

# Static Program Verification by Abstract Interpretation

Patrick Cousot

École normale supérieure, Paris, France

[cousot@ens.fr](mailto:cousot@ens.fr)   [www.di.ens.fr/~cousot](http://www.di.ens.fr/~cousot)

Georgia Tech

November 14<sup>th</sup>, 2005

## Talk Outline

- Motivation (2 mn) ..... 3
- Abstract interpretation, reminder (12 mn) ..... 6
- Applications of abstract interpretation (2 mn) ..... 26
- A practical application to the ASTRÉE static analyzer (18 mn) 29
- Examples of abstractions in ASTRÉE (12 mn) ..... 45
- Grand challenges in the static analysis of systems (6 mn) .. 61
- Conclusion (2 mn) ..... 70

# Motivation

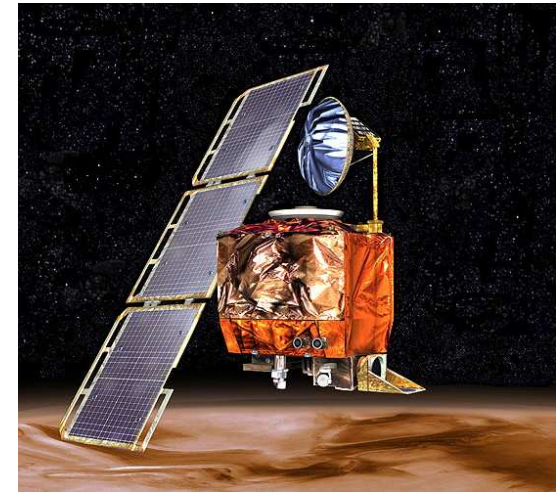
# All Computer Scientists Have Experienced Bugs



Ariane 5.01 failure  
(overflow)



Patriot failure  
(float rounding)



Mars orbiter loss  
(unit error)

It is preferable to verify that mission/safety-critical programs do not go wrong before running them.

# Static Analysis by Abstract Interpretation

**Static analysis:** analyze the program at compile-time to verify a program runtime property

Undecidability  $\longrightarrow$

**Abstract interpretation:** effectively compute an abstraction/  
sound approximation of the program semantics,  
– which is precise enough to imply the desired property, and  
– coarse enough to be efficiently computable.

# Abstract Interpretation, Reminder using a simple example

---

## Reference

---

- [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<sup>th</sup> 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<sup>th</sup> ACM POPL*.

