On Various Abstract Understandings of Abstract Interpretation

Patrick Cousot

cims.nyu.edu/~pcousot

TASE 2015
The 9th International Symposium on Theoretical Aspects of Software Engineering

September 12—14, 2015 — Nanjing, China

Motivation

Formal methods

Reasoning on programs are

- Reasonings on properties of their semantics (*i.e.* execution behaviors)
- Always involve some form of abstraction

Abstract interpretation

A theory establishing a correspondance between

- Concrete semantic properties

  ↑ what you want to prove on the semantics

- Abstract properties

  ↑ how to prove it in the abstract

Objective: formalize

- formal methods
- algorithms for reasoning on programs

Fundamental motivations

Scientific research

in Mathematics/Physics:

trend towards unification and synthesis through universal principles

in Computer science:

trend towards dispersion and parcelization through a collection of local techniques for specific applications

An exponential process, will stop!

Example: reasoning on computational structures

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

Example: reasoning on computational structures

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

Example: reasoning on computational structures

Abstract interpretation

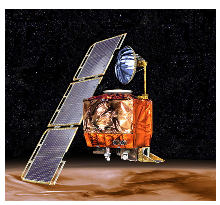
WCET
Axiomatic semantics
Security protocole verification
Systems biology analysis
Operational semantics
Confidentiality analysis
Dataflow analysis
Model checking
Database query
Abstraction refinement
Partial evaluation
Type inference
Program synthesis
Obfuscation
Dependence analysis
Separation logic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

Practical motivations

All computer scientists have experienced bugs

Ariane 5.01 failure (overflow)

Patriot failure (float rounding)



Mars orbiter loss (unit error)

Heartbleed (buffer overrun)

Checking the presence of bugs by debugging is great

Proving their absence by static analysis is even better!

Undecidability and complexity is the challenge for automation

ars technica

MAIN MENU MY STORIES: 25 FORUMS SUBSCRIBE JOBS

TECHNOLOGY LAB / INFORMATION TECHNOLOGY

Boeing 787 Dreamliners contain a potentially catastrophic software bug

Beware of integer overflow-like bug in aircraft's electrical system, FAA warns.

by Dan Goodin - May 1, 2015 7:55pm CEST

Share Tweet 152

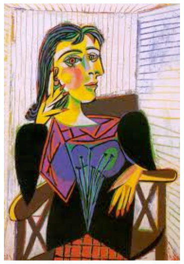
A software vulnerability in Boeing's new 787 Dreamliner jet has the potential to cause pilots to lose control of the aircraft, possibly in mid-flight, Federal Aviation Administration officials warned airlines recently.

The bug—which is either a classic integer overflow or one very much resembling it—resides in one of the electrical systems responsible for generating power, according to memo the FAA issued last week. The vulnerability, which Boeing reported to the FAA, is triggered when a generator has been running continuously for a little more than eight months. As a result, FAA officials have adopted a new airworthiness directive (AD) that airlines will be required to follow, at least until the underlying flaw is fixed.

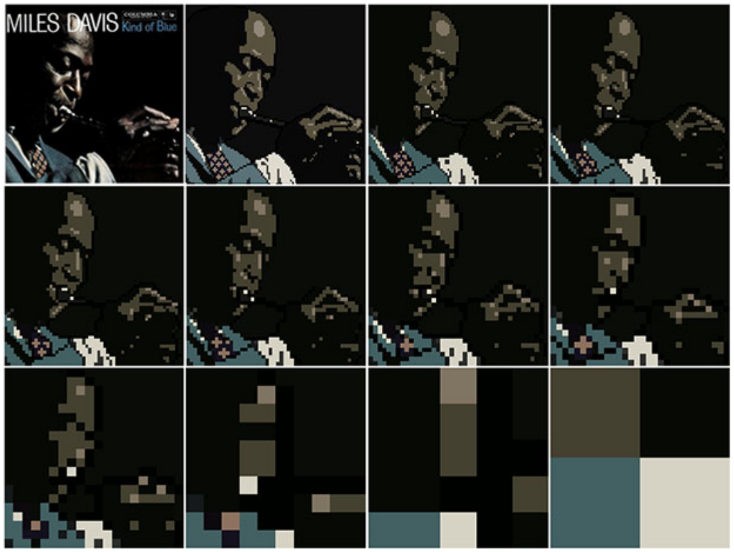
"This AD was prompted by the determination that a Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode," the memo stated. "This condition is caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane."

Informal examples of abstraction

Abstractions of Dora Maar by Picasso



Pixelation



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

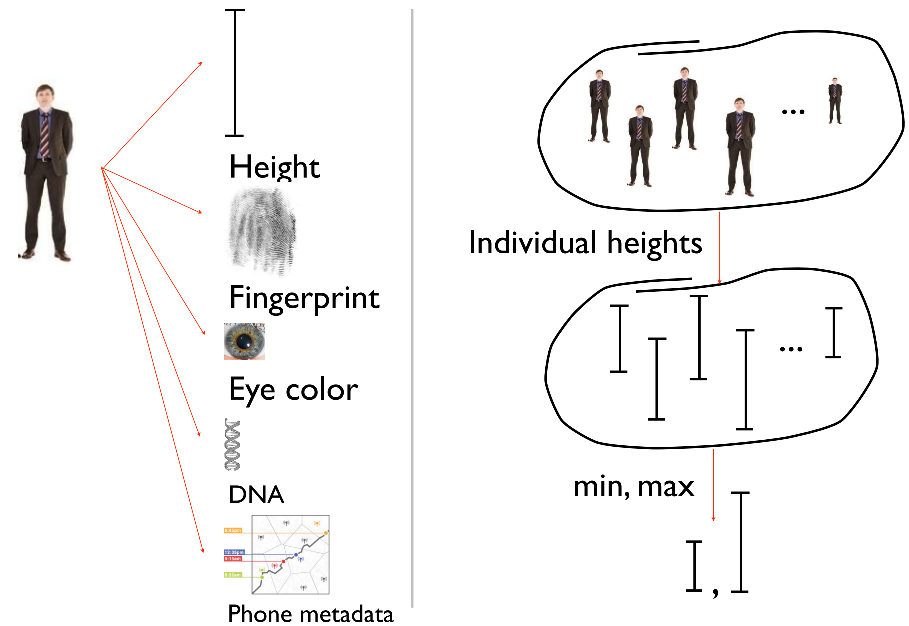
Image credit: *Photograph by Jay Maisel*

An old idea...

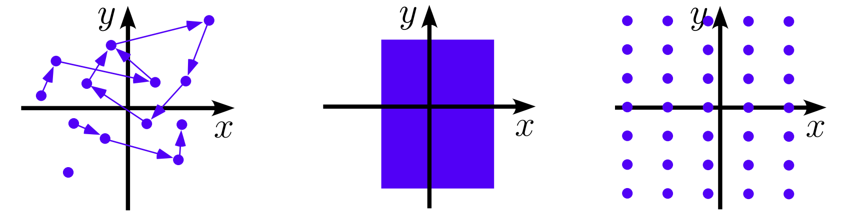
20 000 years old picture in a spanish cave:

(the concrete is unknown)

Abstractions of a man / crowd



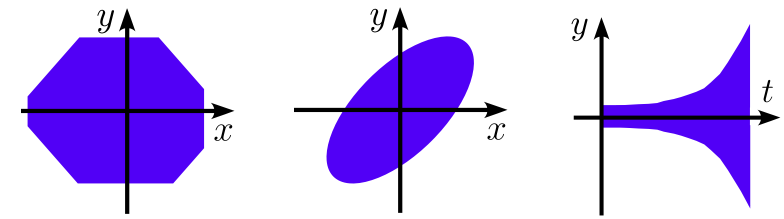
Numerical abstractions in Astrée



Collecting semantics:[1, 5] partial traces

Intervals:[20] $x \in [a, b]$

Simple congruences:[24] $x \equiv a[b]$



Octagons:[25] $\pm x \pm y \leqslant a$

Ellipses:[26] $x^2 + by^2 - axy \leqslant d$

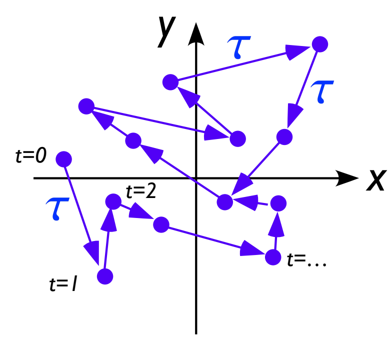
Exponentials:[27] $-a^{bt} \leqslant y(t) \leqslant a^{bt}$

An informal introduction to abstract interpretation

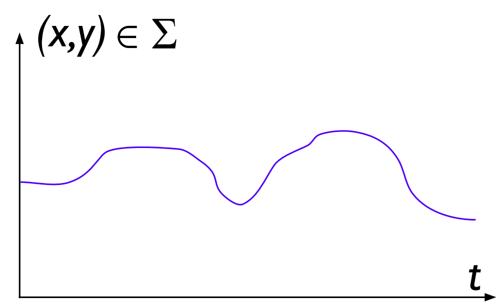
P. Cousot & R. Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In *Logics and Languages for Reliability and Security*, J. Esparza, O. Grumberg, & M. Broy (Eds), NATO Science Series III: Computer and Systems Sciences, © IOS Press, 2010, Pages 1—29.

