« Abstract interpretation and a range of applications »

Patrick Cousot
École normale supérieure
45 rue d'Ulm
75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Radhia Cousot
École polytechnique
91128 Palaiseau cedex, France
Radhia.Cousot@polytechnique.fr
www.enseignement.polytechnique.fr/profs/informatique/Radhia.Cousot/

Dipartimento di Informatica — Università Ca'Foscari di Venezia — October 23rd, 2006

Abstract

Since almost any complex software has bugs, researchers have developed program correctness proof methods. This consists in defining a semantics formally describing the executions of a program and then in proving a theorem stating that these executions have a given property (for example that an expected result is provided in a finite time). Fundamental mathematical results show that these proofs cannot be done automatically by computers.

Confronted with this fundamental difficulty, abstract interpretation proceeds by correct approximation of the semantics. If the approximation is coarse enough, it is computable. If it is precise enough, it yields a correctness proof. The goal is therefore to find cheap approximations which are precise enough.

We will introduce a few elements of abstract interpretation and explain how to formalize the abstraction of semantic properties so as to obtain computable approximations leading to effective algorithms for the static analysis of the possible behaviours of programs.

Finally, we will describe an example of application of the theory to the proof of absence of runtime errors on synchronous control/command and underly the difficulties (such as floating point computations). This approach was applied with success to the verification of the electric flight control of the A380.

Plan

- The importance of software
- Why software is bugged?
- What can be done about bugs?
- Abstract interpretation
  - (1) a very informal introduction
  - (2) a few elements
  - (3) a simple example of application
  - (4) a range of applications
  - (5) application to the A380 flight control software
- Perspectives

The importance of software

Software is hidden everywhere

Origin of accidents (metro)

– Paris métro line 12 accident [1]: the driver was going too fast

– Roma metro line A accident [2]: the driver was given OK to ignore red light in tunnel

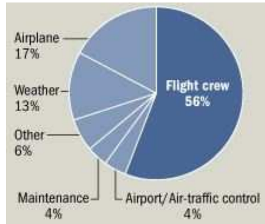
– New high-speed métro line 14 (Météor): fully automated, no operators

[1] On August 30th, 2000, at the Notre-Dame-de-Lorette métro station in Paris, a car flipped over on its side and slid to a stop just a few feet from a train stopped on the opposite platform (24 injured).

[2] On October 17th, 2006

Origin of accidents (aviation)

Worldwide analysis of the primary cause of major commercial-jet accidents betwwen 1995 and 2004 as determined by the investigating authority [3] [1]



Référence

[1] D. Michaels & A. Pasztor. *Incidents Prompt New Scrutiny of Airplane Software Giltches* citing its Boeing source. Wall Street Journal, Vol. CCXLVII, No 125, 30 mai 2006.

[3] includes only accidents with known causes.

Software replaces human operators

– Computer control is recognized as the safest and less expansive way to eliminate human mistakes

– Software is massively present in all mission-critical and safety-critical industrial infrastructures

Why software is bugged?

(1) Software gets huge

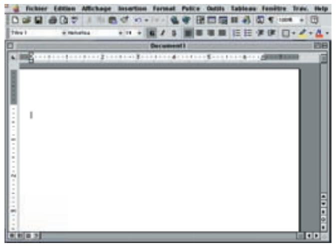
As computer hardware capacity grows...

ENIAC
5,000 flops [4]

NEC Earth Simulator
35×10^{12} flops [5]

4 Floating point operations per second
5 10^{12} = Thousand Billion

Software size grows...

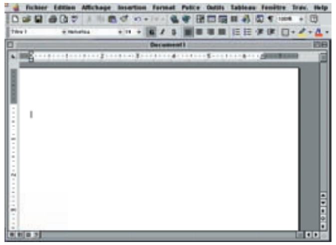


Text editor
1,700,000 lines of C [6]

Operating system
35,000,000 lines of C [7]

6 3 months for full-time reading of the code
7 5 years for full-time reading of the code

. . . and so does the number of bugs



Text editor
1,700,000 lines of C[6]
1,700 bugs (estimation)

Operating system
35,000,000 lines of C[7]
30,000 known bugs

[6] 3 months for full-time reading of the code
[7] 5 years for full-time reading of the code

(2) Computers are finite

Computers are finite

- Scientists use mathematics to deal with continuous, infinite structures (e.g. $\mathbb{R}$)
- Computers can only handle discrete, finite structures

Putting big things into small containers

- Numbers are encoded onto a limited number of bits (*binary digits*)
- Some operations may overflow (e.g. integers: 32 bits $\times$ 32 bits $=$ 64 bits)
- Using different number sizes (32, 64, . . . bits) can also be the source of overflows

The Ariane 5.01 maiden flight

– June 4^{th}, 1996 was the maiden flight of Ariane 5

(3) Computers go round

The Ariane 5.01 maiden flight failure

– June 4^{th}, 1996 was the maiden flight of Ariane 5

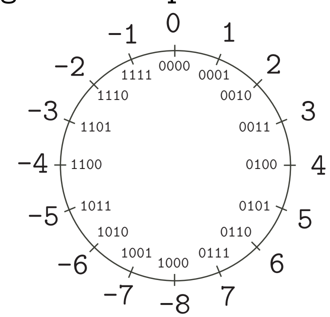
– The launcher was detroyed after 40 seconds of flight because of a software overflow[8]

[8] A 16 bit piece of code of Ariane 4 had been reused within the new 32 bit code for Ariane 5. This caused an uncaught overflow, making the launcher uncontrolable.

Modular arithmetic...

– Todays, computers avoid integer overflows thanks to modular arithmetic

– Example: integer 2's complement encoding on 8 bits



Modular arithmetic is not very intuitive

```
# -1073741823 / -1;;
- : int = 1073741823
# -1073741824 / -1;;
- : int = -1073741824
```

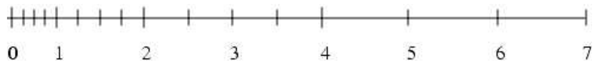

(4) Computers do round

Mapping many to few

– Reals are mapped to floats (floating-point arithmetic)

$$\pm d_0.d_1d_2\ldots d_{p-1}\beta^e$$ [9]

– For example on 6 bits (with $p = 3$, $\beta = 2$, $e_{\text{min}} = -1$, $e_{\text{max}} = 2$), there are 32 normalized floating-point numbers. The 16 positive numbers are



[9] where - $d_0 \neq 0$,
- p is the number of significative digits,
- β is the basis (2), and
- e is the exponant ($e_{\text{min}} \leq e \leq e_{\text{max}}$)

Rounding

– Computations returning reals that are not floats, must be rounded

– Most mathematical identities on $\mathbb{R}$ are no longer valid with floats

– Rounding errors may either compensate or accumulate in long computations

– Computations converging in the reals may diverge with floats (and ultimately overflow)

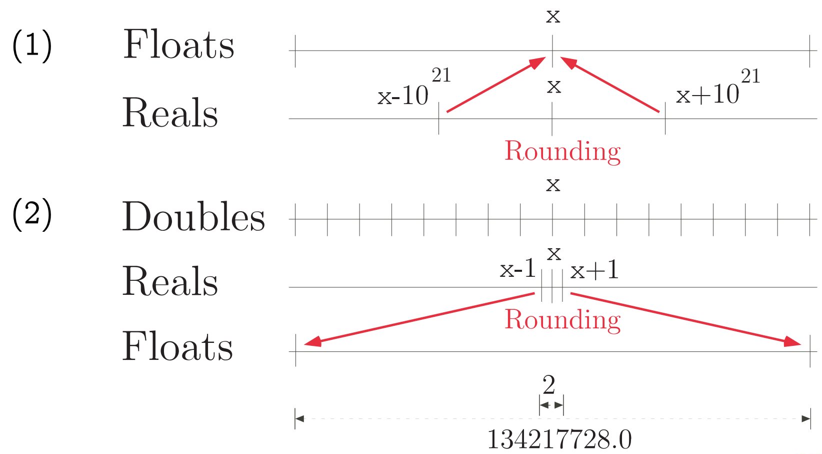
Example of rounding error

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

$$(x + a) - (x - a) \neq 2a$$

Explanation of the huge rounding error



Example of rounding error

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951487.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
0.000000
```

$$(x + a) - (x - a) \neq 2a$$

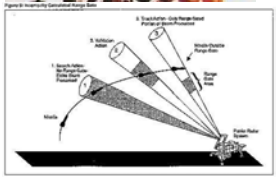
Example of accumulation of small rounding errors