## 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  $B \in \mathbb{B}$

$D ::= T X ;$

$\quad | \quad T X ; D'$

$C ::= X = E ;$

$\quad | \quad \text{while } B \ C'$

$\quad | \quad \text{if } B \ C' \text{ else } C''$

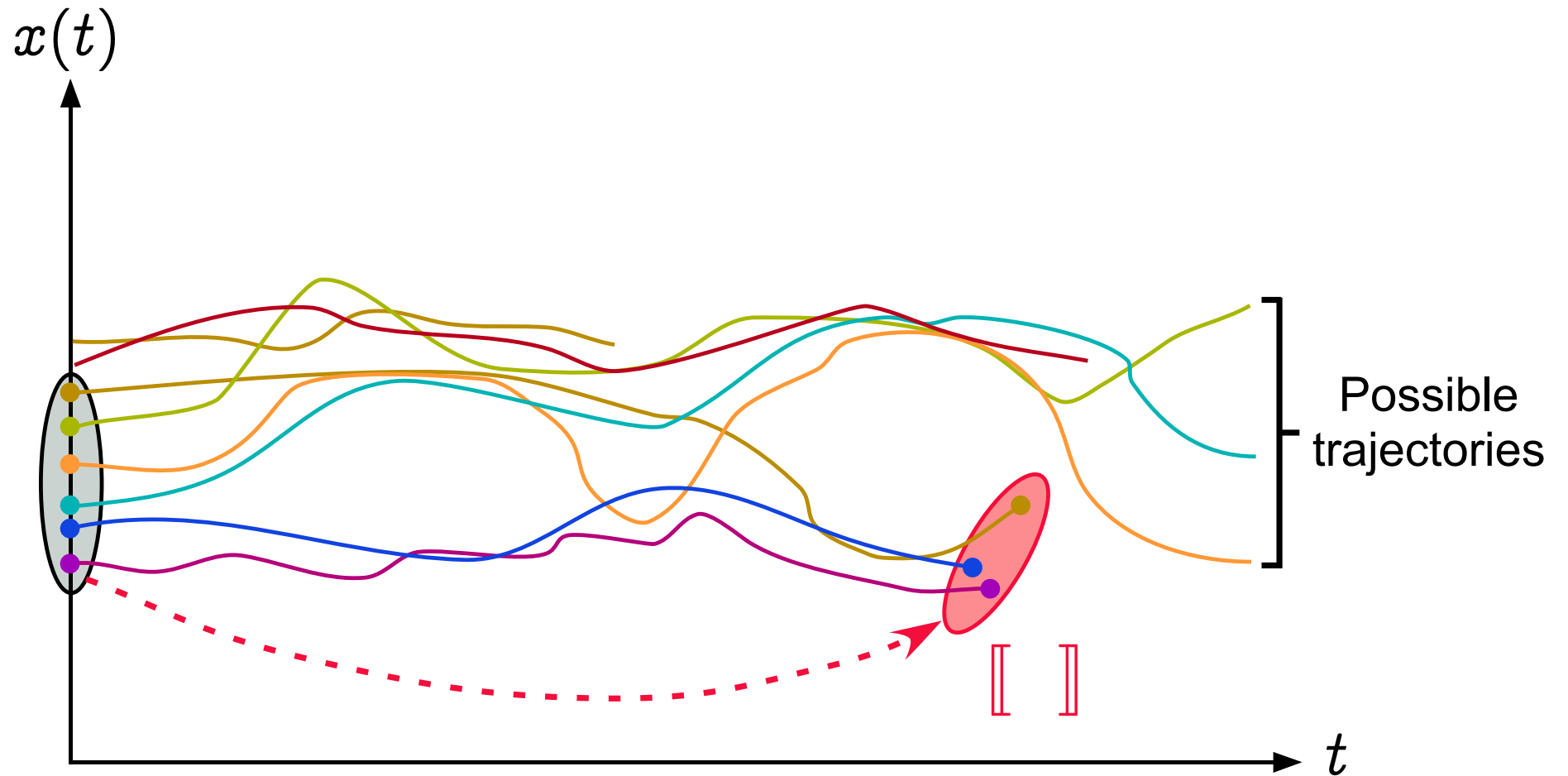
$\quad | \quad \{ C_1 \dots C_n \}, (n \geq 0)$

$P ::= D \ C$

commands  $C \in \mathbb{C}$

program  $P \in \mathbb{P}$

# Postcondition semantics





## States

Values of given type:

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

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

Program states  $\Sigma[[P]]$ <sup>1</sup>:

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

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

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

---

<sup>1</sup> States  $\rho \in \Sigma[[P]]$  of a program  $P$  map program variables  $X$  to their values  $\rho(X)$

## Concrete Semantic Domain of Programs

Concrete semantic domain for reachability properties:

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

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

## Concrete Reachability Semantics of Programs

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

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

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

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

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

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

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

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

$$S[D \text{ } C]R \stackrel{\text{def}}{=} S[C](\Sigma[D]) \quad (\text{uninitialized variables})$$

Not computable (undecidability).

# Abstract Semantic Domain of Programs

$$\langle \mathcal{D}^\# \llbracket P \rrbracket, \sqsubseteq, \perp, \sqcup \rangle$$

such that:

$$\langle \mathcal{D} \llbracket P \rrbracket, \subseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{D}^\# \llbracket P \rrbracket, \sqsubseteq \rangle$$

i.e.

$$\forall X \in \mathcal{D} \llbracket P \rrbracket, Y \in \mathcal{D}^\# \llbracket P \rrbracket : \alpha(X) \sqsubseteq Y \iff X \subseteq \gamma(Y)$$

hence  $\langle \mathcal{D}^\# \llbracket P \rrbracket, \sqsubseteq, \perp, \sqcup \rangle$  is a complete lattice such that  $\perp = \alpha(\emptyset)$  and  $\sqcup X = \alpha(\cup \gamma(X))$

## Example 1 of Abstraction

**Set of 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  $P$

$$\alpha(X) = \lambda P. \{s \mid \exists \sigma_0 \sigma_1 \dots \sigma_n \in X : \sigma_0 \in P \wedge s = \sigma_n\}$$