1) Define the programming language

*Formalize the concrete **execution** of programs (e.g. transition system)*



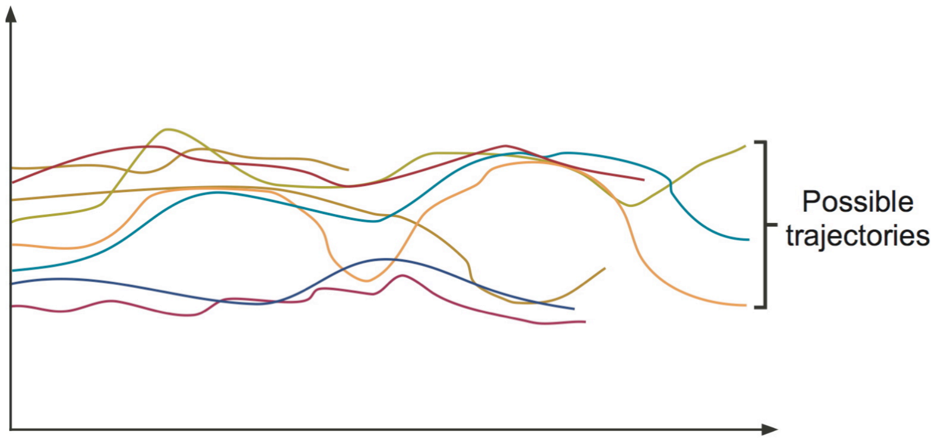
Trajectory in state space Σ



Space/time trajectory

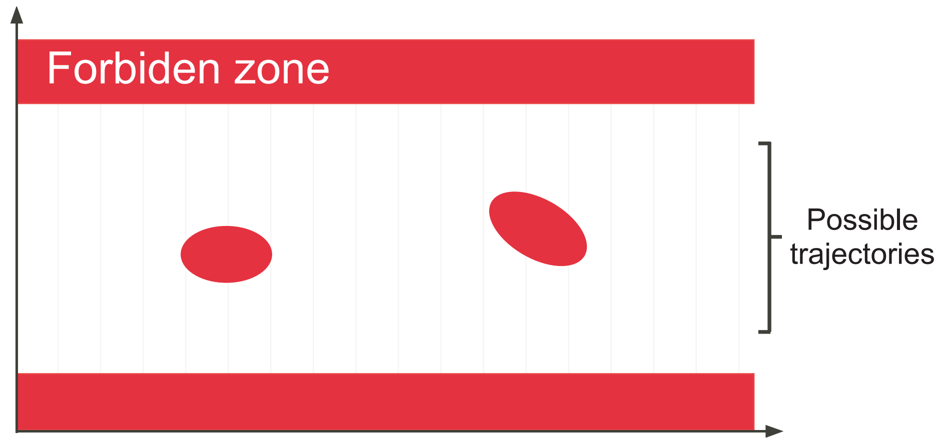
II) Define the program properties of interest

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



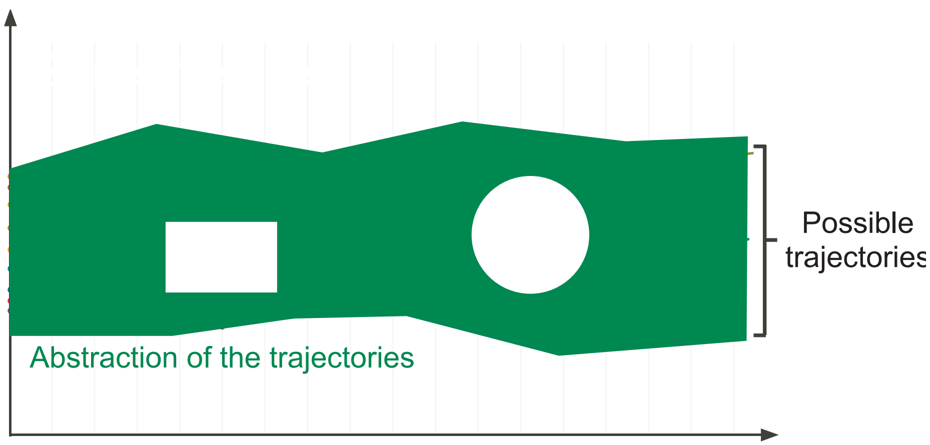
III) Define which specification must be checked

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



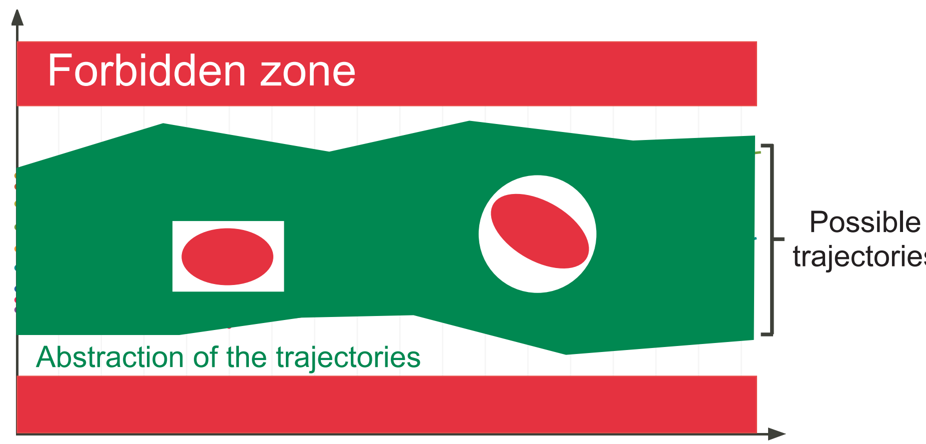
IV) Choose the appropriate abstraction

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



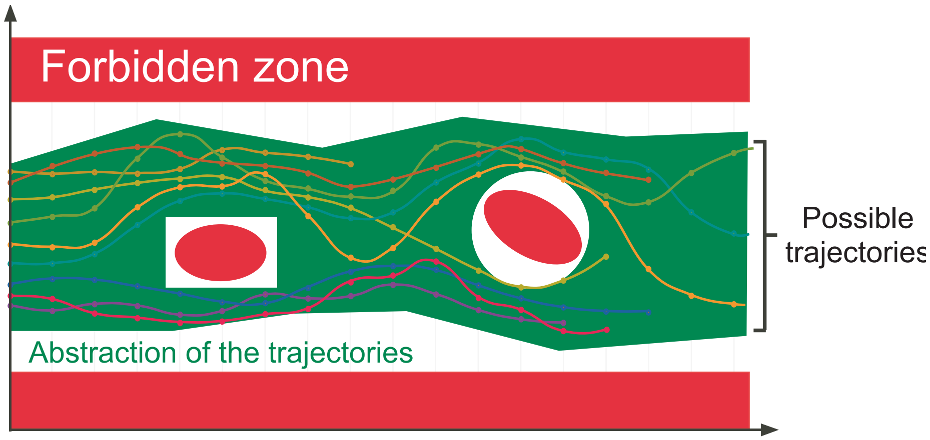
V) Mechanically verify in the abstract

*The proof is fully **automatic***



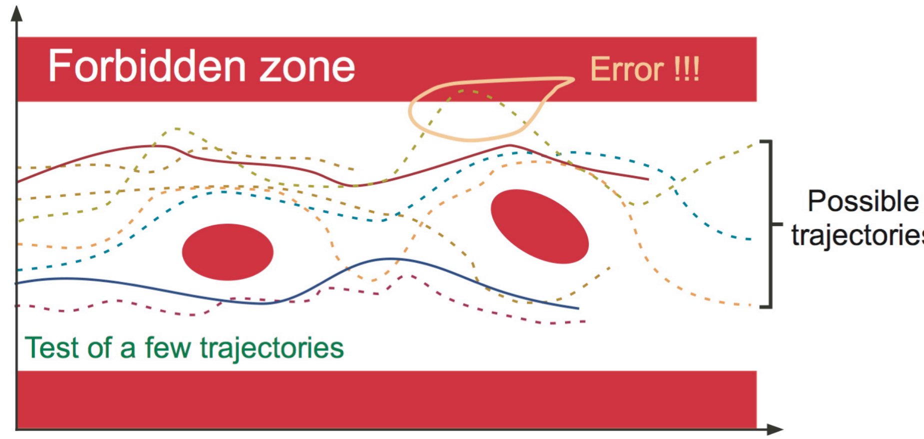
Soundness of the abstract verification

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



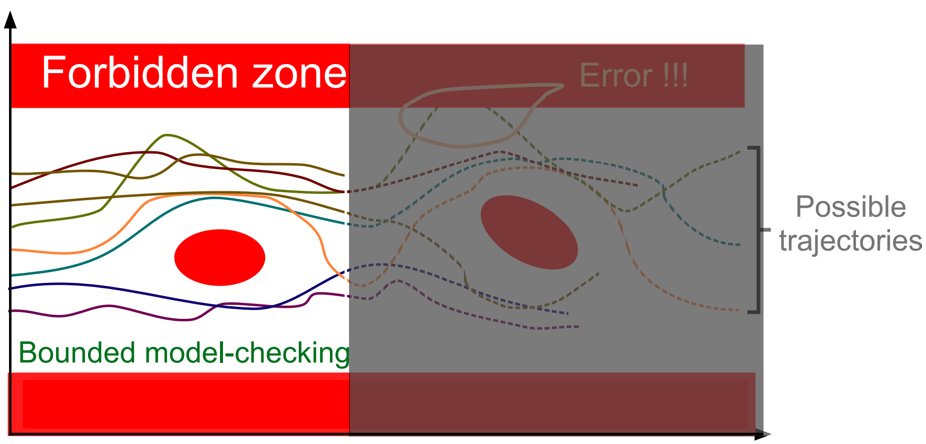
Unsound validation: testing

Try a few cases



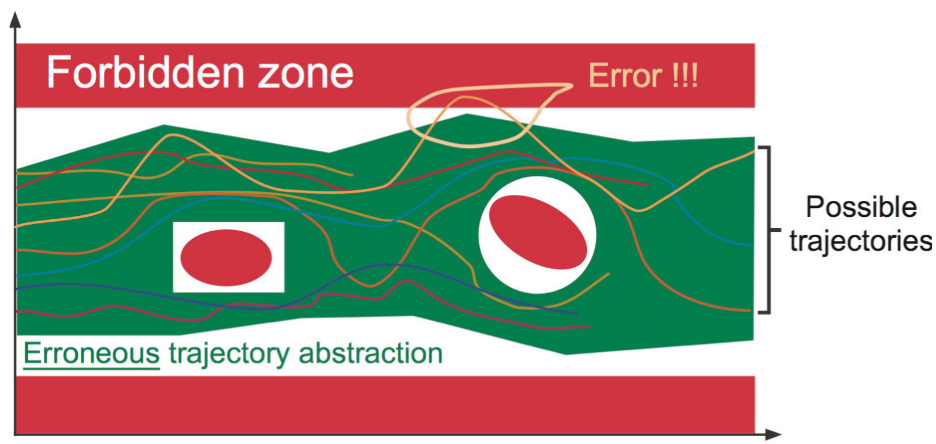
Unsound validation: bounded model-checking