```
% ocaml
        Objective Caml version 3.08.1
# let x = ref 0.0;;
val x : float ref = {contents = 0.}
# for i = 1 to 1000000000 do
     x := !x +. 1.0/.10.0
  done; x;;
- : float ref = {contents = 99999998.7454178184}
```

since $(0.1)_{10} = (0.0001100110011001100\ldots)_2$

The Patriot missile failure

- "On February 25$^{\text{th}}$, 1991, a Patriot missile … failed to track and intercept an incoming Scud[10]."
- The software failure was due to a cumulated rounding error[11]



[10] This Scud subsequently hit an Army barracks, killing 28 Americans.

[11]
- "Time is kept continuously by the system's internal clock in tenths of seconds"
- "The system had been in operation for over 100 consecutive hours"
- "Because the system had been on so long, the resulting inaccuracy in the time calculation caused the range gate to shift so much that the system could not track the incoming Scud"

What can be done about bugs?

Warranty

Excerpt from an GPL open software licence:

> NO WARRANTY. … *BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.*

You get nothing for free!

Warranty

Excerpt from Microsoft software licence:

> DISCLAIMER OF WARRANTIES. … *MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE, AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND MICROSOFT AND ITS SUPPLIERS HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. …*

You get nothing for your money either!

Traditional software validation methods

– The law cannot enforce more than "best practice"
– Manual software validation methods (code reviews, simulations, tests, etc.) do not scale up
– The capacity of programmers/computer scientists remains essentially the same
– The size of software teams cannot grow significantly without severe efficiency losses

Mathematics and computers can help

– Software behavior can be mathematically formalized → semantics
– Computers can perform semantics-based program analyses to realize verification → static analysis
  - but computers are finite so there are intrinsic limitations → undecidability, complexity