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

$$\langle \wp(\Sigma^\infty), \subseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \wp(\Sigma) \xrightarrow{\cup} \wp(\Sigma), \dot{\subseteq} \rangle$$

## Example 2 of Abstraction

**Set of traces:** set of finite or infinite maximal sequences of states for the operational transition semantics

$\alpha_0$   
→ **Trace of sets of states:** sequence of set of states appearing at a given time along at least one of these traces

$$\alpha_0(X) = \lambda i. \{\sigma_i \mid \sigma \in X \wedge 0 \leq i < |\sigma|\}$$

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

$$\alpha_1(\Sigma) = \bigcup \{\Sigma_i \mid 0 \leq i < |\Sigma|\}$$

$\alpha_2$   
→ **Partitionned set of reachable states:** project along each control point (local invariant)

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

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

$$\alpha_3(\lambda c. \{\rho_i \mid i \in \Delta_c\}) = \lambda c. \lambda x. \{\rho_i(x) \mid i \in \Delta_c\}$$

$\alpha_4$   
→ Partitionned cartesian interval of reachable states: take min and max of the values of the variables<sup>2</sup>

$$\alpha_4(\lambda c. \lambda x. \{v_i \mid i \in \Delta_{c,x}\}) = \lambda c. \lambda x. \langle \min\{v_i \mid i \in \Delta_{c,x}\}, \max\{v_i \mid i \in \Delta_{c,x}\} \rangle$$

$\alpha_0, \alpha_1, \alpha_2, \alpha_3$  and  $\alpha_4$ , whence  $\alpha_4 \circ \alpha_3 \circ \alpha_2 \circ \alpha_1 \circ \alpha_0$  are lower-adjoints of Galois connections

---

<sup>2</sup> assuming these values to be totally ordered.

### Example 3: Reduced Product of Abstract Domains

To combine abstractions

$$\langle \mathcal{D}, \sqsubseteq \rangle \xrightleftharpoons[\alpha_1]{\gamma_1} \langle \mathcal{D}_1^\#, \sqsubseteq_1 \rangle \text{ and } \langle \mathcal{D}, \sqsubseteq \rangle \xrightleftharpoons[\alpha_2]{\gamma_2} \langle \mathcal{D}_2^\#, \sqsubseteq_2 \rangle$$