Simulate the beginning of all executions



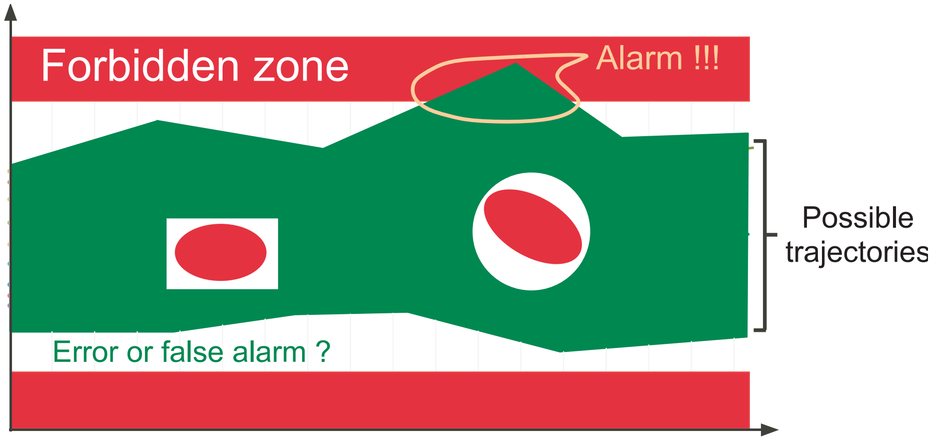
Unsound validation: static analysis

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



Incompleteness

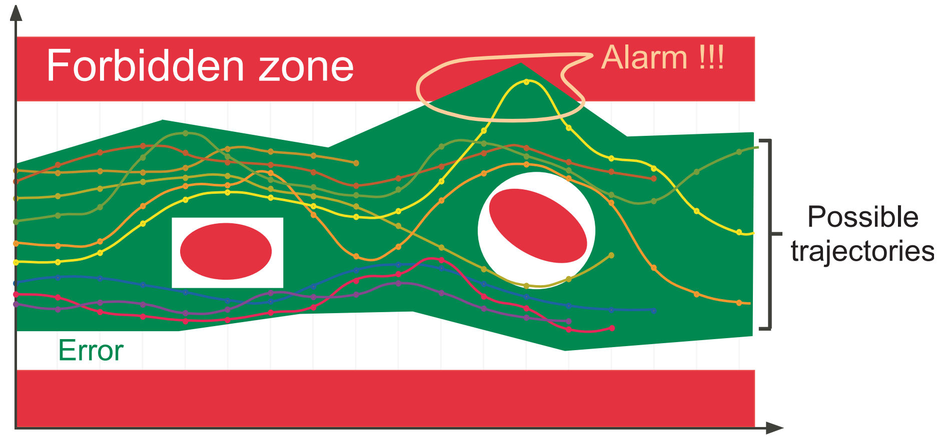
When abstract proofs may fail while concrete proofs would succeed



By soundness an alarm must be raised for this overapproximation!

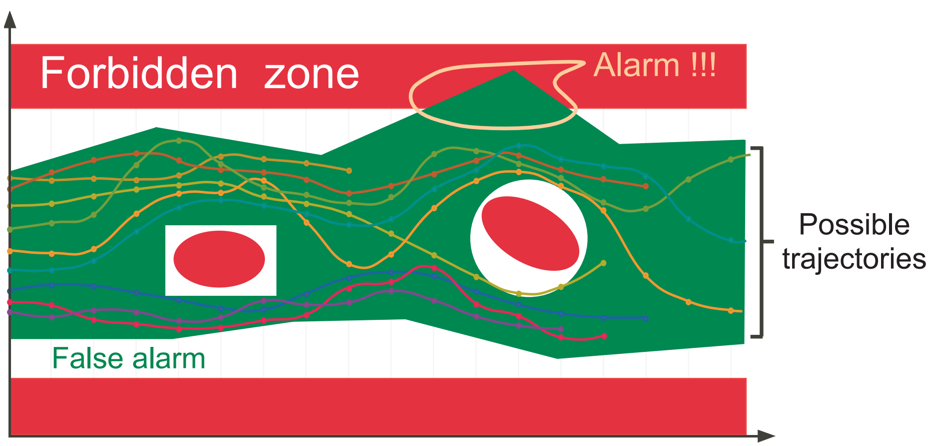
True error

The abstract alarm may correspond to a concrete error



False alarm

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



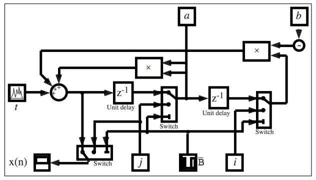
What to do about false alarms?

- Consider special cases: finite (small) models (model-checking), decidable cases (SMT solvers), human interaction (theorem provers, proof verifiers), …
- Automatic refinement: inefficient and may not terminate (Gödel, see next slide)
- Domain-specific abstraction:
  - Adapt the abstraction to the *programming paradigms* typically used in given *domain-specific applications*
  - e.g. *synchronous control/command*: no recursion, simple memory allocation, maximum execution time, etc.

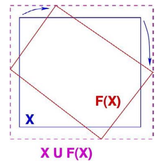
In general refinement does not terminate

- Example:filter invariant abstraction:

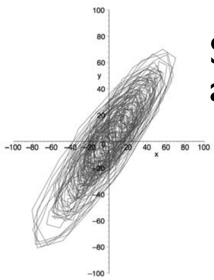
2nd order filter:



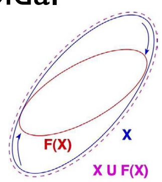
Unstable polyhedral abstraction:



Counter-example guided refinement will indefinitely add missing points according to the execution trace:



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.

Abstract Interpretation

Abstract interpretation is all about:

Soundness

Induction

A very short more formal introduction to 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. Static Determination of Dynamic Properties of Programs. In B. Robinet, editor, *Proceedings of the second international symposium on Programming*, Paris, France, pages 106—130, April 13-15 1976, Dunod, Paris.

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

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.

Properties and their Abstractions

Concrete properties

A concrete property is represented by the set of elements which have that property:

- universe (set of elements) $\mathscr{D}$ (e.g. a semantic domain)
- properties of these elements: $P \in \wp(\mathscr{D})$
- "x has property P" is $x \in P$

$\langle \wp(\mathscr{D}), \subseteq, \cup, \cap, \ldots \rangle$ is a *complete lattice* for inclusion $\subseteq$ (*i.e. logical implication*)

Example of Property

- Odd natural numbers
  $O = \{ 1, 3, 5, 7, \ldots \}$
- x is odd
  $x \in O$
- x is 2
  $x \in \{2\}$
- the strongest property of 2
  $\{2\}$
- 2 and 4 are even
  $\{2,4\} \subseteq \{0,2,4,6,8,\ldots\}$

- $\mathscr{D} = \mathbb{N}$
- $O \in \wp(\mathscr{D})$
- "x has property O"

Abstract properties

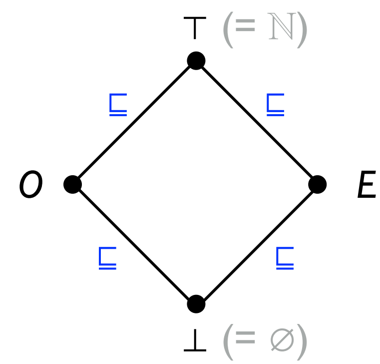
Abstract properties: $Q \in \mathscr{A}$

Abstract domain $\mathscr{A}$: encodes a subset of the concrete properties (e.g. a program logic, type terms, linear algebra, *etc*)

Poset: $\langle \mathscr{A}, \sqsubseteq, \sqcup, \sqcap, \ldots \rangle$

Partial order: $\sqsubseteq$ is *abstract implication*

Example of Abstract Properties



$\mathscr{A} = \{\perp, O, E, \top\}$

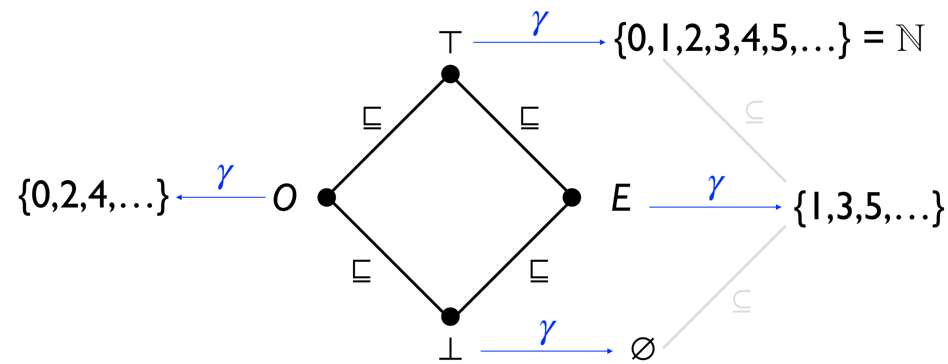
Concretization

Concretization $\gamma \in \mathscr{A} \longrightarrow \wp(\mathscr{D})$

$\gamma(Q)$ is the semantics (concrete meaning) of Q

γ is *increasing* (so $\sqsubseteq$ abstracts $\subseteq$)

