \\
 & & & & \subseteq \\
\{-1\} & \overline{+} & \{1\} & \overset{\Delta}{=} & \{-1,0,1\}
\end{tikzcd}$$

Negative Positive Unkown

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Abstract transformer

- An abstract transformer $\overline{F} \in \mathcal{A} \to \mathcal{A}$ is
  - *Sound* iff

$$\forall P \in \mathcal{P} : \alpha \circ F(P) \sqsubseteq \overline{F} \circ \alpha(P)$$

  - *Complete* iff

$$\forall P \in \mathcal{P} : \alpha \circ F(P) = \overline{F} \circ \alpha(P)$$

- Example (rule of sign)
  - Addition: sound, incomplete
  - Multiplication: sound, complete

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Fixpoint abstraction

- For an increasing and sound abstract transformer, we have a *fixpoint approximation*

$$\alpha(\mathsf{lfp}^{\leqslant} F) \sqsubseteq \mathsf{lfp}^{\sqsubseteq} \overline{F}$$

- For an increasing, sound, and complete abstract transformer, we have an *exact fixpoint abstraction*

$$\alpha(\mathsf{lfp}^{\leqslant} F) = \mathsf{lfp}^{\sqsubseteq} \overline{F}$$

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

# Iterative fixpoint computation

- Fixpoint of increasing transformers on cpos can be computed iteratively as limits of (transfinite) iterates

$$F^0 \triangleq \bot$$

$$F^{\beta+1} \triangleq F(F^{\beta}), \quad \beta+1 \text{ successor ordinal}$$

$$F^{\lambda} \triangleq \bigsqcup_{\beta<\lambda} F^{\beta}, \quad \lambda \text{ limit ordinal}$$

Ultimately stationary at rank $\epsilon$

Converges to $F^{\epsilon} = \mathsf{lfp}^{\sqsubseteq} F$

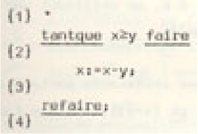
- $\epsilon = \omega$ when $F$ is continuous
- Finite iterates when $F$ operates on a cpo satisfying the ascending chain condition

Patrick Cousot & Radhia Cousot. Constructive versions of Tarski's fixed point theorems. In *Pacific Journal of Mathematics*, Vol. 82, No. 1, 1979, pp. 43—57.

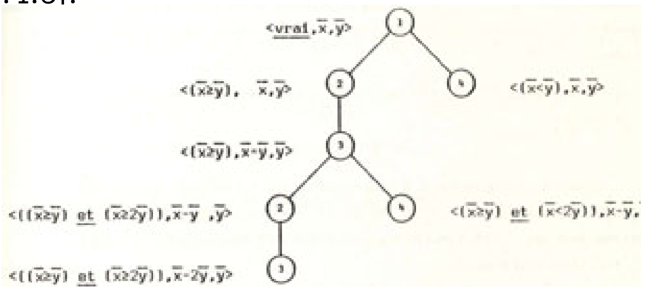
# Example: symbolic execution

- Symbolic execution tree is an abstraction of the prefix of a trace semantics

From [1, Sec. 3.4.5]:



```
{1} *
    tantque x≥y faire
{2}
        x:=x-y;
{3}
    refaire;
{4}
```



$\langle \underline{\text{vrai}}, \bar{x}, \bar{y}\rangle$
1
$\langle(\bar{x}\geq\bar{y}),\ \bar{x},\bar{y}\rangle$
2
4
$\langle(\bar{x}<\bar{y}),\bar{x},\bar{y}\rangle$
$\langle(\bar{x}\geq\bar{y}),\bar{x}-\bar{y},\bar{y}\rangle$
3
$\langle((\bar{x}\geq\bar{y})\ \underline{\text{et}}\ (\bar{x}\geq 2\bar{y})),\bar{x}-\bar{y}\ ,\bar{y}\rangle$
2
4
$\langle(\bar{x}\geq\bar{y})\ \underline{\text{et}}\ (\bar{x}<2\bar{y})),\bar{x}-\bar{y},$
$\langle((\bar{x}\geq\bar{y})\ \underline{\text{et}}\ (\bar{x}\geq 2\bar{y})),\bar{x}-2\bar{y},\bar{y}\rangle$
3

Program Symbolic execution tree

References

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treilli analyse sémantique de programmes.* Th se d tat s sciences math matiques, Universit scienti que et m dicale de Grenobl Grenoble, 21 mars 1978.