the reduced product is

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

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

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

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

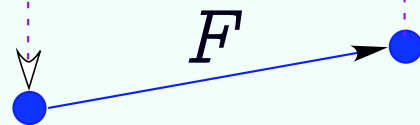


Abstract domain



$\gamma$

$\alpha$



Concrete domain

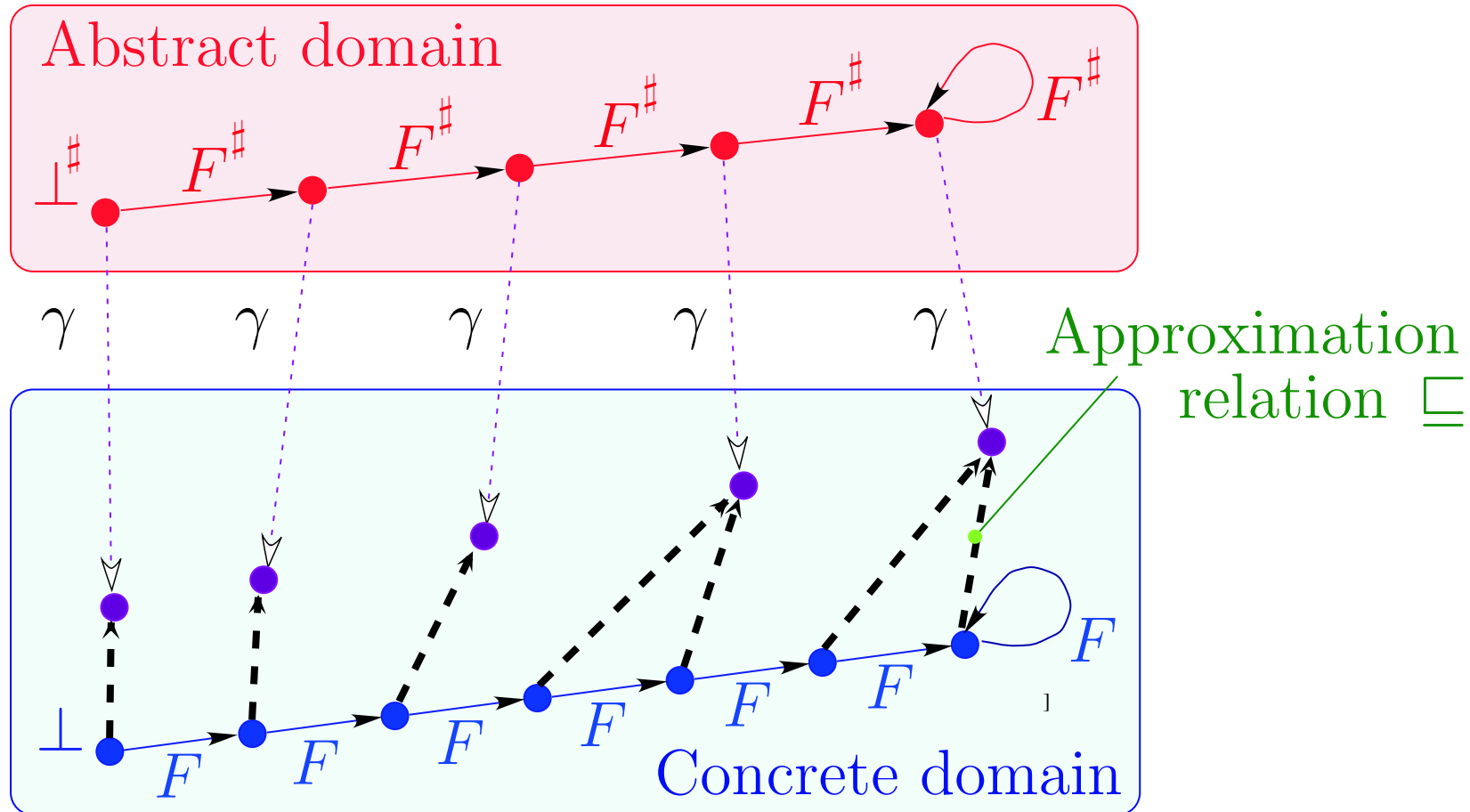
## Function Abstraction

$$F^\# = \alpha \circ F \circ \gamma$$

$$\langle P, \subseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle Q, \sqsubseteq \rangle \Rightarrow$$

$$\langle P \xrightarrow{\text{mon}} P, \dot{\subseteq} \rangle \xrightleftharpoons[\lambda F \cdot \alpha \circ F \circ \gamma]{\lambda F^\# \cdot \gamma \circ F^\# \circ \alpha} \langle Q \xrightarrow{\text{mon}} Q, \dot{\sqsubseteq} \rangle$$

# Approximate Fixpoint Abstraction



$$F \circ \gamma \sqsubseteq \gamma \circ F^\# \Rightarrow \text{lfp } F \sqsubseteq \gamma(\text{lfp } F^\#)$$

## Abstract Reachability Semantics of Programs

$$S^\sharp \llbracket X = E; \rrbracket R \stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E} \llbracket E \rrbracket \rho] \mid \rho \in \gamma(R) \cap \text{dom}(E)\})$$

$$S^\sharp \llbracket \text{if } B \text{ } C' \rrbracket R \stackrel{\text{def}}{=} S^\sharp \llbracket C' \rrbracket (\mathcal{B}^\sharp \llbracket B \rrbracket R) \sqcup \mathcal{B}^\sharp \llbracket \neg B \rrbracket R$$

$$\mathcal{B}^\sharp \llbracket B \rrbracket R \stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \cap \text{dom}(B) \mid B \text{ holds in } \rho\})$$

$$S^\sharp \llbracket \text{if } B \text{ } C' \text{ else } C'' \rrbracket R \stackrel{\text{def}}{=} S^\sharp \llbracket C' \rrbracket (\mathcal{B}^\sharp \llbracket B \rrbracket R) \sqcup S^\sharp \llbracket C'' \rrbracket (\mathcal{B}^\sharp \llbracket \neg B \rrbracket R)$$

$$S^\sharp \llbracket \text{while } B \text{ } C' \rrbracket R \stackrel{\text{def}}{=} \text{let } \mathcal{W} = \text{ifp}_{\perp}^{\sqsubseteq} \lambda \mathcal{X}. R \sqcup S^\sharp \llbracket C' \rrbracket (\mathcal{B}^\sharp \llbracket B \rrbracket \mathcal{X}) \\ \text{in } (\mathcal{B}^\sharp \llbracket \neg B \rrbracket \mathcal{W})$$