Example of Concretization



Best abstraction

A concrete property $P \in \wp(\mathscr{D})$ has a best abstraction $Q \in \mathscr{A}$ iff

- it is sound (over-approximation):

  $P \subseteq \gamma(Q)$

- and more precise than any sound abstraction:

  $P \subseteq \gamma(Q') \implies Q \sqsubseteq Q' \implies \gamma(Q) \subseteq \gamma(Q')$

The best abstraction is unique (by antisymmetry)

Under-approximation is order-dual

Galois connection

Any $P \in \wp(\mathscr{D})$ has a (unique) best abstraction $\alpha(P)$ in $\mathscr{A}$ if and only if

$$\forall P \in \wp(\mathscr{D}) : \forall Q \in \mathscr{A} : \ \alpha(P) \sqsubseteq Q \iff P \subseteq \gamma(Q)$$

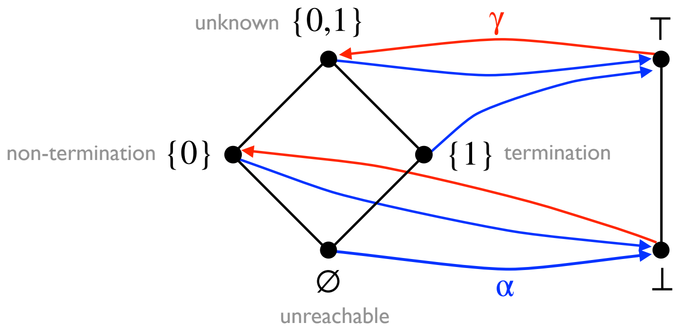
$\Rightarrow$: *over-approximation*
$\Leftarrow$: *best abstraction*

written

$$\langle \wp(\mathscr{D}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathscr{A}, \sqsubseteq \rangle$$

Examples

Needness/strictness analysis (80's)

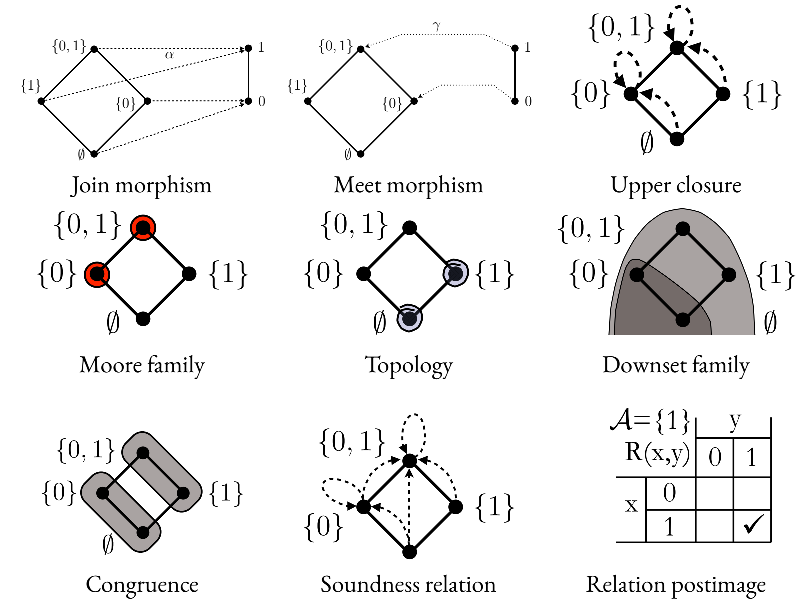


Similar abstraction ($\gamma(\top) \triangleq$ {true, false}) for scalable hardware symbolic trajectory evaluation STE (90)

Alan Mycroft: The Theory and Practice of Transforming Call-by-need into Call-by-value. Symposium on Programming 1980: 269-281

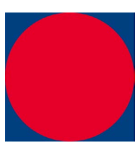
Carl-Johan H. Seger, Randal E. Bryant: Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories. Formal Methods in System Design 6(2): 147-189 (1995)

Equivalent mathematical structures

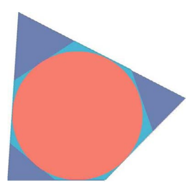


In absence of best abstraction?

Best abstraction of a disk by a rectangular parallelogram (intervals)



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



use only abstraction or concretization or widening (*)

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

Sound semantics abstraction

program	$\mathtt{P} \in \mathbb{L}$	programming language
standard semantics	$\mathsf{S}[\![\mathtt{P}]\!] \in \mathscr{D}$	semantic domain
collecting semantics	$\{\mathsf{S}[\![\mathtt{P}]\!]\} \in \wp(\mathscr{D})$	semantic property
abstract semantics	$\overline{\mathsf{S}}[\![\mathtt{P}]\!] \in \mathscr{A}$	abstract domain
concretization	$\gamma \in \mathscr{A} \longrightarrow \wp(\mathscr{D})$	
soundness	$\{\mathsf{S}[\![\mathtt{P}]\!]\} \subseteq \gamma(\overline{\mathsf{S}}[\![\mathtt{P}]\!])$	

i.e. $\mathsf{S}[\![\mathtt{P}]\!] \in \gamma(\overline{\mathsf{S}}[\![\mathtt{P}]\!])$, *P has abstract property* $\overline{\mathsf{S}}[\![\mathtt{P}]\!]$

Best abstract semantics

If $\langle \wp(\mathscr{D}), \subseteq\rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathscr{A}, \sqsubseteq\rangle$ then the best abstract semantics is the abstraction of the collecting semantics

$$\overline{S[\![P]\!]} \triangleq \alpha(\{S[\![P]\!]\})$$

Proof:

- It is *sound*: $\overline{S[\![P]\!]} \triangleq \alpha(\{S[\![P]\!]\}) \sqsubseteq \overline{S[\![P]\!]} \Longrightarrow \{S[\![P]\!]\} \subseteq \gamma(\overline{S[\![P]\!]}) \Longrightarrow S[\![P]\!] \in \gamma(\overline{S[\![P]\!]})$
- It is the *most precise*: $S[\![P]\!] \in \gamma(\overline{\overline{S[\![P]\!]}}) \Longrightarrow \{S[\![P]\!]\} \subseteq \gamma(\overline{\overline{S[\![P]\!]}}) \Longrightarrow \overline{S[\![P]\!]} \triangleq \alpha(\{S[\![P]\!]\}) \sqsubseteq \overline{\overline{S[\![P]\!]}}$ ∎

Calculational design of the abstract semantics

The (standard hence collecting) semantics are defined by composition of mathematical structures (such as set unions, products, functions, fixpoints, *etc*)

If you know best abstractions of properties, you also know best abstractions of these mathematical structures

So, by composition, you also know the best abstraction of the collecting semantics $\rightsquigarrow$ calculational design of the abstract semantics

Orthogonally, there are many styles of

- *semantics* (traces, relations, transformers,…)
- *induction* (transitional, structural, segmentation [POPL 2012])
- *presentations* (fixpoints, equations, constraints, rules [CAV 1995])

Example: functional connector

If $g = \langle \mathscr{C}, \subseteq\rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathscr{A}, \sqsubseteq\rangle$ then

$$g \Longmapsto g = \langle \mathscr{C} \xrightarrow{\nearrow} \mathscr{C}, \subseteq\rangle \xrightleftharpoons[\lambda F.\alpha\circ F\circ\gamma]{\lambda \overline{F}.\gamma\circ\overline{F}\circ\alpha} \langle \mathscr{A} \xrightarrow{\nearrow} \mathscr{A}, \sqsubseteq\rangle$$

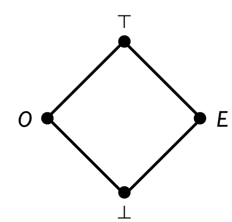
$$\begin{tikzcd}
\mathscr{A} \arrow[r, "\overline{F}"] \arrow[d, shift right, "\gamma"'] & \mathscr{A} \arrow[d, shift right, "\gamma"'] \\
\mathscr{C} \arrow[r, "F"] \arrow[u, shift right, "\alpha"'] & \mathscr{C} \arrow[u, shift right, "\alpha"']
\end{tikzcd}$$

g g

($\Longmapsto$ is a called a *Galois connector*)