  - which can only be handled by semantics approximations → abstract interpretation

Interprétation abstraite

There are two fundamental concepts in computer science (and in sciences in general) :

– **Abstraction** : to reason on complex systems
– **Approximation** : to make effective undecidable computations

These concepts are formalized by abstract interpretation

References

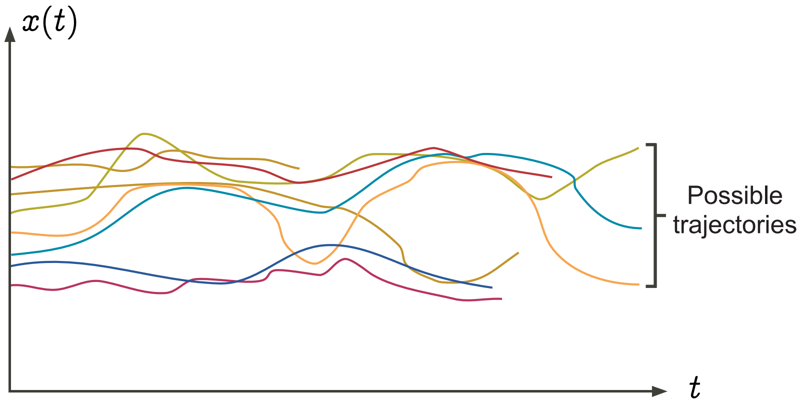
[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *ACM POPL*.

[Thesis '78] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse ès sci. math. Grenoble, march 1978.

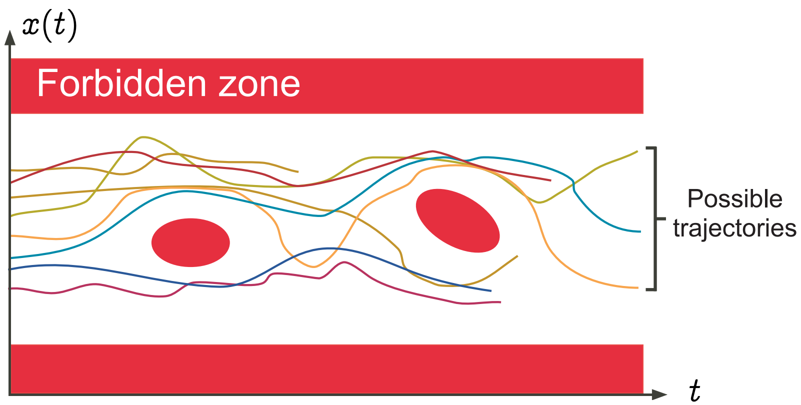
[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *ACM POPL*.

Abstract interpretation (1) a very informal introduction

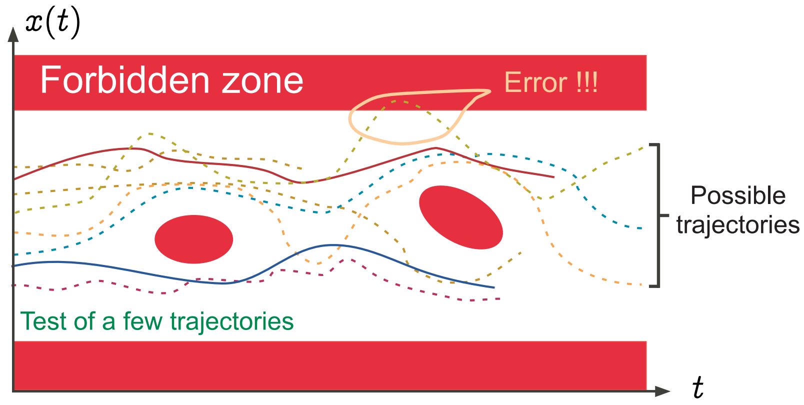
Operational semantics



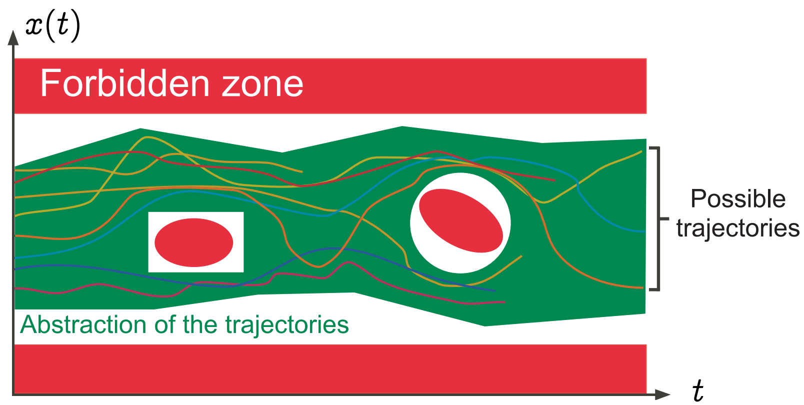
Safety property



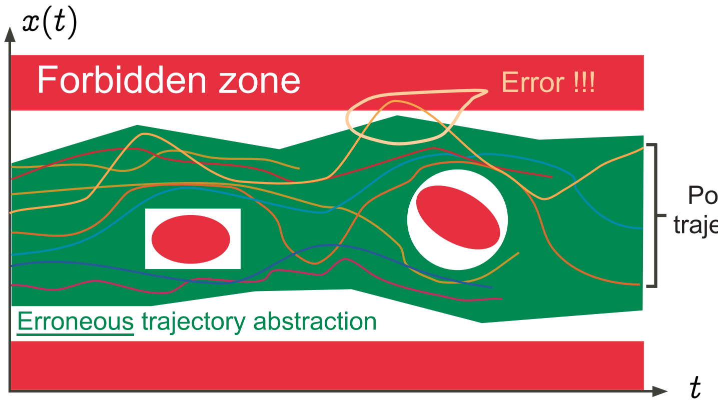
Test/debugging is unsafe



Abstract interpretation is safe

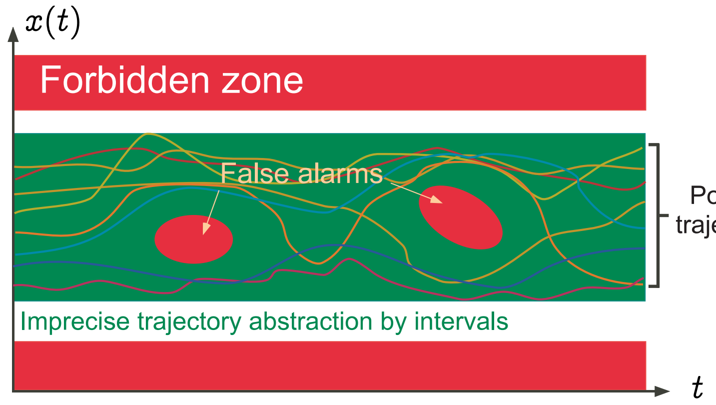


Soundness requirement: erroneous abstraction [12]

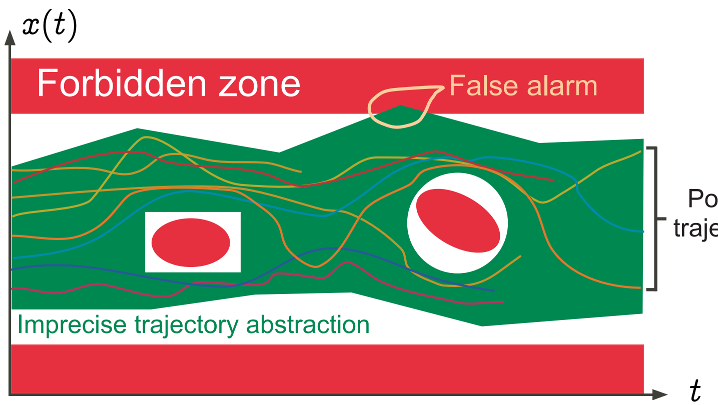


[12] This situation is always excluded in static analysis by abstract interrpetation.

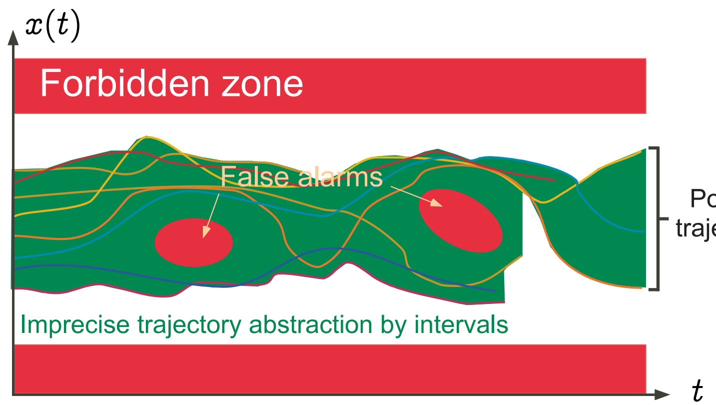
Global interval abstraction → false alarms



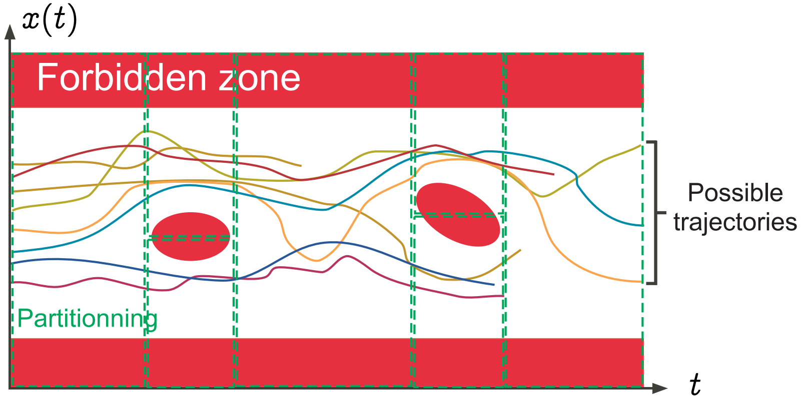
Imprecision ⇒ false alarms



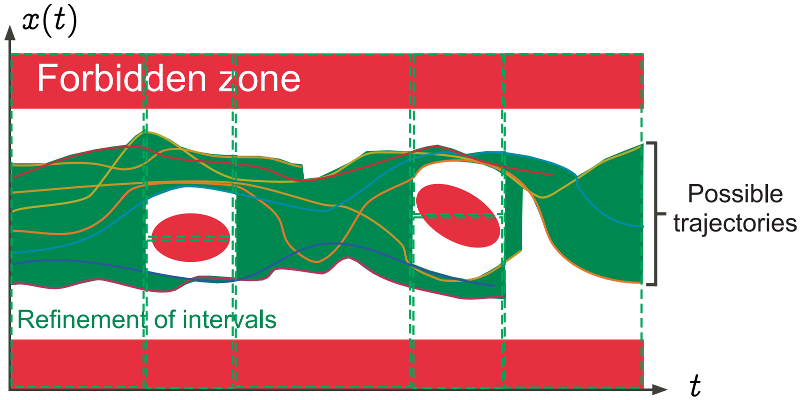
Local interval abstraction → false alarms



Refinement by partitionning



Intervals with partitionning



Abstract interpretation (2) a few elements

(2.1) Program semantics

Description of a computation step

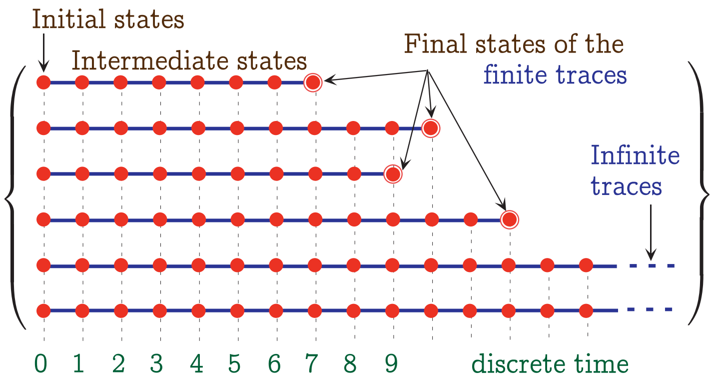
– Transition system $\langle \Sigma, \tau \rangle$, states $\Sigma = \{\bullet, \ldots, \bullet \ldots\}$, transitions $\tau = \{\bullet \longrightarrow \bullet, \ldots, \bullet \longrightarrow \bullet \ldots\}$
– Example
  - States : $\langle p, v \rangle$, p is a program point, v assigns values to variables
  - Transitions $\langle p, v \rangle \longrightarrow \langle p', v' \rangle$ for assignment:

```
p:
   X = X + 1;
p'
```

$v'(\mathtt{X}) = v(\mathtt{X}) + 1$ if $v(\mathtt{X}) < \mathtt{maxint}$
$v'(\mathtt{Y}) = v(\mathtt{Y})$ if $\mathtt{Y} \neq \mathtt{X}$

Blocking state ($\bullet$) if $v(\mathtt{X}) \geqslant \mathtt{maxint}$.

Description of a complete computation by a trace



States $\Sigma = \{\bullet, \ldots, \bullet \ldots\}$, transitions $\tau = \{\bullet \longrightarrow \bullet, \ldots, \bullet \longrightarrow \bullet \ldots\}$

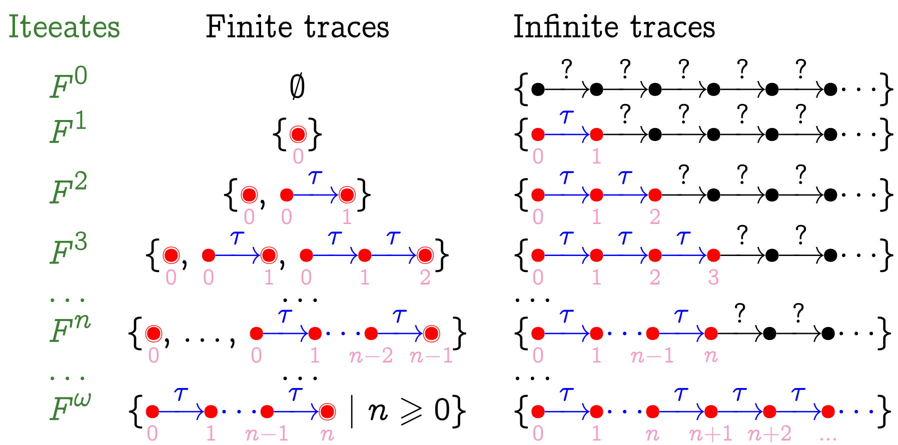
Least Fixpoint Trace Semantics

Traces $= \{\bullet \mid \bullet$ is a final state$\}$
$\cup \ \{\bullet\!-\!\bullet\!-\ldots-\!\bullet \mid \bullet\!-\!\bullet$ is a transition step & $\bullet\!-\ldots-\!\bullet \in$ Traces$^+\}$
$\cup \ \{\bullet\!-\!\bullet\!-\ldots-\ldots \mid \bullet\!-\!\bullet$ is a transition step & $\bullet\!-\ldots-\ldots \in$ Traces$^\infty\}$

- In general, the equation has multiple solutions;
- Choose the least one for the computational ordering:

"more finite traces & less infinite traces".

Iterative computation of the trace semantics



Trace Semantics, Formally

Trace semantics of a transition system $\langle \Sigma, \tau \rangle$:

- $\Sigma^+ \stackrel{\text{def}}{=} \bigcup_{n>0} [0, n[\longmapsto \Sigma$ finite traces
- $\Sigma^w \stackrel{\text{def}}{=} [0, \omega[\longmapsto \Sigma$ infinite traces
- $S = \mathsf{lfp}^{\sqsubseteq}\ F \in \Sigma^+ \cup \Sigma^\omega$ trace semantics
- $F(X) = \{s \in \Sigma^+ \mid s \in \Sigma \wedge \forall s' \in \Sigma : \langle s, s' \rangle \notin \tau\}$
  $\cup \{ss'\sigma \mid \langle s, s' \rangle \in \tau \wedge s'\sigma \in X\}$ trace transformer
- $X \sqsubseteq Y \stackrel{\text{def}}{=} (X \cap \Sigma^+) \subseteq (Y \cap \Sigma^+) \wedge (X \cap \Sigma^\omega) \supseteq (Y \cap \Sigma^\omega)$
  computational ordering

(2.2) Program properties

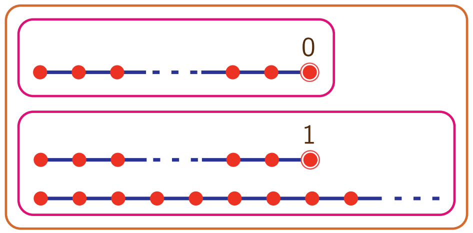
Program properties & Static analysis

- A program property $\mathcal{P} \in \wp(\mathcal{D})$ is a set of semantics for that program (and so a subset of the semantic domain $\mathcal{D}$)
- The strongest program property [13] is $\{\mathcal{S}[\![P]\!]\} \in \wp(\mathcal{D})$
- A Static analysis consists ineffectively approximating the strongest program property :

Compute $\mathcal{P} \in \wp(\mathcal{D})$: $\{\mathcal{S}[\![P]\!]\} \subseteq \mathcal{P}$

[13] also called *collecting semantics*

Example of program property



- Correct implementations: `print 0`, `print 1`, `[print 1|loop]`, ...
- Incorrect implementations: `[print 0|print 1]`
- Note for specialists: neither a safety nor a liveness property.

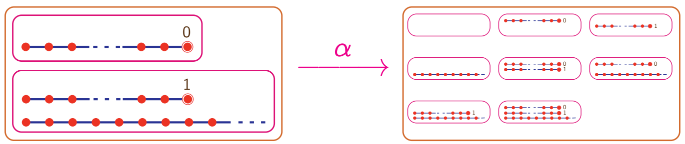
(2.3) Abstraction of program properties

Abstraction

– Replace a concrete property $\mathcal{P} \in \wp(\mathcal{D})$ by an abstract property $\alpha(\mathcal{P})$

– Example:

- $\mathcal{D} = \wp(\Sigma^+ \cup \Sigma^\omega)$ semantic domain
- $\mathcal{P} \in \wp(\mathcal{D})$ concrete property
- $\alpha(\mathcal{P}) \stackrel{\text{def}}{=} \wp(\bigcup P)$ abstract property



Common requirements for abstraction

– [In this talk,] we consider overapproximations:

$$\mathcal{P} \subseteq \alpha(\mathcal{P})$$

- If the abstract property $\alpha(\mathcal{P})$ is true then the concrete property $\mathcal{P}$ is also true
- If the abstract property $\alpha(\mathcal{P})$ is false then the concrete property $\mathcal{P}$ may be true[14] or false!

– All information is lost at once:

$$\alpha(\alpha(\mathcal{P})) = \alpha(\mathcal{P})$$

– The abstraction of more precise properties is more precise:

$$\text{si } \mathcal{P} \subseteq \mathcal{Q} \text{ alors } \alpha(\mathcal{P}) \subseteq \alpha(\mathcal{Q})$$

[14] In this case, this is called a *"false alarm"*.

Galois Connections

– One obtain a Galois connection:

$$\langle \wp(\mathcal{D}), \subseteq \rangle \xleftarrow[\alpha]{1} \xrightarrow{} \langle \wp(\mathcal{D}), \subseteq \rangle$$

↑ Concrete properties ↑ Abstract properties

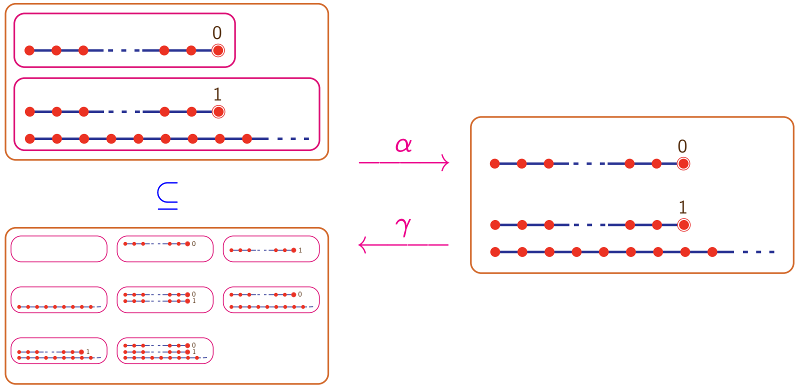
– With an isomorphic mathematical/computer representation

$$\langle \wp(\mathcal{D}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$$

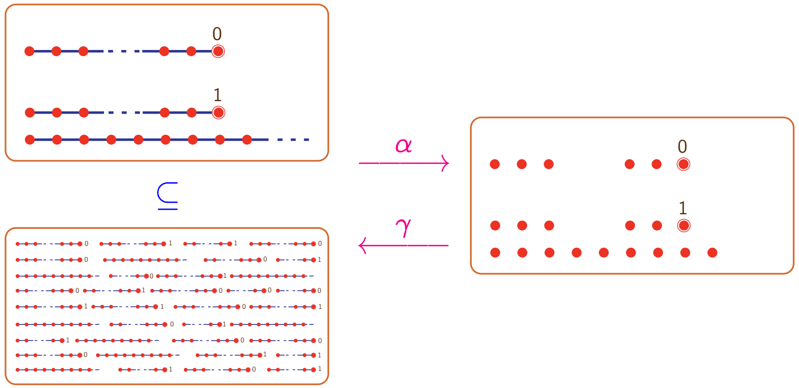
↑ Concrete properties ↑ Abstract domain

$$\forall \mathcal{P} \in \wp(\mathcal{D}) : \forall \mathcal{Q} \in \mathcal{D}^\sharp : \alpha(\mathcal{P}) \sqsubseteq \mathcal{Q} \iff \mathcal{P} \subseteq \gamma(\mathcal{Q})$$

Example 1 of Galois connection



Example 2 of Galois connection



Example 3 of Galois connection

Traces: set of finite or infinite maximal sequences of states for the operational transition semantics

$\xrightarrow{\alpha_1}$ Set of reachable states: set of states appearing at least once along one of these traces (global invariant)

$$\alpha_1(X) = \{\sigma_i \mid \sigma \in X \wedge 0 \leq i < |\sigma|\}$$

$\xrightarrow{\alpha_2}$ Partitionned set of reachable states: project along each control point (local invariant)

$$\alpha_2(\{\langle c_i,\ \rho_i\rangle \mid i \in \Delta\}) = \lambda c \cdot \{\rho_i \mid i \in \Delta \wedge c = c_i\}$$

$\xrightarrow{\alpha_3}$ Partitionned cartesian set of reachable states: project along each program variable (relationships between variables are now lost)

$$\alpha_3(\lambda c \cdot \{\rho_i \mid i \in \Delta_c\}) = \lambda c \cdot \lambda \mathtt{X} \cdot \{\rho_i(\mathtt{X}) \mid i \in \Delta_c\}$$

$\xrightarrow{\alpha_4}$ Partitionned cartesian interval of reachable states: take min and max of the values of the variables [15]

$$\alpha_4(\lambda c \cdot \lambda \mathtt{X} \cdot \{v_i \mid i \in \Delta_{c,\mathtt{X}}\} = \lambda c \cdot \lambda \mathtt{X} \cdot \langle \min\{v_i \mid i \in \Delta_{c,\mathtt{X}}\},\ \max\{v_i \mid i \in \Delta_{c,\mathtt{X}}\}\rangle$$

α_1, α_2, α_3 and α_4, whence $\alpha_4 \circ \alpha_3 \circ \alpha_2 \circ \alpha_1$ are lower-adjoints of Galois connections

15 assuming these values to be totally ordered.

Example 4: Reduced Product of Abstract Domains

To combine abstractions