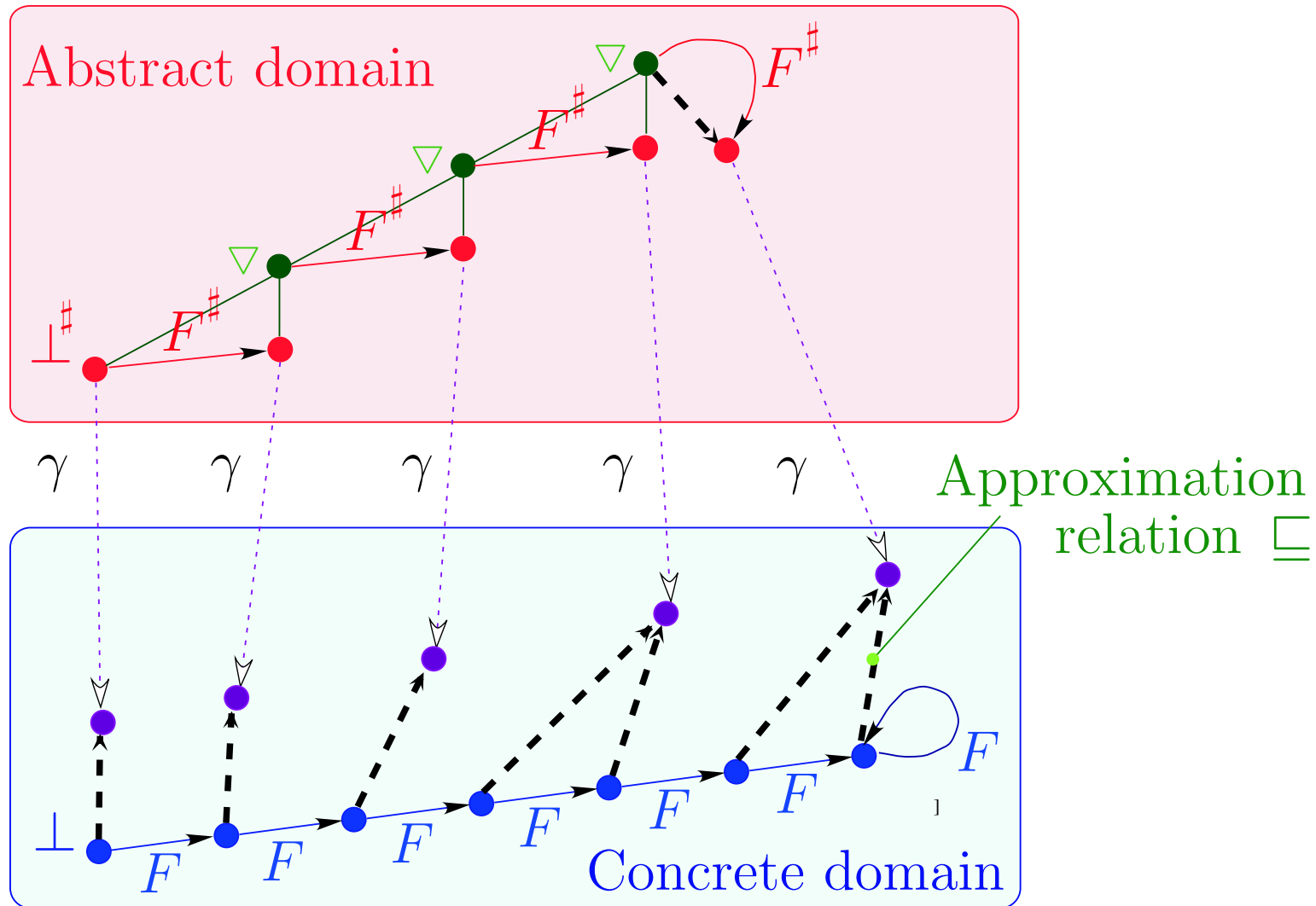
$$S^\sharp \llbracket \{\} \rrbracket R \stackrel{\text{def}}{=} R$$

$$S^\sharp \llbracket \{C_1 \dots C_n\} \rrbracket R \stackrel{\text{def}}{=} S^\sharp \llbracket C_n \rrbracket \circ \dots \circ S^\sharp \llbracket C_1 \rrbracket R \quad n > 0$$

$$S^\sharp \llbracket D \text{ } C \rrbracket R \stackrel{\text{def}}{=} S^\sharp \llbracket C \rrbracket (\top) \quad (\text{uninitialized variables})$$