Simple example

$F(x_2) = \{x+2 \mid x \in x_2\}$

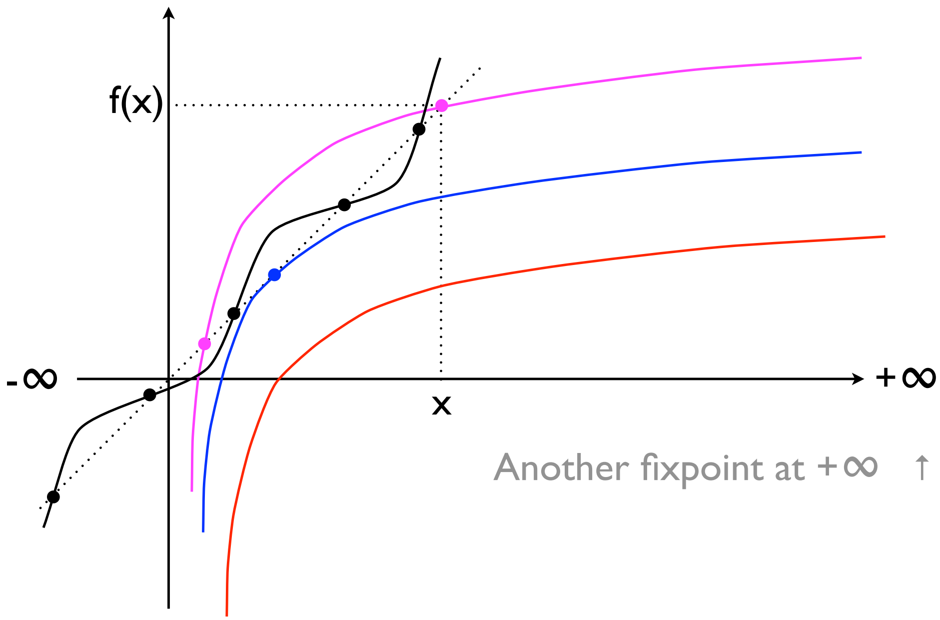


$\overline{F}(x_2)$	$\perp$	O	E	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
O	$\perp$	E	O	$\top$
E	$\perp$	O	E	$\top$
$\top$	$\perp$	$\top$	$\top$	$\top$

$$\begin{aligned}
&\alpha\circ F\circ\gamma(\perp)\\
&= \alpha(\{x+2 \mid x \in \varnothing\}\\
&= \alpha(\varnothing)\\
&= \perp
\end{aligned}$$

$$\begin{aligned}
&\alpha\circ F\circ\gamma(O)\\
&= \alpha(\{x+2 \mid x \in \gamma(O)\}\\
&= \alpha(\{x+2 \mid x \in \{1,3,5,\ldots\}\})\\
&= \alpha(\{3,5,7,\ldots\})\\
&= O
\end{aligned}$$

Fixpoints of increasing functions (Tarski)



Fixpoint abstraction

Best abstraction (completeness case)

if $\alpha \circ F = \overline{F} \circ \alpha$ then $\overline{F} = \alpha \circ F \circ \gamma$ and $\alpha(\text{lfp } F) = \text{lfp } \overline{F}$

e.g. semantics, proof methods, static analysis of finite state systems

Best approximation (incompleteness case)

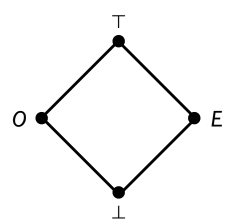
if $\overline{F} = \alpha \circ F \circ \gamma$ but $\alpha \circ F \sqsubseteq \overline{F} \circ \alpha$ then $\alpha(\text{lfp } F) \sqsubseteq \text{lfp } \overline{F}$

e.g. static analysis of infinite state systems

idem for equations, constraints, rule-based deductive systems, *etc*

Simple Example

```
0: x := 1;
1: while x < 10 do
2: x := x + 2;
3: od;
4:
```



$(x_0,\ldots,x_4)=F(x_0,\ldots,x_4)$

$x_0 = \{\ldots\text{-}2,\text{-}1,0,1,2,\ldots\}$
$x_1 = \{1\}$
$x_2 = (x_1 \cup x_3) \cap \{\ldots,\text{-}8\text{-}9\}$
$x_3 = \{x+2 \mid x \in x_2\}$
$x_4 = (x_1 \cup x_3) \cap \{10,11,\ldots\}$

$(x_0,\ldots,x_4)=\overline{F}(x_0,\ldots,x_4)$

$x_0 = \top$
$x_1 = O$
$x_2 = (x_1 \sqcup x_3) \sqcap \top$
$x_3 = x_2 \oplus E$
$x_4 = (x_1 \sqcup x_3) \sqcap \top$

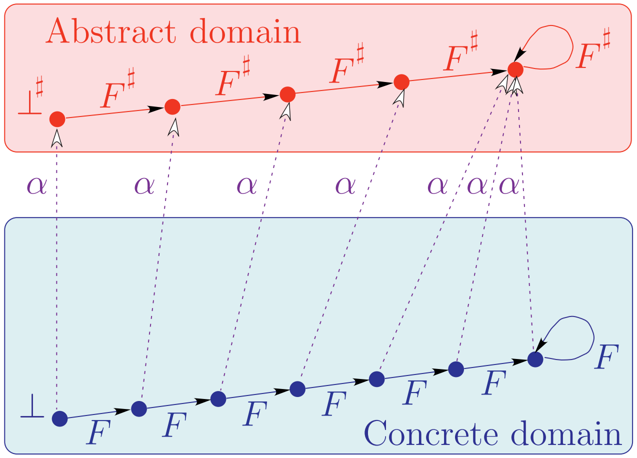
Iterative resolution

$x_0 = \top$
$x_1 = O$
$x_2 = (x_1 \sqcup x_3) \sqcap \top$
$x_3 = x_2 \oplus E$
$x_4 = (x_1 \sqcup x_3) \sqcap \top$

iteration 0	1	2	3	4	5	6
$x_0 = \bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$x_1 = \bot$	$\bot$	O	O	O	O	O
$x_2 = \bot$	$\bot$	$\bot$	O	O	O	O
$x_3 = \bot$	$\bot$	$\bot$	$\bot$	O	O	O
$x_4 = \bot$	$\bot$	$\bot$	$\bot$	$\bot$	O	O

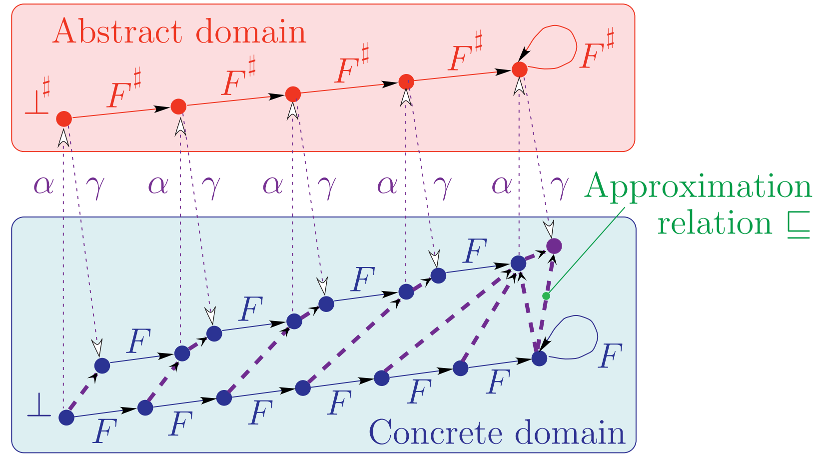
↑ fixpoint

Exact fixpoint abstraction



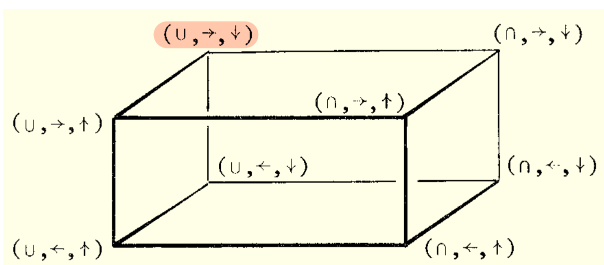
$$\alpha \circ F = F^\sharp \circ \alpha \;\Rightarrow\; \alpha(\mathit{lfp}\, F) = \mathit{lfp}\, F^\sharp$$

Approximate fixpoint abstraction



$$\mathit{lfp}\, F \sqsubseteq \gamma(\mathit{lfp}\, F^\sharp)$$

Duality



Order duality: join (∪) or meet (∩)

Inversion duality: forward (→) or backward (← = $(\rightarrow)^{-1}$)

Fixpoint duality: least (↓) or greatest (↑)

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

Why abstracting properties of semantics, not semantics or models?

Understandings of Abstract Interpretation

1. Abstract interpretation = a non-standard semantics (computations on values in the standard semantics are replaced by computations on abstract values) $\Longrightarrow$ extremely limited
2. Abstract interpretation = an abstraction of the standard semantics $\Longrightarrow$ limited
3. Abstract interpretation = an abstraction of properties of the standard semantics $\Longrightarrow$ more

i.e. (1) is an abstraction of (2), (2) is an abstraction of (3)

Example: trace semantics properties

Domain of [in]finite traces on states: Π

"Standard" trace semantics domain: $\mathcal{D} = \wp(\Pi)$

"Standard" trace semantics $S[\![\mathtt{P}]\!] \in \mathcal{D} = \wp(\Pi)$

Domain of semantics properties is $\wp(\mathcal{D}) = \wp(\wp(\Pi))$

Collecting semantics $C[\![\mathtt{P}]\!] \triangleq \{S[\![\mathtt{P}]\!]\} \in \wp(\mathcal{D}) = \wp(\wp(\Pi))$

How to abstract the standard semantics?