[2] J.C. King. Symbolic Execution and Program Testing, CACM 19:7(385 394), 1976.

# Example: symbolic execution (cont'd)

An abstract interpretation

The abstract properties of $\mathcal{A}$ have the form:

$$\prod_{c\in\text{Control}} \{\langle Q_i,\, E_i\rangle \mid i \in \Delta_c\}$$

(where $Q_i$ is a *path condition* and $E_i$ is a *valuation* in terms of initial values $\bar{x}$) with concretization

$$\{\langle c,\, x\rangle \mid \exists \bar{x} : \bigvee_{i\in\Delta_c} Q_i(\bar{x}) \wedge x = E_i(\bar{x})\}$$

# Example: symbolic execution (cont'd)

– Test transformer:

$$\begin{aligned}\texttt{test}[\![b]\!](\{\langle Q_i,\ E_i\rangle \mid i \in \Delta_c\}) = \\ \{\langle Q_i \wedge b[x \backslash E_i(\bar{x})],\ E_i\rangle \mid i \in \Delta_c\}\end{aligned}$$

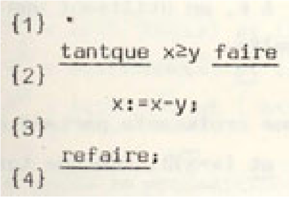
– Assignment transformer:

$$\begin{aligned}\texttt{assign}[\![x := e(x)]\!](\{\langle Q_i,\ E_i\rangle \mid i \in \Delta_c\}) = \\ \{\langle Q_i,\ e[x \backslash E_i(\bar{x})]\rangle \mid i \in \Delta_c\}\end{aligned}$$

# Example: symbolic execution (cont'd)

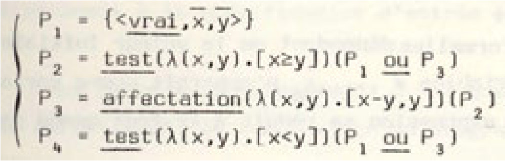
Example:

– Program:



```
{1}  •
     tantque x≥y faire
{2}
          x:=x-y;
{3}
     refaire;
{4}
```

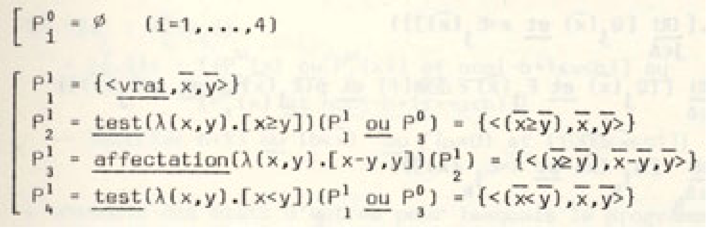
– Program transformer $\mathcal{F}$:



$$\begin{cases} P_1 = \{<\underline{\text{vrai}},\bar{x},\bar{y}>\} \\ P_2 = \underline{\text{test}}(\lambda(x,y).[x \geq y])(P_1\ \underline{\text{ou}}\ P_3) \\ P_3 = \underline{\text{affectation}}(\lambda(x,y).[x-y,y])(P_2) \\ P_4 = \underline{\text{test}}(\lambda(x,y).[x<y])(P_1\ \underline{\text{ou}}\ P_3) \end{cases}$$

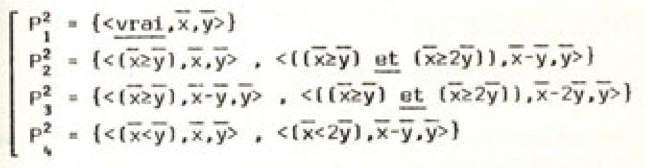
# Example: symbolic execution (cont'd)