# Convergence Acceleration with Widening



## Hypotheses on widenings

Given a poset  $\langle L, \sqsubseteq \rangle$ , a widening operator on  $L$  is  $\nabla \in L \times L \mapsto L$  satisfying

- $y \sqsubseteq x \nabla y$
- For all sequences  $x^0, x^1, \dots$  in  $L^\omega$ , the sequence defined by
$$\begin{aligned} y^0 &\stackrel{\text{def}}{=} x^0 \\ y^{n+1} &\stackrel{\text{def}}{=} y^\ell && \text{if } \exists \ell \leq n : x^\ell \sqsubseteq y^\ell \\ &\stackrel{\text{def}}{=} y^n \nabla x^n && \text{otherwise} \end{aligned}$$

is not strictly increasing.

The sequence  $\langle y^k, k \in \mathbb{N} \rangle$  is strictly increasing up to a least  $\ell \in \mathbb{N}$  such that  $x^\ell \sqsubseteq y^\ell$  and the sequence is stationary at  $\ell$  onwards.

## Abstract Semantics with Convergence Acceleration<sup>3</sup>

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

$$S^\sharp[\text{if } B \text{ } C']R \stackrel{\text{def}}{=} S^\sharp[C'](\mathcal{B}^\sharp[B]R) \sqcup \mathcal{B}^\sharp[\neg B]R$$

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

$$S^\sharp[\text{if } B \text{ } C' \text{ else } C'']R \stackrel{\text{def}}{=} S^\sharp[C'](\mathcal{B}^\sharp[B]R) \sqcup S^\sharp[C''](\mathcal{B}^\sharp[\neg B]R)$$

$$S^\sharp[\text{while } B \text{ } C']R \stackrel{\text{def}}{=} \text{let } \mathcal{F}^\sharp = \lambda \mathcal{X}. \text{let } \mathcal{Y} = R \sqcup S^\sharp[C'](\mathcal{B}^\sharp[B]\mathcal{X}) \\ \text{in if } \mathcal{Y} \sqsubseteq \mathcal{X} \text{ then } \mathcal{X} \text{ else } \mathcal{X} \nabla \mathcal{Y}$$

$$\text{and } \mathcal{W} = \text{lfp}_{\perp}^{\sqsubseteq} \mathcal{F}^\sharp \quad \text{in } (\mathcal{B}^\sharp[\neg B]\mathcal{W})$$

$$S^\sharp[\{\}]R \stackrel{\text{def}}{=} R$$

$$S^\sharp[\{C_1 \dots C_n\}]R \stackrel{\text{def}}{=} S^\sharp[C_n] \circ \dots \circ S^\sharp[C_1]R \quad n > 0$$

$$S^\sharp[D \text{ } C]R \stackrel{\text{def}}{=} S^\sharp[C](\top) \quad (\text{uninitialized variables})$$

---

<sup>3</sup> Note:  $\mathcal{F}^\sharp$  not monotonic!

## Why widenings cannot be monotone

– Let  $X$  and  $Y$  be such that  $X \sqsubseteq Y$  (e.g.  $X \sqsubseteq Y = F(X)$  since the iterates for  $F$  with widening  $\nabla$  are increasing)

– Assume that  $\nabla$  is monotone, we have

$$X \nabla Y \sqsubseteq Y \nabla Y$$

– It is desirable that  $(Y \sqsubseteq X) \implies (X \nabla Y = Y)$  (since e.g. if  $Y = F(X) \sqsubseteq X$  then we have converged so there should be no further loss of information)

– In particular for  $X = Y$ , we have

$$Y \nabla Y = Y$$

– It follows, by transitivity, that

$$X \nabla Y \sqsubseteq Y$$

which prevents extrapolations!