$\langle \mathcal{D}, \subseteq \rangle \xleftrightarrow[\alpha_1]{\gamma_1} \langle \mathcal{D}^\sharp_1, \sqsubseteq_1 \rangle$ and $\langle \mathcal{D}, \subseteq \rangle \xleftrightarrow[\alpha_2]{\gamma_2} \langle \mathcal{D}^\sharp_2, \sqsubseteq_2 \rangle$

the reduced product is

$$\alpha(X) \stackrel{\text{def}}{=} \sqcap\{\langle x, y\rangle \mid X \subseteq \gamma_1(x) \wedge X \subseteq \gamma_2(y)\}$$

such that $\sqsubseteq \stackrel{\text{def}}{=} \sqsubseteq_1 \times \sqsubseteq_2$ and

$$\langle \mathcal{D}, \subseteq \rangle \xleftrightarrow[\alpha]{\gamma_1 \times \gamma_2} \langle \alpha(\mathcal{D}), \sqsubseteq \rangle$$

Example: $\mathtt{x} \in [1, 9] \wedge \mathtt{x} \bmod 2 = 0$ reduces to $\mathtt{x} \in [2, 8] \wedge \mathtt{x} \bmod 2 = 0$

Abstraction of functions

- Let $\langle \wp(\mathcal{D}), \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$
- How can we abstract an *operator* $F \in \wp(\mathcal{D}) \stackrel{\text{m}}{\longmapsto} \wp(\mathcal{D})$?
- The most precise overapproximation is

$$F^\sharp \in \mathcal{D}^\sharp \stackrel{\text{m}}{\longmapsto} \mathcal{D}^\sharp$$
$$F^\sharp = \alpha \circ F \circ \gamma$$

- This is a Galois connection

$$\langle \wp(\mathcal{D}) \stackrel{\text{m}}{\longmapsto} \wp(\mathcal{D}), \subseteq \rangle \xleftrightarrow[\lambda F \cdot \alpha \circ F \circ \gamma]{\lambda F^\sharp \cdot \gamma \circ F^\sharp \circ \alpha} \langle \mathcal{D}^\sharp \stackrel{\text{m}}{\longmapsto} \mathcal{D}^\sharp, \sqsubseteq \rangle$$

Abstraction of fixpoints

- Let $\langle \wp(\mathcal{D}), \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$
- How can we abstract a *fixpoint property* $\mathsf{lfp}^{\subseteq} F$ where $F \in \wp(\mathcal{D}) \stackrel{\text{m}}{\longmapsto} \wp(\mathcal{D})$?
- Approximate correct abstraction:

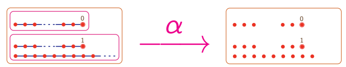
$$\mathsf{lfp}^{\subseteq} F \subseteq \gamma(\mathsf{lfp}^{\sqsubseteq} \alpha \circ F \circ \gamma)$$

- Complete abstraction: if $\alpha \circ F = F^\sharp \circ \alpha$ then

$$F^\sharp = \alpha \circ F \circ \gamma, \text{ and}$$
$$\alpha(\mathsf{lfp}^{\subseteq} F) = \mathsf{lfp}^{\sqsubseteq} F^\sharp$$

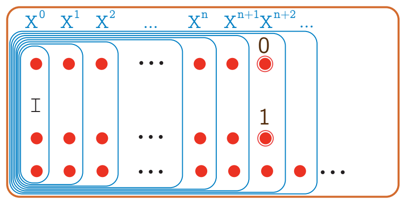
Example 5: reachable states

- Transition system: $\langle \Sigma, \tau \rangle$
- Initial states: $\mathcal{I} \subseteq \Sigma$
- Abstraction:



- Reachable states: $\mathsf{lfp}^{\subseteq} F^\sharp$,

$$F^\sharp(X) = \mathcal{I} \cup \{s' \mid \exists s \in X : \langle s, s' \rangle \in \tau\}$$



Accelerating the convergence of iterative fixpoint computations

– The fixpoint $\mathsf{lfp}^{\sqsubseteq} F^\sharp$, $F^\sharp \in \mathcal{D}^\sharp \xrightarrow{\text{m}} \mathcal{D}^\sharp$ is computed iteratively [16]:

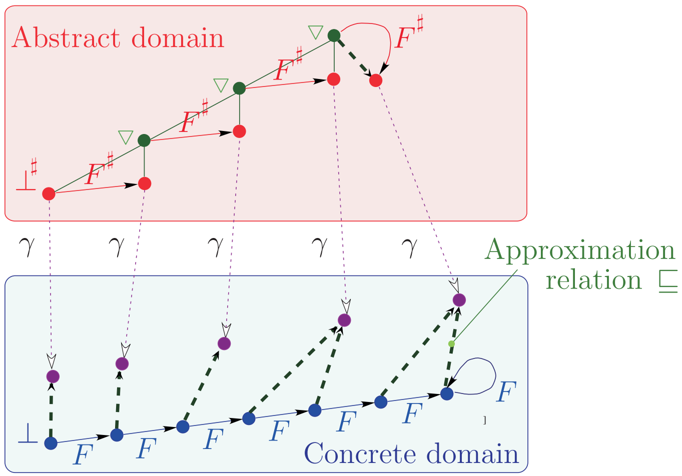
$$X^0 = \bot \qquad X^{n+1} = F^\sharp(X^n) \qquad X^\omega = \bigsqcup_{n \geqslant 0} X^n$$

– For systems of equations $\mathcal{D}^\sharp = \prod_{i=1}^{n} \mathcal{D}^\sharp_n$, one can use asynchronous iterations

– Convergence acceleration techniques have been developped to overapproximated the limit.

[16] $\langle \mathcal{D}^\sharp, \sqsubseteq \rangle$ is a partially ordered set, $F^\sharp$ is monotone, $\bot$ is the infimum, the least upper bound $\sqcup$ must exist for all iterates (in general transfinite).

Convergence acceleration with widening



Abstract-interpretation-based static analysis

1. Define the semantics of the language $\mathcal{S} \in \mathcal{L} \mapsto \mathcal{D}$ and the concrete properties $\wp(\mathcal{D})$;
2. Let $\mathcal{Q} \in \wp(\mathcal{D})$ be a property to be proved for program P: $\mathcal{S}[\![P]\!] \in \mathcal{Q}$
3. Choose an abstraction $\langle \wp(\mathcal{D}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$
4. Use abstract interpretation theory to formally design an abstract semantics $\mathcal{S}^\sharp[\![P]\!] \sqsupseteq \alpha(\{\mathcal{S}[\![P]\!]\})$
5. The static analysis algorithm is the computation of this abstract semantics (whence is correct by construction)

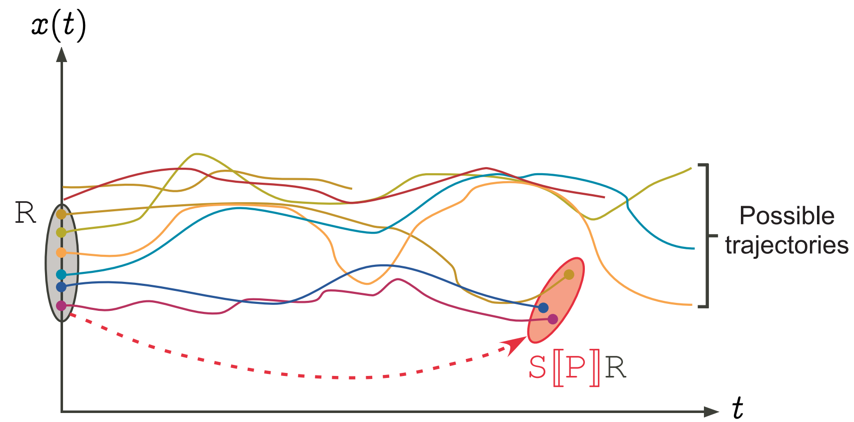
6. The result of the computation is either
   – $\mathcal{S}[\![P]\!] \in \gamma(\mathcal{S}^\sharp[\![P]\!]) \subseteq \mathcal{Q}$ (correctness proof), or
   – $\gamma(\mathcal{S}^\sharp[\![P]\!]) \not\subseteq \mathcal{Q}$ (the property is not satisfied or the approximation is too coarse)
7. The abstraction must be chosen depending on the property $\mathcal{Q}$ to be proved
   – coarse enough to be automatically computable,
   – precise enough to obtain a formal correctness proof: $\gamma(\mathcal{S}^\sharp[\![P]\!]) \subseteq \mathcal{Q}$;

Abstract interpretation (3) a simple example of application

Syntax of programs

X	variables $X \in \mathbb{X}$
T	types $T \in \mathbb{T}$
E	arithmetic expressions $E \in \mathbb{E}$
B	boolean expressions $B \in \mathbb{B}$
$D ::= T\ X\,;$ $\quad \mid\ T\ X\ ;\ D'$	
$C ::= X$ = E; $\quad \mid$ `while` $B\ C'$ $\quad \mid$ `if` $B\ C'$ `else` C'' $\quad \mid$ { $C_1 \ldots C_n$ }, $(n \geq 0)$	commands $C \in \mathbb{C}$
$P ::= D\ C$	program $P \in \mathbb{P}$

Postcondition semantics



Traces to postcondition abstraction

Traces: set of finite or infinite maximal sequences of states for the operational transition semantics

$\xrightarrow{\alpha}$ Strongest liberal postcondition: final states s reachable from a given precondition P

$\alpha(X) = \lambda R \cdot \{s \mid \exists \sigma_0\sigma_1 \ldots \sigma_n \in X : \sigma_0 \in R \wedge s = \sigma_n\}$

We have (Σ: set of states, $\dot{\subseteq}$ pointwise):

$$\langle \wp(\Sigma^\infty),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \wp(\Sigma) \overset{\cup}{\longmapsto} \wp(\Sigma),\ \dot{\subseteq}\rangle$$

States

Values of given type:

$$\mathcal{V}[\![T]\!] \quad : \quad \text{values of type } T \in \mathbb{T}$$

$$\mathcal{V}[\![\texttt{int}]\!] \stackrel{\text{def}}{=} \{z \in \mathbb{Z} \mid \texttt{min\_int} \leq z \leq \texttt{max\_int}\}$$

Program states $\Sigma[\![P]\!]$ [17]:

$$\Sigma[\![D\ C]\!] \stackrel{\text{def}}{=} \Sigma[\![D]\!]$$

$$\Sigma[\![T\ X\,;]\!] \stackrel{\text{def}}{=} \{X\} \mapsto \mathcal{V}[\![T]\!]$$