– Fixpoint iteration: (chaotic iterations)



$$\left[\ P^0_1 = \emptyset \quad (1=1,\ldots,4)\right.$$

$$\left[\begin{array}{l} P^1_1 = \{<\underline{\text{vrai}},\bar{x},\bar{y}>\} \\ P^1_2 = \underline{\text{test}}(\lambda(x,y).[x \geq y])(P^1_1\ \underline{\text{ou}}\ P^0_3) = \{<(\bar{x} \geq \bar{y}),\bar{x},\bar{y}>\} \\ P^1_3 = \underline{\text{affectation}}(\lambda(x,y).[x-y,y])(P^1_2) = \{<(\bar{x} \geq \bar{y}),\bar{x}-\bar{y},\bar{y}>\} \\ P^1_4 = \underline{\text{test}}(\lambda(x,y).[x<y])(P^1_1\ \underline{\text{ou}}\ P^0_3) = \{<(\bar{x}<\bar{y}),\bar{x},\bar{y}>\} \end{array}\right.$$



$$\left[\begin{array}{l} P^2_1 = \{<\underline{\text{vrai}},\bar{x},\bar{y}>\} \\ P^2_2 = \{<(\bar{x} \geq \bar{y}),\bar{x},\bar{y}>\ ,\ <((\bar{x} \geq \bar{y})\ \underline{\text{et}}\ (\bar{x} \geq 2\bar{y})),\bar{x}-\bar{y},\bar{y}>\} \\ P^2_3 = \{<(\bar{x} \geq \bar{y}),\bar{x}-\bar{y},\bar{y}>\ ,\ <((\bar{x} \geq \bar{y})\ \underline{\text{et}}\ (\bar{x} \geq 2\bar{y})),\bar{x}-2\bar{y},\bar{y}>\} \\ P^2_4 = \{<(\bar{x}<\bar{y}),\bar{x},\bar{y}>\ ,\ <(\bar{x}<2\bar{y}),\bar{x}-\bar{y},\bar{y}>\} \end{array}\right.$$

. . .

# Example: symbolic execution (cont'd)

- Chaotic fixpoint iteration explores all finite/infinite execution paths symbolically.
- These chaotic iterates have a termination problem

# Example: symbolic execution (cont'd)

- Solutions to the iteration termination problem:
  - Bounded symbolic execution
  - or, ask the end-user for a loop invariant
  - or, pass to the limit:
    - Generalization: express iterates in terms of the iterate's rank (using a relational abstraction such as linear (in-)equalities)
    - Infinite disjunctions (~ existential quantifier elimination)
- or, more generally, accelerate convergence

# Widening

- Definition (widening $\nabla \in \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$)
  - $\langle \mathcal{A}, \sqsubseteq \rangle$ poset
  - Over-approximation

$$\forall x, y \in \mathcal{A} : x \sqsubseteq x \nabla y \wedge y \sqsubseteq x \nabla y$$

  - Termination

Given any sequence $\langle x^n,\ n \in \mathbb{N} \rangle$, the widened sequence $\langle y^n,\ n \in \mathbb{N} \rangle$

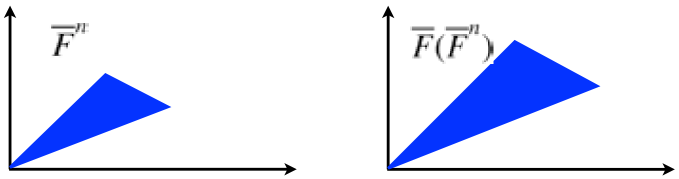
$$y^0 \triangleq x^0, \ldots, y^{n+1} \triangleq y^n \nabla x^n, \ldots$$

converges to a limit $y^\ell$ (such that $\forall m \geqslant \ell : y^m = y^\ell$)

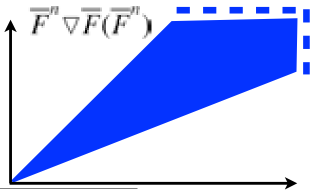
Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

# Example: (simple) widening for polyhedra

- Iterates



- Widening



Patrick 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*, Université Joseph Fourier, Grenoble, France, 21 March 1978.