The join abstraction:

$$\langle \wp(\wp(\Pi)), \subseteq\rangle \underset{\alpha_\cup}{\overset{\gamma_\cup}{\leftrightarrows}} \langle \wp(\Pi), \subseteq\rangle$$

$$\alpha_\cup(X) \triangleq \bigcup X$$

$$\gamma_\cup(Y) \triangleq \wp(Y)$$

Join abstraction of the collecting semantics:

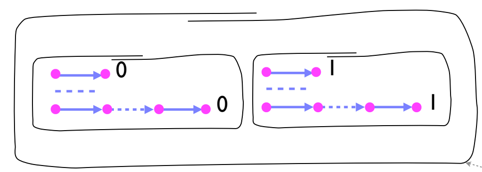
$$\alpha_\cup(C[\![\mathtt{P}]\!]) \triangleq \bigcup\{S[\![\mathtt{P}]\!]\} \triangleq S[\![\mathtt{P}]\!]$$

(*i.e.* the semantics is the join abstraction of its strongest property)

Loss of information

"Always terminate with the same value, either 0 or 1"

$P =$

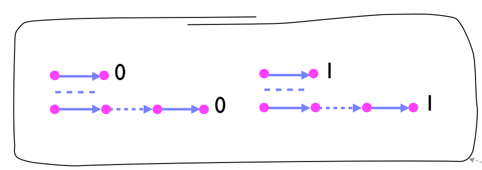


$P \in \wp(\wp(\Pi))$

always the same result

Join abstraction:

$\alpha_\cup(P) =$



$\alpha_\cup(P) \in \wp(\Pi)$

results can be different

"Always terminate, either with 0 or 1"

Limitations of the union abstraction

Complete iff any property of the semantics $S[\![P]\!]$ is also valid for any subset $\gamma(S[\![P]\!]) = \wp(S[\![P]\!])$:

- Examples: safety, liveness
- Counter-example: security (e.g. authentication using a random cryptographic nonce)

Exact abstractions

Exact abstractions

The concrete properties of the standard semantics $S[\![P]\!]$ that you want to prove can always be proved in the abstract (which is simpler):

$$\forall Q \in \mathscr{A}: S[\![P]\!] \in \gamma(Q) \iff S[\![\overline{P}]\!] \sqsubseteq Q$$

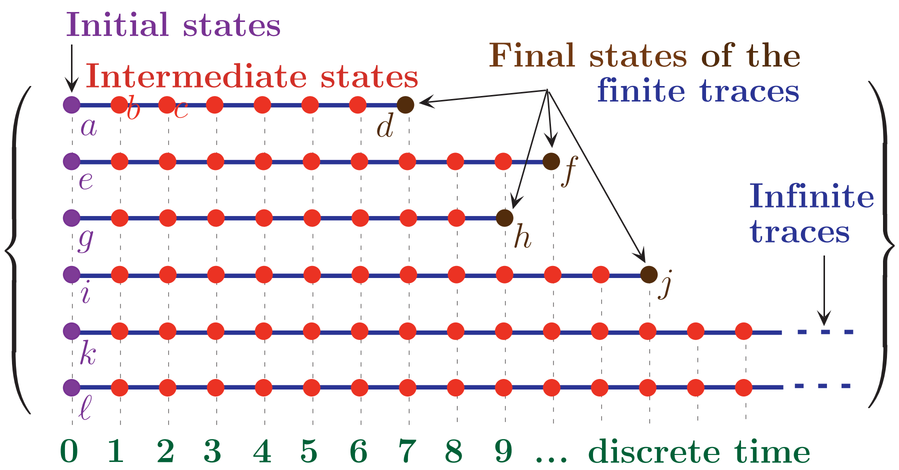
where

$$S[\![\overline{P}]\!] \triangleq \alpha \circ S[\![P]\!] \circ \gamma$$

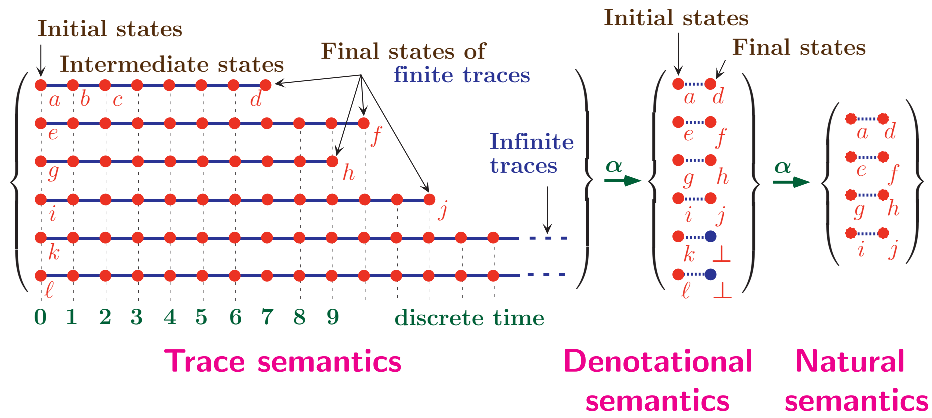
Example III of exact abstractions: semantics

Patrick Cousot: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theor. Comput. Sci. 277(1-2): 47-103 (2002)

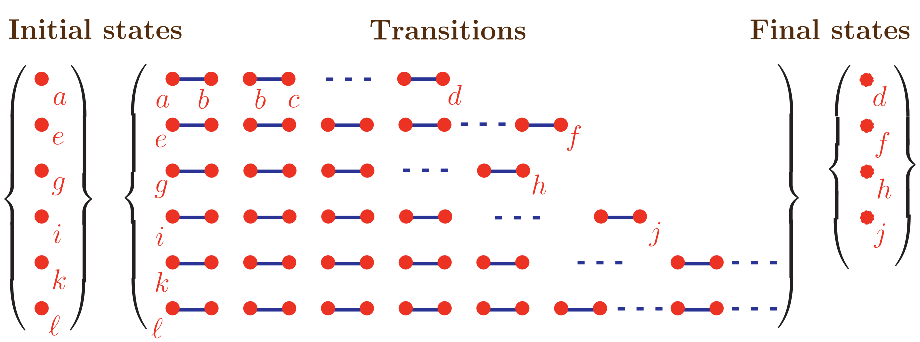
Trace semantics



Abstraction to denotational/natural semantics

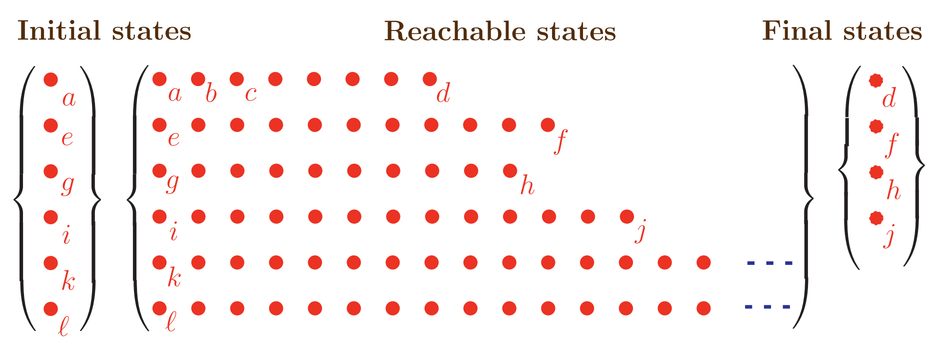


Abstraction to small-steps operational semantics



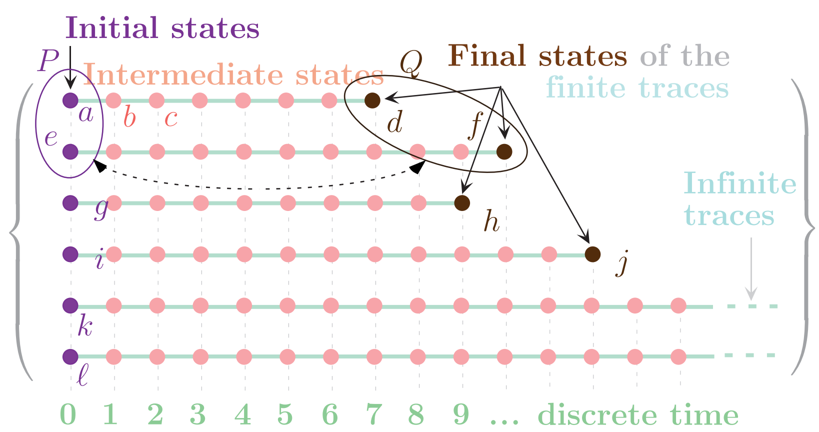
(Small-Step) Operational Semantics

Abstraction to reachability/invariance



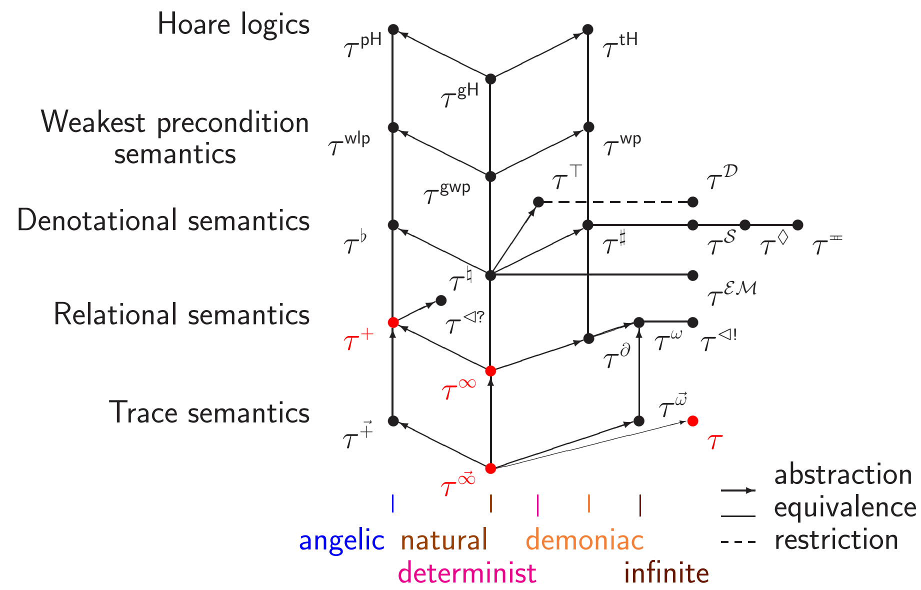
Partial Correctness / Invariance Semantics

Abstraction to Hoare logic



$\{P\}C\{Q\} \Leftrightarrow \{\bullet \mid \bullet \in P \wedge \bullet\!-\!\bullet\!-\!\bullet\!-\ldots-\!\bullet \in [\![C]\!]\} \subseteq Q$

Poset of semantics



Approximate abstractions

Approximate abstractions

The concrete properties of the standard semantics $S[\![\mathrm{P}]\!]$ that you want to prove may not always be provable in the abstract:

$$\forall Q \in \mathscr{A} : S[\![\mathrm{P}]\!] \in \gamma(Q) \quad \underset{\not\Rightarrow}{\Leftarrow} \quad \overline{S}[\![\mathrm{P}]\!] \sqsubseteq Q$$

where

$$\overline{S}[\![\mathrm{P}]\!] \stackrel{\Delta}{=} \alpha \circ S[\![\mathrm{P}]\!] \circ \gamma$$

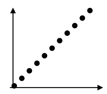
Why abstraction may be approximate?