$$\Sigma[\![T\ X\,;\ D]\!] \stackrel{\text{def}}{=} (\{X\} \mapsto \mathcal{V}[\![T]\!]) \cup \Sigma[\![D]\!]$$

17 States $\rho \in \Sigma[\![P]\!]$ of a program P map program variables X to their values $\rho(X)$

Concrete Reachability Semantics of Programs

$$\mathcal{S}[\![X = E;]\!]R \stackrel{\text{def}}{=} \{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in R \cap \text{dom}(E)\}$$

$$\rho[X \leftarrow v](X) \stackrel{\text{def}}{=} v, \qquad \rho[X \leftarrow v](Y) \stackrel{\text{def}}{=} \rho(Y)$$

$$\mathcal{S}[\![\texttt{if}\ B\ C']\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]R) \cup \mathcal{B}[\![\neg B]\!]R$$

$$\mathcal{B}[\![B]\!]R \stackrel{\text{def}}{=} \{\rho \in R \cap \text{dom}(B) \mid B \text{ holds in } \rho\}$$

$$\mathcal{S}[\![\texttt{if}\ B\ C'\ \texttt{else}\ C'']\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]R) \cup \mathcal{S}[\![C'']\!](\mathcal{B}[\![\neg B]\!]R)$$

$$\mathcal{S}[\![\texttt{while}\ B\ C']\!]R \stackrel{\text{def}}{=} \text{let } \mathcal{W} = \mathsf{lfp}_{\emptyset}^{\subseteq}\ \lambda \mathcal{X} \cdot R \cup \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]\mathcal{X}) \text{ in } (\mathcal{B}[\![\neg B]\!]\mathcal{W})$$

$$\mathcal{S}[\![\{\}]\!]R \stackrel{\text{def}}{=} R$$

$$\mathcal{S}[\![\{C_1 \ldots C_n\}]\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C_n]\!] \circ \ldots \circ \mathcal{S}[\![C_1]\!] \quad n > 0$$

$$\mathcal{S}[\![D\ C]\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C]\!](R)$$

Not computable (undecidability).

Concrete Semantic Domain of Programs

Concrete semantic domain for reachability properties:

$$\mathcal{D}[\![P]\!] \stackrel{\text{def}}{=} \wp(\Sigma[\![P]\!]) \qquad \text{sets of states}$$

i.e. program properties where $\subseteq$ is implication, $\emptyset$ is false, $\cup$ is disjunction.

Abstract Semantic Domain of Programs

$$\langle \mathcal{D}^\sharp[\![P]\!],\ \sqsubseteq,\ \bot,\ \sqcup \rangle$$

such that:

$$\langle \mathcal{D}[\![P]\!],\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp[\![P]\!],\ \sqsubseteq \rangle$$

i.e.

$$\forall X \in \mathcal{D}[\![P]\!], Y \in \mathcal{D}^\sharp[\![P]\!] : \alpha(X) \sqsubseteq Y \iff X \subseteq \gamma(Y)$$

hence $\langle \mathcal{D}^\sharp[\![P]\!],\ \sqsubseteq,\ \bot,\ \sqcup \rangle$ is a complete lattice such that $\bot = \alpha(\emptyset)$ and $\sqcup X = \alpha(\cup\, \gamma(X))$

Abstract Reachability Semantics of Programs

$$
\begin{aligned}
\mathcal{S}^\sharp[\![X = E;]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in \gamma(R) \cap \text{dom}(E)\})\\
\mathcal{S}^\sharp[\![\texttt{if } B\ C']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{B}^\sharp[\![\neg B]\!]R\\
\mathcal{B}^\sharp[\![B]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \cap \text{dom}(B) \mid B \text{ holds in } \rho\})\\
\mathcal{S}^\sharp[\![\texttt{if } B\ C' \texttt{ else } C'']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{S}^\sharp[\![C'']\!](\mathcal{B}^\sharp[\![\neg B]\!]R)\\
\mathcal{S}^\sharp[\![\texttt{while } B\ C']\!]R &\stackrel{\text{def}}{=} \text{let } \mathcal{W} = \mathsf{lfp}_{\perp}^{\sqsubseteq}\, \lambda \mathcal{X} \cdot R \sqcup \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]\mathcal{X})\\
&\qquad \text{in } (\mathcal{B}^\sharp[\![\neg B]\!]\mathcal{W})\\
\mathcal{S}^\sharp[\![\{\}]\!]R &\stackrel{\text{def}}{=} R\\
\mathcal{S}^\sharp[\![\{C_1 \ldots C_n\}]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C_n]\!] \circ \ldots \circ \mathcal{S}^\sharp[\![C_1]\!] \quad n > 0\\
\mathcal{S}^\sharp[\![D\ C]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C]\!](R)
\end{aligned}
$$

Abstract Semantics with Convergence Acceleration [18]

$$
\begin{aligned}
\mathcal{S}^\sharp[\![X = E;]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in \gamma(R) \cap \text{dom}(E)\})\\
\mathcal{S}^\sharp[\![\texttt{if } B\ C']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{B}^\sharp[\![\neg B]\!]R\\
\mathcal{B}^\sharp[\![B]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \cap \text{dom}(B) \mid B \text{ holds in } \rho\})\\
\mathcal{S}^\sharp[\![\texttt{if } B\ C' \texttt{ else } C'']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{S}^\sharp[\![C'']\!](\mathcal{B}^\sharp[\![\neg B]\!]R)\\
\mathcal{S}^\sharp[\![\texttt{while } B\ C']\!]R &\stackrel{\text{def}}{=} \text{let } \mathcal{F}^\sharp = \lambda \mathcal{X} \cdot \text{let } \mathcal{Y} = R \sqcup \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]\mathcal{X})\\
&\qquad \text{in if } \mathcal{Y} \sqsubseteq \mathcal{X} \text{ then } \mathcal{X} \text{ else } \mathcal{X} \mathbin{\triangledown} \mathcal{Y}\\
&\quad \text{and } \mathcal{W} = \mathsf{lfp}_{\perp}^{\sqsubseteq}\, \mathcal{F}^\sharp \qquad \text{in } (\mathcal{B}^\sharp[\![\neg B]\!]\mathcal{W})\\
\mathcal{S}^\sharp[\![\{\}]\!]R &\stackrel{\text{def}}{=} R\\
\mathcal{S}^\sharp[\![\{C_1 \ldots C_n\}]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C_n]\!] \circ \ldots \circ \mathcal{S}^\sharp[\![C_1]\!] \quad n > 0\\
\mathcal{S}^\sharp[\![D\ C]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C]\!](R)
\end{aligned}
$$

[18] Note: $\mathcal{F}^\sharp$ not monotonic!

Abstract interpretation (4) a range of applications

Applications of abstract interpretation

Any reasoning on complex computer systems must consider a correct approximation of its behaviors formalized by Abstract interpretation [5, 20, 21, 34]

- Syntax of programming languages [30]
- Semantics of programming languages [13, 27]
- Proofs of program correctness [11, 12]
- Typing and type inference [18]

– Static analysis of programming languages [3, 7, 15, 16, 22, 26]
  - imperative [2, 4, 6, 9, 19]
  - parallel [10, 8]
  - logial [14]
  - fonctionnal [17]
– Model-checking [23, 28, 31]
– Transformation of programs [29]
– Steganographie [33]
– ...

Abstract interpretation (5) application to the A380 flight control software

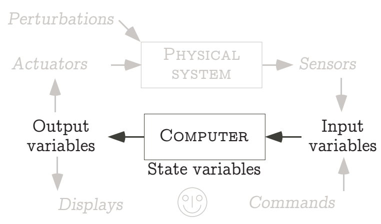
(5.1) The ASTRÉE static analyzer

www.astree.ens.fr [25, 32, 35]

ASTRÉE is a specialized static analyzer

– Embedded real-time synchronous control/command C programs:

```
Declare and initialize state
variables;
loop forever
  read volatile input variables,
  compute output and
  state variables,
  write state variables;
  wait for next clock tick
end loop
```

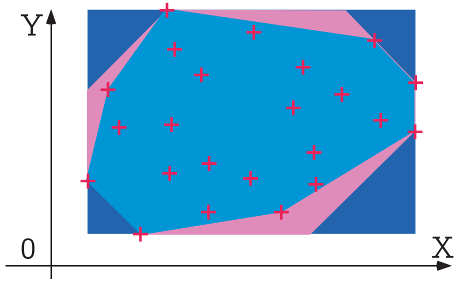


Objective of ASTRÉE

- Prove automatically the absence of runtime errors:
  - No division by 0, `NaN`, out of range array access
  - No signed integer/float overflows
  - Verification of user-defined properties (for example machine dependent properties)
- Requirements:
  - efficiency (must operate on a workstation)
  - precision (few false alarms)
- No alarm $\rightarrow$ full certification

(5.2) Examples of abstractions

General purpose numerical abstract domains



Approximation of a set of points

Intervals: [2]

$$\bigwedge_{i=1}^{n} a_i \leqslant x_i \leqslant b_i$$

Octogans: [37]

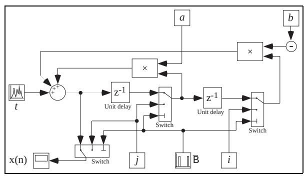
$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \pm x_i \pm y_j \leqslant a_{ij}$$

Polyhedra: [6]

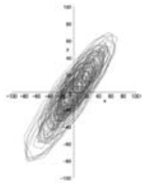
$$\bigwedge_{j=1}^{m} (\sum_{i=1}^{n} a_{ji} x_i) \leqslant b_j$$

Ellipsoid Abstract Domain for Filters

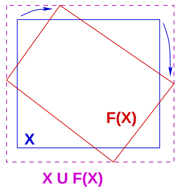
2nd Order Digital Filter:



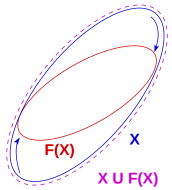
- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