Patrick Cousot, Nicolas Halbwachs: Automatic Discovery of Linear Restraints Among Variables of a Program. POPL 1978: 84-96

# Iteration with widening

- *Iterates with widening* for transformer $\overline{F} \in \mathcal{A} \rightarrow \mathcal{A}$

$$\begin{array}{lcll} \overline{F}^0 & \triangleq & \bot & \\ \overline{F}^{n+1} & \triangleq & \overline{F}^n & \text{when } \overline{F}(\overline{F}^n) \sqsubseteq \overline{F}^n \\ \overline{F}^{n+1} & \triangleq & \overline{F}^n \nabla \overline{F}(\overline{F}^n) & \text{otherwise} \end{array}$$

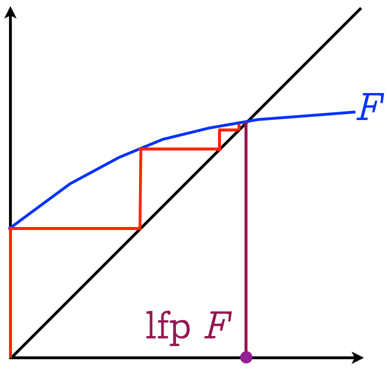
- The *widening speeds up convergence* (at the cost of imprecision)

**Theorem** (*Limit of iterates with widening*) The iterates of $\overline{F}$ with widening $\nabla$ from $\bot$ on a poset $\langle \mathcal{A}, \sqsubseteq, \bot \rangle$ converge to a limit $\overline{F}^\ell$ such that $\overline{F}(\overline{F}^\ell) \sqsubseteq \overline{F}^\ell$ (and so $\mathsf{lfp}^{\sqsubseteq} \overline{F} \sqsubseteq \overline{F}^\ell$ when $\overline{F}$ is increasing).

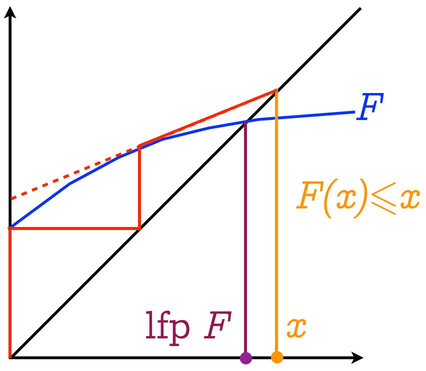
- Can be improved by a *narrowing*.

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

## Intuition for iteration with widening



Infinite iteration



Accelerated iteration with widening
(e.g. with a widening based on the derivative as in Newton-Raphson method)

## Reduced product

- The reduced product combines abstractions by performing their conjunction in the abstract

$$\langle \mathcal{P},\ \leqslant\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle \mathcal{A}_1,\ \sqsubseteq_1\rangle$$

$$\langle \mathcal{P},\ \leqslant\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle \mathcal{A}_2,\ \sqsubseteq_2\rangle$$

$$\mathcal{A}_1 \otimes \mathcal{A}_2 \triangleq \{\langle \alpha_1(\gamma_1(P_1) \wedge \gamma_2(P_2)),\ \alpha_2(\gamma_1(P_1) \wedge \gamma_2(P_2))\rangle \mid P_1 \in \mathcal{A}_1 \wedge P_2 \in \mathcal{A}_2\}$$

$$\langle \mathcal{P},\ \leqslant\rangle \underset{\alpha_1 \times \alpha_2}{\overset{\gamma_1 \times \gamma_2}{\leftrightarrows}} \langle \mathcal{A}_1 \otimes \mathcal{A}_2,\ \sqsubseteq_1 \times \sqsubseteq_2\rangle$$

- Example: (positive or zero) ⊗ odd = <positive,odd>

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282
Patrick Cousot, Radhia Cousot, Laurent Mauborgne: The Reduced Product of Abstract Domains and the Combination of Decision Procedures. FOSSACS 2011: 456-472