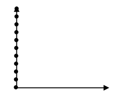
Example

$\{\ x = y \land 0 \leqslant x \leqslant 10\ \}$

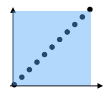
```
x := x - y;
```

$\{\ x = 0 \land\ 0 \leqslant y \leqslant 10\ \}$





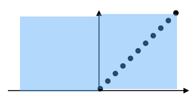
Interval abstraction:



$\{\ x \in [0, 10] \land y \in [0, 10]\ \}$

```
x := x - y;
```

$\{\ x \in [-10, 10] \land y \in [0, 10]\ \}$



(but for constants, the interval abstraction can't express equality)

Finite versus infinite abstractions

[In]finite abstractions

Given a program $\mathtt{P}$ and a program property Q which holds (*i.e.* lfp $F[\![\mathtt{P}]\!] \in Q$) there exists a most abstract abstraction in a finite domain $\mathscr{A}[\![\mathtt{P}]\!]$ to prove it (*)

Example:

x=0; while x<1 do x++ $\longrightarrow \{\bot, [0,0], [0,1], [-\infty,\infty]\}$

x=0; while x<2 do x++ $\longrightarrow \{\bot, [0,0], [0,1], [0,2], [-\infty,\infty]\}$

…

x=0; while x<n do x++ $\longrightarrow \{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,n], [-\infty,\infty]\}$

…

(*) Patrick Cousot: Partial Completeness of Abstract Fixpoint Checking. SARA 2000: 1-25

[In]finite abstractions

No such domain exists for infinitely many programs

1. $\bigcup_{\mathtt{P} \in \mathbb{L}} \mathscr{A}[\![\mathtt{P}]\!]$ is infinite

   Example: $\{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,n], [0,n+1], \ldots, [-\infty,\infty]\}$

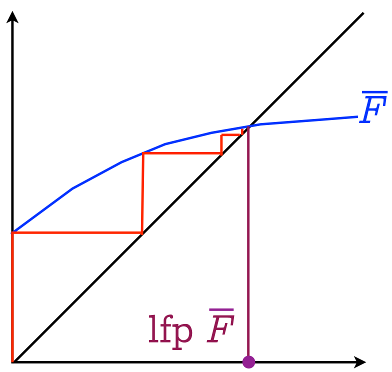
2. $\lambda \mathtt{P} \in \mathbb{L} \cdot \mathscr{A}[\![\mathtt{P}]\!]$ is not computable (for undecidable properties)

$\Longrightarrow$ finite abstractions will fail infinitely often while infinite abstractions will succeed!

Fixpoint approximation in infinite abstractions

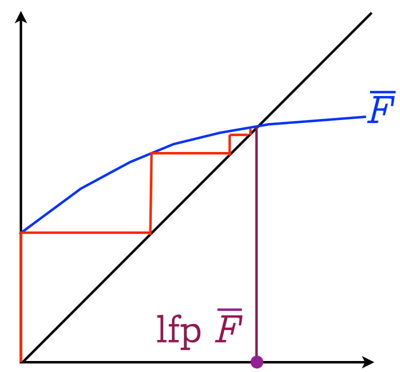
Abstract Induction
(in non-Noetherian domains)

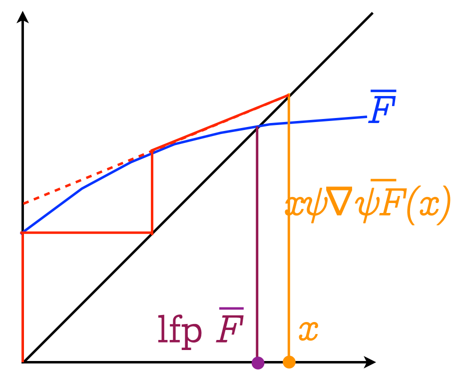
Convergence acceleration



Infinite iteration

Convergence acceleration





Infinite iteration

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

(*) Javier Esparza, Stefan Kiefer, Michael Luttenberger: Newtonian program analysis. J. ACM 57(6): 33 (2010)

Problem with infinite abstractions

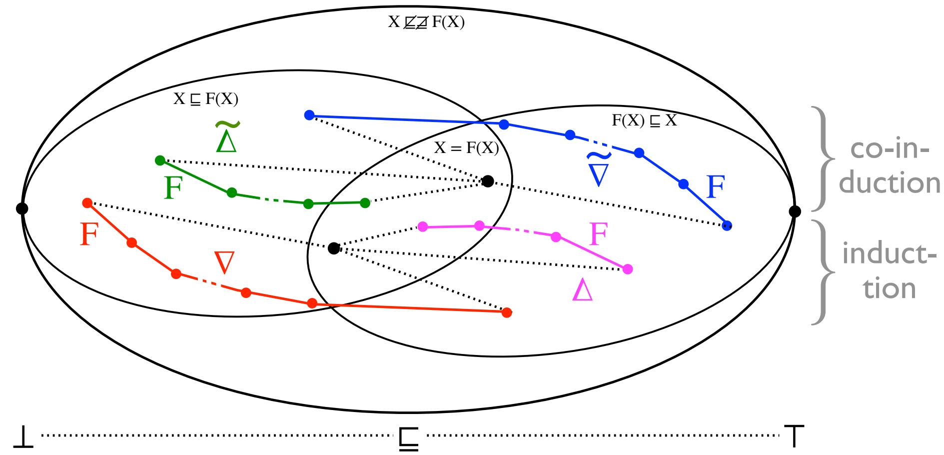
For non-Noetherian iterations, we need

- finitary abstract induction, and
- finitary passage to the limit

$$X^0 = \bot, \ldots, X^{n+1} = \mathfrak{F}(X^0, \ldots, X^n, F(X^0), \ldots, F(X^n)), \ldots, \lim_{n \to \infty} X^n$$

		iteration converging	
	$\mathfrak{F}$	above the limit	below the limit
Iteration starting from	below the limit	widening $\triangledown$	dual narrowing $\tilde{\triangle}$
	above the limit	narrowing $\triangle$	dual widening $\tilde{\triangledown}$

[Semi-]dual abstract induction methods



(separate from termination conditions)

Examples of widening/narrowing

Abstract induction for intervals:

- a widening [1,2]

```
(x ∇̄ y) = cas x ∈ V_a, y ∈ V_a dans
   ┼ □, ? ==> y ;
   ┼ ?, □ ==> x ;
   ┼ [n_1,m_1],[n_2,m_2] ==>
        [si n_2 < n_1 alors -∞ sinon n_1 fsi ;
           si m_2 > m_1 alors +∞ sinon m_1 fsi] ;
   fincas ;
```

```
[a_1, b_1] ∇̄ [a_2, b_2] =
    [if a_2 < a_1 then -∞ else a_1 fi,
         if b_2 > b_1 then +∞  else b_1 fi]
```

- a narrowing [2]

```
[a_1,b_1] Δ̄ [a_2,b_2] =
    [if a_1 = -∞ then a_2 else MIN (a_1,a_2),
         if b_1 = +∞ then b_2 else MAX (b_1,b_2)]
```

[1] Patrick Cousot, Radhia Cousot: Vérification statique de la cohérence dynamique des programmes, Rapport du contrat IRIA-SESORI No 75-032, 23 septembre 1975.
[2] 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

On widening/narrowing/and their duals

Because the abstract domain is non-Noetherian, *any* widening/narrowing/duals can be *strictly* improved infinitely many times (*i.e.* no best widening)

E.g. widening with thresholds [1]

$$\forall x \in \bar{L}_2, \bot \, \nabla_2(j) \, x = x \, \nabla_2(j) \, \bot = x$$

$$[l_1, u_1] \, \nabla_2(j) \, [l_2, u_2] = [\textit{if } 0 \le l_2 < l_1 \textit{ then } 0 \textit{ elsif } l_2 < l_1 \textit{ then } -b-1 \textit{ else } l_1 \textit{ fi}, \; \textit{if } u_1 < u_2 \le 0 \textit{ then } 0 \textit{ elsif } u_1 < u_2 \textit{ then } b \textit{ else } u_1 \textit{ fi}]$$

Any *terminating* widening is *not* increasing (in its first parameter)

Any abstraction done with Galois connections *can be done* with widenings (*i.e.* a widening calculus)

[1] Patrick Cousot, Semantic foundations of program analysis, Ch. 10 of Program flow analysis: theory and practice, N. Jones & S. Muchnich (eds), Prentice Hall, 1981.