```
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; }
}
```

Filter Example [36]

(5.3) Results

Slow divergences by rounding accumulation

```
X = 1.0;
while (TRUE) { ①
  X = X / 3.0;
  X = X * 3.0;
}
```

– With reals $\mathbb{R}$: $x = 1.0$ at ①
– With floats: rounding errors
– Accumulation of rounding errors: possible cause of divergence

Solution [35] : bound the cumulated rounding error as a function of the number of iterations by arithmetico-geometric progressions:

– Relation $|\mathtt{X}| \leq a \cdot b^n + c$, where a, b, c are constants determined by the analysis, n is the iterate number
– Number of iterates bounded by N : $|\mathtt{X}| \leq a \cdot b^N + c$

Application to the A 340/A 380

– Primary flight control software of the electric flight control system of the Airbus A340 family and the A380


– C program, automatically generated from of high-level specification (à la Simulink/SCADE)
– A340 : 100.000 to 250.000 LOCs
– A380 : 400.000 to 1.000.000 LOCs

A world première

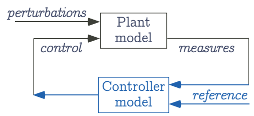
Analysis of 400.000 lines of C code [19]

time	memory	false alarms
13h 52mn	2,2 Gb	0

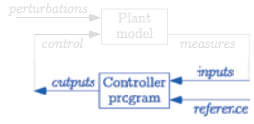
[19] on an AMD Opteron 248, 64 bits, a single processor

Perspectives

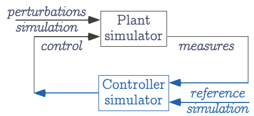
The Current Situation [20]



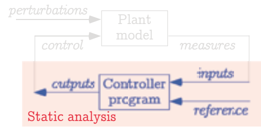
(1) Model design



(3) Implementation



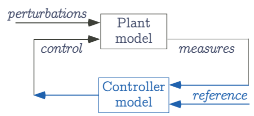
(2) Simulation



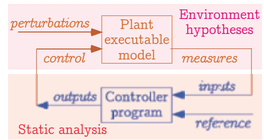
(4) Program analysis

[20] greatly simplified, system dependability is simply ignored!

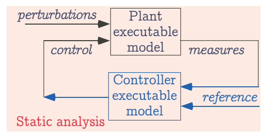
The Project [21]



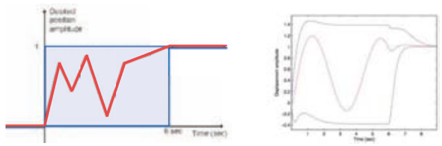
(1) Model design



(3) Program analysis



(2) Model analysis



Example (response analysis)

[21] greatly simplified, system dependability is simply ignored!

The End, thank you for your attention

References on the web: www.di.ens.fr/~cousot.

[2] P. Cousot and R. Cousot. – Static determination of dynamic properties of programs. *In : Proceedings of the Second International Symposium on Programming*, Paris, France, 1976. pp. 106–130. – Dunod, Paris, France.

[3] P. Cousot and R. Cousot. – Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *In : Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, California, 1977. pp. 238–252. – ACM Press, New York, New York, United States.

[4] P. Cousot and R. Cousot. – Static determination of dynamic properties of recursive procedures. *In : IFIP Conference on Formal Description of Programming Concepts, St-Andrews, N.B., Canada*, edited by E. Neuhold. pp. 237–277. – North-Holland Pub. Co., Amsterdam, The Netherlands, 1977.

[5] P. Cousot. – *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, Grenoble, France, 21 March 1978.

[6] P. Cousot and N. Halbwachs. – Automatic discovery of linear restraints among variables of a program. *In : Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Tucson, Arizona, 1978. pp. 84–97. – ACM Press, New York, New York, United States.

[7] P. Cousot and R. Cousot. – Systematic design of program analysis frameworks. *In : Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Antonio, Texas, 1979. pp. 269–282. – ACM Press, New York, New York, United States.

Bibliographic References

[8] P. Cousot and R. Cousot. – Semantic analysis of communicating sequential processes. *In : Seventh International Colloquium on Automata, Languages and Programming*, edited by J. de Bakker and J. van Leeuwen. *Lecture Notes in Computer Science 85*, pp. 119–133. – Springer, Berlin, Germany, July 1980.

[9] P. Cousot. – Semantic Foundations of Program Analysis, invited chapter. *In : Program Flow Analysis: Theory and Applications*, edited by S. Muchnick and N. Jones, Chapter 10, pp. 303–342. – Prentice-Hall, Inc., Englewood Cliffs, New Jersey, United States, 1981.

[10] P. Cousot and R. Cousot. – Invariance Proof Methods and Analysis Techniques For Parallel Programs, invited chapter. *In : Automatic Program Construction Techniques*, edited by A. Biermann, G. Guiho and Y. Kodratoff, Chapter 12, pp. 243–271. – Macmillan, New York, New York, United States, 1984.

[11] P. Cousot and R. Cousot. – 'À la Floyd' induction principles for proving inevitability properties of programs, invited chapter. *In : Algebraic Methods in Semantics*, edited by M. Nivat and J. Reynolds, Chapter 8, pp. 277–312. – Cambridge University Press, Cambridge, United Kingdom, 1985.

[12] P. Cousot. – Methods and Logics for Proving Programs, invited chapter. *In : Formal Models and Semantics*, edited by J. van Leeuwen, Chapter 15, pp. 843–993. – Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990, *Handbook of Theoretical Computer Science*, Vol. B.

[13] P. Cousot and R. Cousot. – Inductive Definitions, Semantics and Abstract Interpretation. *In : Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Albuquerque, New Mexico, United States, 1992. pp. 83–94. – ACM Press, New York, New York, United States.

[14] P. Cousot and R. Cousot. – Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, Vol. 13, nº 2–3, 1992, pp. 103–179. – (The editor of Journal of Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see http://www.di.ens.fr/~cousot.).

[15] P. Cousot and R. Cousot. – Abstract Interpretation Frameworks. *Journal of Logic and Computation*, Vol. 2, nº 4, August 1992, pp. 511–547.

[16] P. Cousot and R. Cousot. – Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation, invited paper. *In : Proceedings of the Fourth International Symposium Programming Language Implementation and Logic Programming, PLILP '92*, edited by M. Bruynooghe and M. Wirsing. *Leuven, Belgium, 26–28 August 1992, Lecture Notes in Computer Science 631*, pp. 269–295. – Springer, Berlin, Germany, 1992.

[17] P. Cousot and R. Cousot. – Higher-Order Abstract Interpretation (and Application to Comportment Analysis Generalizing Strictness, Termination, Projection and PER Analysis of Functional Languages), invited paper. *In : Proceedings of the 1994 International Conference on Computer Languages*, Toulouse, France, 16–19 May 1994. pp. 95–112. – IEEE Computer Society Press, Los Alamitos, California, United States.

[18] P. Cousot. – Types as Abstract Interpretations, invited paper. *In : Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, France, January 1997. pp. 316–331. – ACM Press, New York, New York, United States.

[19] P. Cousot. – The Calculational Design of a Generic Abstract Interpreter, invited chapter. *In : Calculational System Design*, edited by M. Broy and R. Steinbrüggen, pp. 421–505. – NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, Amsterdam, The Netherlands, 1999, Volume 173.

[20] P. Cousot. – Interprétation abstraite (in French). *Technique et science informatique*, Vol. 19, nº 1-2-3, January 2000, pp. 155–164.

[21] P. Cousot. – Abstract Interpretation Based Formal Methods and Future Challenges, invited chapter. *In : « Informatics — 10 Years Back, 10 Years Ahead »*, edited by R. Wilhelm, pp. 138–156. – Springer, Berlin, Germany, 2001, *Lecture Notes in Computer Science* , Vol. 2000.

[22] P. Cousot. – Partial Completeness of Abstract Fixpoint Checking, invited paper. *In : Proceedings of the Fourth International Symposium on Abstraction, Reformulation and Approximation, SARA '2000*, edited by B. Choueiry and T. Walsh, pp. 1–25. – Springer, Berlin, Germany, 26–29 July 2000, *Horseshoe Bay, Texas, United States, Lecture Notes in Artificial Intelligence 1864*.

[23] P. Cousot and R. Cousot. – Temporal Abstract Interpretation. *In : Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, United States, January 2000. pp. 12–25. – ACM Press, New York, New York, United States.

[24] P. Cousot and R. Cousot. – Static Analysis of Embedded Software: Problems and Perspectives, invited paper. *In : Proceedings of the First International Workshop on Embedded Software, EMSOFT '2001*, edited by T. Henzinger and C. Kirsch. *Lecture Notes in Computer Science* , Vol. 2211, pp. 97–113. – Springer, Berlin, Germany, 2001.

[25] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software, invited chapter. *In : The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, edited by T. Mogensen, D. Schmidt and I. Sudborough, pp. 85–108. – Springer, Berlin, Germany, 2002, *Lecture Notes in Computer Science 2566*.

[26] P. Cousot and R. Cousot. – Modular Static Program Analysis, invited paper. *In : Proceedings of the Eleventh International Conference on Compiler Construction, CC '2002*, edited by R. Horspool, Grenoble, France, 6–14 April 2002. pp. 159–178. – Lecture Notes in Computer Science 2304, Springer, Berlin, Germany.

[27] P. Cousot. – Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science*, Vol. 277, nº 1—2, 2002, pp. 47–103.

[28] P. Cousot and R. Cousot. – On Abstraction in Software Verification, invited paper. *In : Proceedings of the Fourteenth International Conference on Computer Aided Verification, CAV '2002*, edited by E. Brinksma and K. Larsen. *Copenhagen, Denmark, Lecture Notes in Computer Science 2404*, pp. 37–56. – Springer, Berlin, Germany, 27–31 July 2002.

[29] P. Cousot and R. Cousot. – Systematic Design of Program Transformation Frameworks by Abstract Interpretation. *In : Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, United States, January 2002. pp. 178–190. – ACM Press, New York, New York, United States.

[30] P. Cousot and R. Cousot. – Parsing as Abstract Interpretation of Grammar Semantics. *Theoretical Computer Science*, Vol. 290, nº 1, January 2003, pp. 531–544.

[31] P. Cousot. – Verification by Abstract Interpretation, invited chapter. *In : Proceedings of the International Symposium on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, edited by N. Dershowitz, pp. 243–268. – Taormina, Italy, Lecture Notes in Computer Science 2772, Springer, Berlin, Germany, 29 June – 4 July 2003.

[32] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – A Static Analyzer for Large Safety-Critical Software. *In : Proceedings of the ACM SIGPLAN '2003 Conference on Programming Language Design and Implementation (PLDI)*, San Diego, California, United States, 7–14 June 2003. pp. 196–207. – ACM Press, New York, New York, United States.

[33] P. Cousot and R. Cousot. – An Abstract Interpretation-Based Framework for Software Watermarking. *In : Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Venice, Italy, 14–16 January 2004. pp. 173–185. – ACM Press, New York, New York, United States.

[34] P. Cousot and R. Cousot. – Basic Concepts of Abstract Interpretation% invitedchapter. *In : Building the Information Society*, edited by P. Jacquart, Chapter 4, pp. 359–366. – Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.

[35] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – The ASTRÉE analyser. *In : Proceedings of the Fourteenth European Symposium on Programming Languages and Systems, ESOP '2005, Edinburg, Scotland*, edited by M. Sagiv, pp. 21–30. – Springer, Berlin, Germany, 2–10 April 2005, *Lecture Notes in Computer Science* , Vol. 3444.

[36] J. Feret. – Static analysis of digital filters. – *ESOP'04*, Barcelona, Spain, *Lecture Notes in Computer Science* , Vol. 2986, pp. 33—-48, Springer, Berlin, Germany, 2004.

[37] A. Miné. – Relational abstract domains for the detection of floating-point run-time errors. *ESOP'04*, Barcelona, Spain, *Lecture Notes in Computer Science* , Vol. 2986, pp. 3—-17, Springer, Berlin, Germany, 2004.