## The abstract interpretation methodology for static analysis

- Define the semantics, and strongest properties of any program in fixpoint form
- Define your abstraction (by composition and combination of elementary abstractions)
- Lift your abstraction to abstract transformers
- Lift your abstraction to abstract fixpoints (using widening/narrowing when becessary)
- Iterate by refinements guided by experiments (or automate them in simple cases)
- Correct by construction

## Recent advances

- The same principles apply to *termination*

Patrick Cousot, Radhia Cousot: *An abstract interpretation framework for termination*. POPL 2012: 245-258

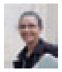
- and to *probabilistic programs*

Patrick Cousot and Michaël Monerau. Probabilistic Abstract Interpretation. In H. Seidel (Ed), *22nd European Symposium on Programming (ESOP 2012)*, Tallinn, Estonia, 24 March—1 April 2012. Lecture Notes in Computer Science, vol. 7211, pp. 166—190, © Springer, 2012.

# ASTRÉE

Bruno Blanchet[68]

Patrick Cousot



Radhia Cousot

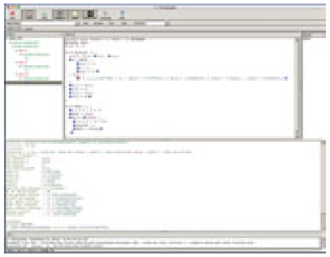
J r me Feret

Laurent Mauborgne

Antoine Min

David Monniaux[69]

Xavier Rival



[68] Nov. 2001   Nov. 2003.
[69] Nov. 2001   Aug. 2007.
[70] Nov. 2001 – Aug. 2010.

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, Xavier Rival: Why does Astrée scale up? Formal Methods in System Design 35(3): 229-264 (2009)

Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné, Laurent Mauborgne, David Monniaux, Xavier Rival: Varieties of Static Analyzers: A Comparison with ASTREE. TASE 2007: 3-20

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: Combination of Abstractions in the ASTRÉE Static Analyzer. ASIAN 2006: 272-300

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: The ASTREÉ Analyzer. ESOP 2005: 21-30

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: A static analyzer for large safety-critical software. PLDI 2003: 196-207

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software. The Essence of Computation 2002: 85-108

# Target language and applications

- C programming language
  - Without recursion, `longjump`, dynamic memory allocation, conflicting side effects, backward jumps, system calls (stubs)
  - With all its horrors (`union`, pointer arithmetics, etc)
  - Reasonably extending the standard (e.g. size & endianess of integers, IEEE 754-1985 floats, etc)
- Originally for synchronous control/command
  - e.g. generated from Scade

# The semantics of C implementations is very hard to define

What is the effect of out-of-bounds array indexing?

```
% cat unpredictable.c
#include <stdio.h>
int main () { int n, T[1];
 n = 2147483647;
 printf("n = %i, T[n] = %i\n", n, T[n]);
}
```

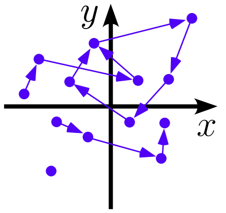
Yields different results on different machines:

```
        n = 2147483647, T[n] = 2147483647   Macintosh PPC
        n = 2147483647, T[n] = -1208492044  Macintosh Intel
        n = 2147483647, T[n] = -135294988   PC Intel 32 bits
        Bus error                           PC Intel 64 bits
```

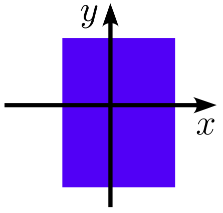
# Implicit specification

- Absence of runtime errors: overflows, division by zero, buffer overflow, null & dangling pointers, alignment errors, …
- Semantics of runtime errors:
  - Terminating execution: stop (e.g. floating-point exceptions when traps are activated)
  - Predictable outcome: go on with worst case (e.g. signed integer overflows result in some integer, some options: e.g. modulo arithmetics)
  - Unpredictable outcome: stop (e.g. memory corruption)