Infinitary static analysis with abstract induction

Widening

$\langle \mathscr{A}, \sqsubseteq \rangle$ poset

$\triangledown \in \mathscr{A} \times \mathscr{A} \longrightarrow \mathscr{A}$

Sound widening (upper bound):

$\forall\, x, y \in \mathscr{A} : x \sqsubseteq x \triangledown y \;\wedge\; y \sqsubseteq x \triangledown y$

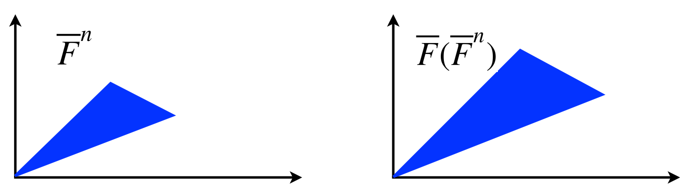
Terminating widening: for any $\langle x^n \in \mathscr{A}, n \in \mathbb{N} \rangle$, the sequence $y^0 \triangleq x^0, \ldots, y^{n+1} \triangleq y^n \triangledown x^n, \ldots$ is *ultimately stationary* ($\exists \varepsilon \in \mathbb{N}: \forall n \geqslant \varepsilon: y^n = y^\varepsilon$)

(Note: sound and terminating are independent properties)

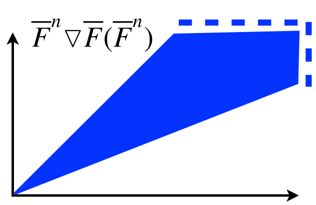
Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

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 for static analysis

Problem: compute I such that $\mathrm{lfp}^{\sqsubseteq} F \sqsubseteq I \sqsubseteq Q$

Compute I as the limit of the iterates:

- $X^0 \triangleq \bot$,
- $X^{n+1} \triangleq X^n$ when $F(X^n) \sqsubseteq X^n$ so $I = X^n$
- $X^{n+1} \triangleq (X^n \triangledown F(X^n)) \vartriangle Q$ otherwise

I can be improved by an iteration with narrowing $\vartriangle$

Check that $F(I) \sqsubseteq Q$

Example: Astrée

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

Dual narrowing

$\langle \mathscr{A}, \sqsubseteq \rangle$ poset

$\widetilde{\triangle} \in \mathscr{A} \times \mathscr{A} \longrightarrow \mathscr{A}$

Sound dual narrowing (interpolation):

$$\forall x, y \in \mathscr{A} : x \sqsubseteq y \implies x \sqsubseteq x \,\widetilde{\triangle}\, y \sqsubseteq y$$

Terminating dual narrowing: for any $\langle x^n \in \mathscr{A}, n \in \mathbb{N} \rangle$, the sequence $y^0 \triangleq x^0, \ldots, y^{n+1} \triangleq y^n \,\widetilde{\triangle}\, x^n, \ldots$ is ultimately stationary ($\exists \varepsilon \in \mathbb{N}: \forall n \geqslant \varepsilon: y^n = y^\varepsilon$)

(Note: sound and terminating are independent properties)

Cousot, P. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French). Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, France 1978.

Iteration with dual narrowing for static checking

Problem: find I such that $\mathrm{lfp}^{\sqsubseteq} F \sqsubseteq I \sqsubseteq Q$

Compute I as the limit of the iterates:

- $X^0 \triangleq \bot$,
- $X^{n+1} \triangleq X^n$ when $F(X^n) \sqsubseteq X^n$ so $I = X^n$
- $X^{n+1} \triangleq F(X^n) \,\widetilde{\triangle}\, Q$, otherwise

Check that $F(I) \sqsubseteq Q$

Example: First-order logic + Graig interpolation (with some choice of one of the solutions, control of combinatorial explosion, and convergence enforcement)

Kenneth L. McMillan: Applications of Craig Interpolants in Model Checking. TACAS 2005: 1-12

Industrialization

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

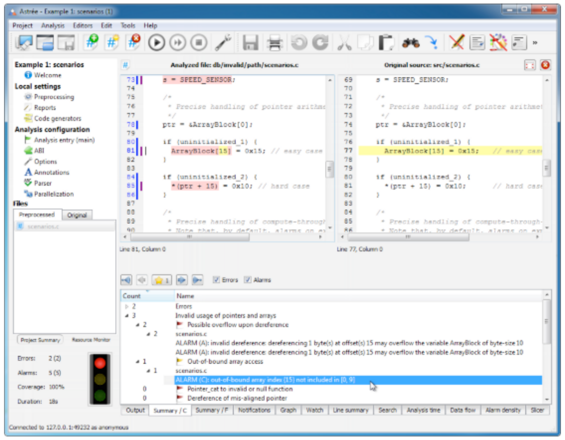
David Delmas, Jean Souyris: Astrée: From Research to Industry. SAS 2007: 437-451

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

Astrée

Commercially available: `www.absint.com/astree/`

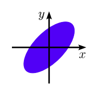


<u>Effectively</u> used in production to qualify truly large and complex software in transportation, communications, medicine, *etc*

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

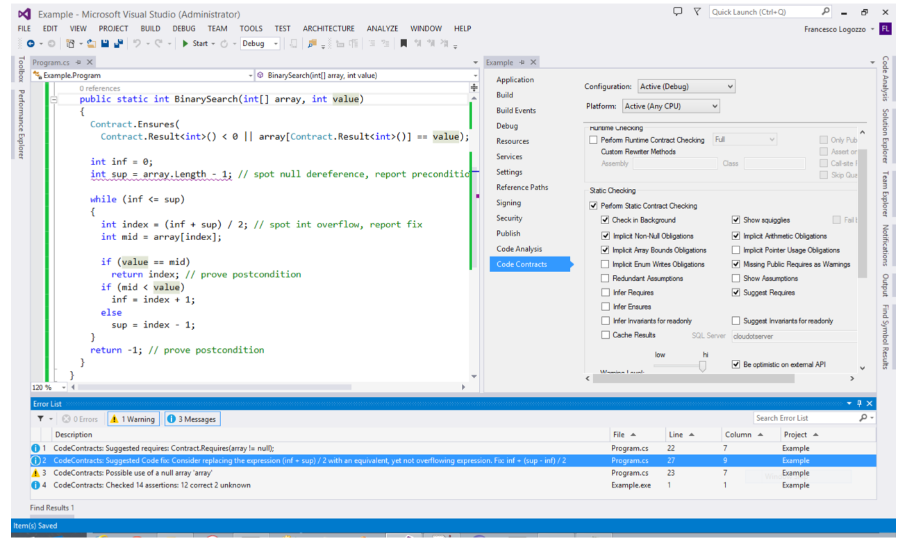
Example of domain-specific abstraction: ellipses

```
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; }
}
```



Code Contract Static Checker (cccheck)

https://github.com/Microsoft/CodeContracts (public domain)



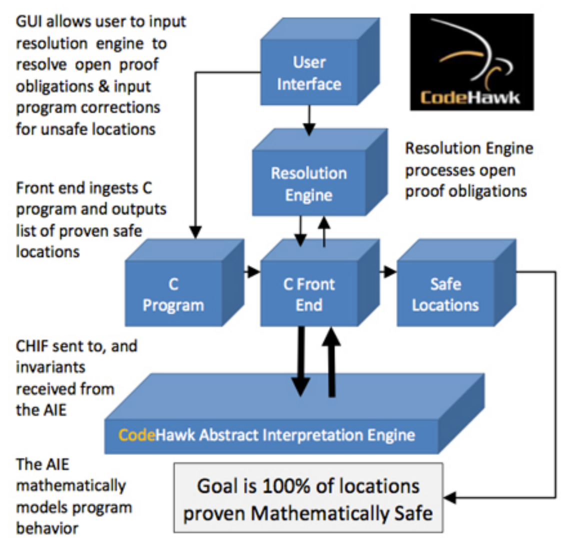
Manuel Fähndrich, Francesco Logozzo: **Static Contract Checking with Abstract Interpretation**. *FoVeOOS 2010*: 10-30

Comments on screenshot (courtesy Francesco Logozzo)

1. A screenshot from Clousot/cccheck on the classic binary search.
2. The screenshot shows from left to right and top to bottom
   1. C# code + CodeContracts with a buggy BinarySearch
   2. cccheck integration in VS (right pane with all the options integrated in the VS project system)
   3. cccheck messages in the VS error list
3. The features of cccheck that it shows are:
   1. basic abstract interpretation:
      1. the loop invariant to prove the array access correct and that the arithmetic operation may overflow is inferred fully automatically
      2. different from deductive methods as e.g. ESC/Java or Boogie where the loop invariant must be provided by the end-user
   2. inference of necessary preconditions:
      1. Clousot finds that array may be null (message 3)
      2. Clousot suggests and propagates a necessary precondition invariant (message 1)
   3. array analysis (+ disjunctive reasoning):
      1. to prove the postcondition should infer property of the content of the array
      2. please note that the postcondition is true even if there is no precondition requiring the array to be sorted.
   4. verified code repairs:
      1. from the inferred loop invariant does not follow that index computation does not overflow

Example III: CodeHawk

- http://www.kestreltechnology.com



Conclusion

Abstract interpretation

Intellectual tool (not to be confused with its specific application to iterative static analysis with $\triangledown$ & $\vartriangle$)

No cathedral would have been built without plumb-line and square, certainly not enough for skyscrapers:

Powerful tools are needed for progress and applicability of formal methods

Abstract interpretation

Varieties of researchers in formal methods:

(i) explicitly use abstract interpretation, and are happy to extend its scope and broaden its applicability

(ii) implicitly use abstract interpretation, and hide it

(iii) pretend to use abstract interpretation, but misuse it

(iv) don't know that they use abstract interpretation, but would benefit from it

Never too late to upgrade

The End

The End
Thank You