## Example of non-monotone widening

- The classical widening on intervals is:

$$\begin{aligned}\perp \nabla X &= X \nabla \perp = X \\ [l_0, u_0] \nabla [l_1, u_1] &= [(l_1 < l_0 ? -\infty : l_0), \\ &\quad (u_1 > u_0 ? +\infty : u_0)]\end{aligned}$$

- Not monotone in its first argument:  $[0, 1] \sqsubseteq [0, 2]$  but  $[0, 1] \nabla [0, 2] = [0, +\infty] \not\sqsubseteq [0, 2] = [0, 2] \nabla [0, 2]$
- Monotone in its second parameter:  $(I' \sqsubseteq I'') \implies (I \nabla I' \sqsubseteq I \nabla I'')$



## The power of the widening/narrowing approach to static program analysis by abstract interpretation

1. For each program there exists a finite lattice which can be used for this program to obtain results equivalent to those obtained using widening/narrowing operators;
2. No lattice satisfying the ascending chain condition will do for all programs;
3. For all programs, infinitely many abstract values are necessary;
4. For a particular program it is not possible to infer the set of needed abstract values by a simple inspection of the program text.

# Applications of Abstract Interpretation

## A few applications of Abstract Interpretation

- **Static Program Analysis** [POPL '77], [POPL '78], [POPL '79]  
including a.o. **Dataflow Analysis** [POPL '79], [POPL '00],  
**Set-based Analysis** [FPCA '95], **Predicate Abstraction**  
[Manna's festschrift '03], ...
- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92],  
[TCS 277(1–2) 2002]
- **Typing & Type Inference** [POPL '97]

## A few applications of Abstract Interpretation (Cont'd)

- (Abstract) Model Checking [POPL '00]
- Program Transformation [POPL '02]
- Software Watermarking [POPL '04]
- Bisimulations [RT-ESOP '04]
- ...

All these techniques involve **sound approximations** that can be formalized by **abstract interpretation**

# A Practical Application of Abstract Interpretation to the *ASTRÉE* Static Analyzer

---

## Reference

- [1] <http://www.astree.ens.fr/> P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, X. Rival

## Programs analysed by ASTRÉE

- **Application Domain:** large safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.
- **C programs:**
  - with
    - basic numeric datatypes, structures and arrays
    - pointers (including on functions),
    - floating point computations
    - tests, loops and function calls
    - limited branching (forward goto, break, continue)

– without

- union
- dynamic memory allocation
- recursive function calls
- backward branching
- conflicting side effects
- C libraries, system calls (parallelism)

## Concrete Operational Semantics

- International **norm of C** (ISO/IEC 9899:1999)
- *restricted by* **implementation-specific behaviors** depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
- *restricted by* user-defined **programming guidelines** (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
- *restricted by* program specific **user requirements** (e.g. assert, execution stops on first runtime error<sup>4</sup>)

---

<sup>4</sup> semantics of C unclear after an error, equivalent if no alarm



## Abstract Semantics

- Reachable states for the concrete trace operational semantics
- Volatile environment is specified by a *trusted* configuration file.

### Requirements:

- Soundness: absolutely essential
- Precision: few or no false alarm<sup>5</sup> (full certification)
- Efficiency: rapid analyses and fixes during development

---

<sup>5</sup> Potential runtime error signaled by the analyzer due to overapproximation but impossible in any actual program run.

## Implicit Specification: Absence of Runtime Errors

- No violation of the **norm of C** (e.g. array index out of bounds, division by zero)
- **No** implementation-specific **undefined behaviors** (e.g. maximum short integer is 32767, NaN)
- No violation of the **programming guidelines** (e.g. static variables cannot be assumed to be initialized to 0)
- No violation of the **programmer assertions** (must all be statically verified).

## Example application

- Primary flight control software of the Airbus A340 family/A380 fly-by-wire system



- C program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)
- A340 family: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays
- A380:  $\times 3/7$



## The Class of Considered Periodic Synchronous Programs

```
declare volatile input, state and output variables;  
initialize state and output variables;  
loop forever  
  - read volatile input variables,  
  - compute output and state variables,  
  - write to output variables;  
  __ ASTREE_wait_for_clock ();  
end loop
```

Task scheduling is static:

- Requirements: the only interrupts are clock ticks;
- Execution time of loop body less than a clock tick  
[EMSOFT '01].