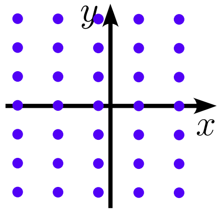
# Abstractions



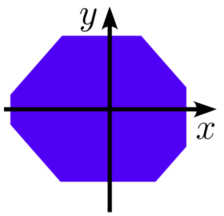
Collecting semantics:
partial traces



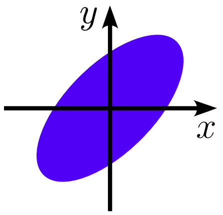
Intervals:
$x \in [a, b]$



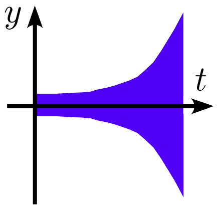
Simple congruences:
$x \equiv a[b]$



Octagons:
$\pm x \pm y \leqslant a$



Ellipses:
$x^2 + by^2 - axy \leqslant d$



Exponentials:
$-a^{bt} \leqslant y(t) \leqslant a^{bt}$

# Example of general purpose abstraction: octagons

- Invariants of the form $\pm x \pm y \leq c$, with $\mathcal{O}(\mathbf{N}^2)$ memory and $\mathcal{O}(\mathbf{N}^3)$ time cost.
- Example:

```
while (1) {
  R = A-Z;
  L = A;
  if (R>V)
    { ★ L = Z+V; }
  ★
}
```

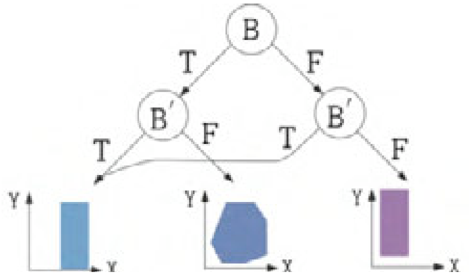
- At ★, the interval domain gives L ≤ max(max A, (max Z)+(max V)).
- In fact, we have L ≤ A.
- To discover this, we must know at ★ that R = A-Z and R > V.

- Here, R = A-Z cannot be discovered, but we get L-Z ≤ max R which is sufficient.
- We use many octagons on **small packs** of variables instead of a large one using all variables to cut costs.

Antoine Miné: *The octagon abstract domain*. Higher-Order and Symbolic Computation 19(1): 31-100 (2006)

# Example of general purpose abstraction: decision trees

```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leaves

# Example of domain-specific abstraction: ellipses

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
             + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```

Jérôme Feret: *Static Analysis of Digital Filters*. ESOP 2004: 33-48

## Example of domain-specific abstraction: exponentials

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
% cat count.config
__ASTREE_volatile_input((I [0,1]));
__ASTREE_max_clock((3600000));
% astree -exec-fn main -config-sem count.config count.c|grep '|R|'
|R| <= 0. + clock *1. <= 3600001.
```

← potential overflow!

## Example of domain-specific abstraction: exponentials

```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{ X=E;
  if (FIRST) { P = X; }
  else
    { P =  (P - ((((2.0 * P) - A) - B)
            * 4.491048e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1 +
1.19209290217e-07)^clock - 5.87747175411e-39
/ 1.19209290217e-07 <= 23.0393526881
```

Jérôme Feret: *The Arithmetic-Geometric Progression Abstract Domain*. VMCAI 2005: 42-58

## An erroneous common belief on static analyzers

"The properties that can be proved by static analyzers are often simple" [2]

Like in mathematics:

- May be simple to state (no overflow)
- But harder to discover (S[0], S[1] in [-1327.02698354, 1327.02698354]
- And difficult to prove (since it requires finding a non trivial non-linear invariant for second order filters with complex roots [Fer04], which can hardly be found by exhaustive enumeration)

Reference

[2] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, No. 7, July 2008.

[Fer04] Jérôme Feret: Static Analysis of Digital Filters. ESOP 2004: 33-48

# Industrial applications

Daniel Kästner, Christian Ferdinand, Stephan Wilhelm, Stefana Nevona, Olha Honcharova, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, Xavier Rival, and Élodie-Jane Sims. Astrée: Nachweis der Abwesenheit von Laufzeitfehlern. In *Workshop ``Entwicklung zuverlässiger Software-Systeme''*, Regensburg, Germany, June 18th, 2009.

Olivier Bouissou, Éric Conquet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Khalil Ghorbal, Éric Goubault, David Lesens, Laurent Mauborgne, Antoine Miné, Sylvie Putot, Xavier Rival, & Michel Turin. Space Software Validation using Abstract Interpretation. In *Proc. of the Int. Space System Engineering Conf., Data Systems in Aerospace (DASIA 2009)*. Istambul, Turkey, May 2009, 7 pages. ESA.

Jean Souyris, David Delmas: Experimental Assessment of Astrée on Safety-Critical Avionics Software. SAFECOMP 2007: 479-490

David Delmas, Jean Souyris: Astrée: From Research to Industry. SAS 2007: 437-451

Jean Souyris: Industrial experience of abstract interpretation-based static analyzers. IFIP Congress Topical Sessions 2004: 393-400

Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, Christian Ferdinand: An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. DSN 2003: 625-632

# Examples of applications

- Verification of the absence of runtime-errors in
  - Fly-by-wire flight control systems [(*)]

  - ATV docking system [(*)]

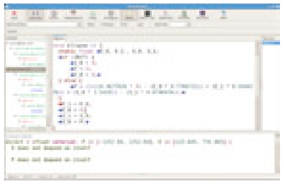
  - Flight warning system (on-going work)

(*) No false alarm a all!

# Industrialization

- 8 years of research (CNRS/ENS/INRIA):

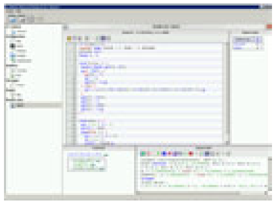
www.astree.ens.fr

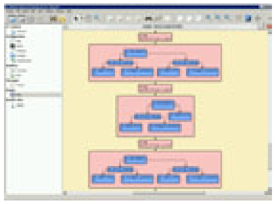


- Industrialization by AbsInt (since Jan. 2010):

www.absint.com/astree/

AbsInt Angewandte Informatik





# On-going work

# ASTRÉEA: Verification of embedded real-time parallel C programs

Antoine Miné: Static *Analysis of Run-Time Errors in Embedded Critical Parallel C Programs*. ESOP 2011: 398-418

# Parallel programs

- Bounded number of processes with shared memory, events, semaphores, message queues, blackboards,…
- Processes created at initialization only
- Real time operating system (ARINC 653) with fixed priorities (highest priority runs first)
- Scheduled on a single processor

# Verified properties

- Absence of runtime errors
- Absence of unprotected data races

# Semantics

- No memory consistency model for C
- Optimizing compilers consider sequential processes out of their execution context

```
               init:  flag1 = flag2 = 0
           process 1:           |           process 2:
flag1 = 1;                      | flag2 = 1;
if (!flag2)                     | if (!flag1)
{                               | {
  /* critical section */        |   /* critical section */
```

write to `flag1/2` and read of `flag2/1` are independent so can be reordered → error!

- We assume:
  - sequential consistency in absence of data race
  - for data races, values are limited by possible interleavings between synchronization points

# Abstractions

- Based on Astrée for the sequential processes
- Takes scheduling into account
- OS entry points (semaphores, logbooks, sampling and queuing ports, buffers, blackboards, …) are all stubbed (using Astrée stubbing directives)
- Interference between processes: flow-insensitive abstraction of the writes to shared memory and inter-process communications

Note: interference abstraction is currently being made more precise

# Example of application: FWS

- Degraded mode (5 processes, 100 000 LOCS)
  - 1h40 on 64-bit 2.66 GHz Intel server
  - A few dozens of alarms (64)
- Full mode (15 processes, 1 600 000 LOCS)
  - 24 h
  - a few hundreds of alarms !!! work going on !!! (e.g. analysis of complex data structures, logs, etc)

# Abstract interpretation based static analyzers

# Software

- Ait: static analysis of the worst-case execution time of control/command software (www.absint.com/ait/)
- Astrée: proof of absence of runtime errors in embedded synchronous real time control/command software (www.absint.com/astree/), AstréeA for asynchronous programs (www.astreea.ens.fr/)
- C Global Surveyor, NASA, static analyzer for flight software of NASA missions (www.cmu.edu/silicon-valley/faculty-staff/venet-arnaud.html)
- IKOS (Inference Kernel for Open Static Analyzers), (www.cmu.edu/silicon-valley/software-systems-management/software-verification.html)
- Checkmate: static analyzer of multi-threaded Java programs (www.pietro.ferrara.name/checkmate/)
- CodeContracts Static Checker, Microsoft (msdn.microsoft.com/en-us/devlabs/dd491992.aspx)
- Fluctuat: static analysis of the precision of numerical computations (www-list.cea.fr/labos/gb/LSL/fluctuat/index.html)

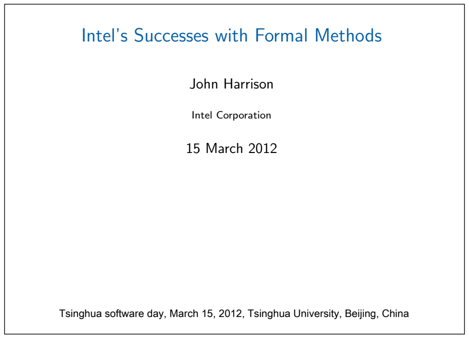
# Software

- Infer: Static analyzer for C/C$^{++}$ (monoidics.com/)
- Julia: static analyzer for Java and Android programs (www.juliasoft.com/juliasoft-android-java-verification.aspx?Id=201177234649)
- Predator: static analyzer of C dynamic data structures using separation logic (www.fit.vutbr.cz/research/groups/verifit/tools/predator/)
- Terminator: termination proof (www.cs.ucl.ac.uk/staff/p.ohearn/Invader/Invader/Invader_Home.html)
- etc.

- Apron numerical domains library (apron.cri.ensmp.fr/library/)
- Parma Polyhedral Library (bugseng.com/products/ppl/)
- etc.

# Hardware

- (Generalized) symbolic trajectory evaluation (Intel)



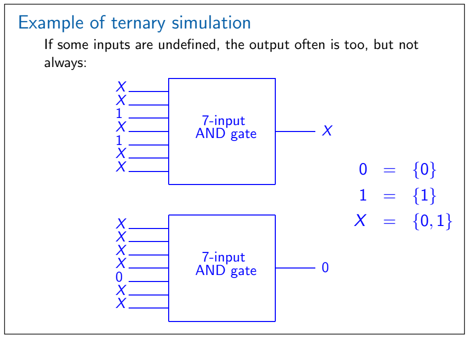
Intel's Successes with Formal Methods

John Harrison

Intel Corporation

15 March 2012

Tsinghua software day, March 15, 2012, Tsinghua University, Beijing, China



Example of ternary simulation

If some inputs are undefined, the output often is too, but not always:

$$0 = \{0\}$$
$$1 = \{1\}$$
$$X = \{0,1\}$$

Jin Yang and Carl-Johan H. Seger, *Generalized Symbolic Trajectory Evaluation — Abstraction in Action*, Formal Methods in Computer-Aided Design, Lecture Notes in Computer Science, 2002, Volume 2517/2002, 70–87.

Jin Yang; Seger, C.-J.H.; *Introduction to generalized symbolic trajectory evaluation*, IEEE Transactions onVery Large Scale Integration (VLSI) Systems 11(3), June 2003, 345–353.

## Steps towards larger adoption in industry

- RTCA/DO-333 Formal Methods Supplement to DO-178C and DO-278A
- Mandatory use of Astrée in the software production chain of a European civil airplane manusfacturer on all planes in production and design
- ...

# Conclusion

## On research

If you reason/compute on computer/biological/... systems behaviors, you probably do abstract interpretation

## On applications

If the simulation/analysis/checking of your computer/biological/… systems model does not scale up, consider using (sound (and complete)) abstract interpretations

# The End, Thank You