## Challenging aspects

- Size:  $> 100$  kLOC,  $> 10\,000$  variables
- Floating point computations  
including interconnected networks of filters, non linear control with feedback, interpolations...
- Interdependencies among variables:
  - Stability of computations should be established
  - Complex relations should be inferred among numerical and boolean data
  - Very long data paths from input to outputs

## Characteristics of the **ASTRÉE** Analyzer

**Static:** compile time analysis ( $\neq$  run time analysis **Rational Purify**, **Parasoft Insure++**)

**Program Analyzer:** analyzes programs not micromodels of programs ( $\neq$  **PROMELA** in **SPIN** or **Alloy** in the **Alloy Analyzer**)

**Automatic:** no end-user intervention needed ( $\neq$  **ESC Java**, **ESC Java 2**)

**Sound:** covers the whole state space ( $\neq$  **MAGIC**, **CBMC**) so never omit potential errors ( $\neq$  **UNO**, **CMC** from **coverity.com**) or sort most probable ones ( $\neq$  **Splint**)

## Characteristics of the ASTRÉE Analyzer (Cont'd)

- Multiabstraction:** uses many numerical/symbolic abstract domains ( $\neq$  symbolic constraints in **Bane** or the canonical abstraction of **TVLA**)
- Infinitary:** all abstractions use infinite abstract domains with widening/narrowing ( $\neq$  model checking based analyzers such as **VeriSoft**, **Bandera**, **Java PathFinder**)
- Efficient:** always terminate ( $\neq$  counterexample-driven automatic abstraction refinement **BLAST**, **SLAM**)

## Characteristics of the **ASTRÉE** Analyzer (Cont'd)

**Specializable:** can easily incorporate new abstractions (and reduction with already existing abstract domains) ( $\neq$  general-purpose analyzers **PolySpace Verifier**)

**Domain-Aware:** knows about control/command (e.g. digital filters) (as opposed to specialization to a mere programming style in **C Global Surveyor**)

**Parametric:** the precision/cost can be tailored to user needs by options and directives in the code



## Characteristics of the **ASTRÉE** Analyzer (Cont'd)

**Automatic Parametrization:** the generation of parametric directives in the code can be programmed (to be specialized for a specific application domain)

**Modular:** an analyzer instance is built by selection of **O-CAML** modules from a collection each implementing an abstract domain

**Precise:** very few or no false alarm when adapted to an application domain  $\longrightarrow$  **it is a VERIFIER!**

# Example of Analysis Session

The screenshot displays the Visualizer application window, which is used for analyzing C programs. The interface is divided into several panes:

- Top Menu Bar:** Contains icons for Quit, Clods, Trees, Octagons, Filters, Geom. dev., Symbolics, and Help.
- Search string:** A text field for searching through the code.
- Program points:** A dropdown menu currently set to 'Current'.
- Context Pane (Left):** Shows a tree view of the program's execution context, including calls to `main` and `filtre2` at various memory addresses.
- Source Pane (Center):** Displays the source code of `filtre2.c`. The code includes a `typedef enum` for `BOOLEAN`, a `float` array `P`, and a `void filtre2()` function that performs a complex calculation. The `main()` function calls `filtre2()` and updates `INIT`.
- Variables and Invariant Pane (Bottom):**
  - Location:** `filtre2.c:12:6[call#main@20:loop@23]>=4:call#filtre2@25]`
  - Variables:** `P (1)`
  - Invariant:** `<interval: P in [-1252.84, 1252.84] inter [-3362.7, 3491.96]+clock inter [-3362.7, 3491.96]-clock>`
  - Filter d'ordre 2:** A list of variables and their values, such as `Var_entree 1 : E[0]`, `Var_entree 2 : E[1]`, `Var_sortie : P`, and various coefficients like `coef_e1 : 0.4677826`.
  - Octagon:** A section showing the octagon lattice for the variable `P`, with a range from `-5430.9504421651563462` to `39396.917979075267795`.
- Info Pane (Bottom):** Contains metadata about the analysis session, including the date and time (2004/3/16 20:41:58), the command line used to launch the analyzer, and the user (P. Cousot).

## Benchmarks (Airbus A340 Primary Flight Control Software)

- 132,000 lines, 75,000 LOCs after preprocessing
- Comparative results (commercial software):
  - 4,200 (false?) alarms,
  - 3.5 days;
- Our results:
  - 0 alarms,
  - 40mn on 2.8 GHz PC,
  - 300 Megabytes
  - A world première!



## (Airbus A380 Primary Flight Control Software)

- 350,000 lines
- 0 alarms (Nov. 2004),  
7h<sup>6</sup> on 2.8 GHz PC,  
1 Gigabyte  
→ A world grand première!
- November 2005 → 1,000,000 lines

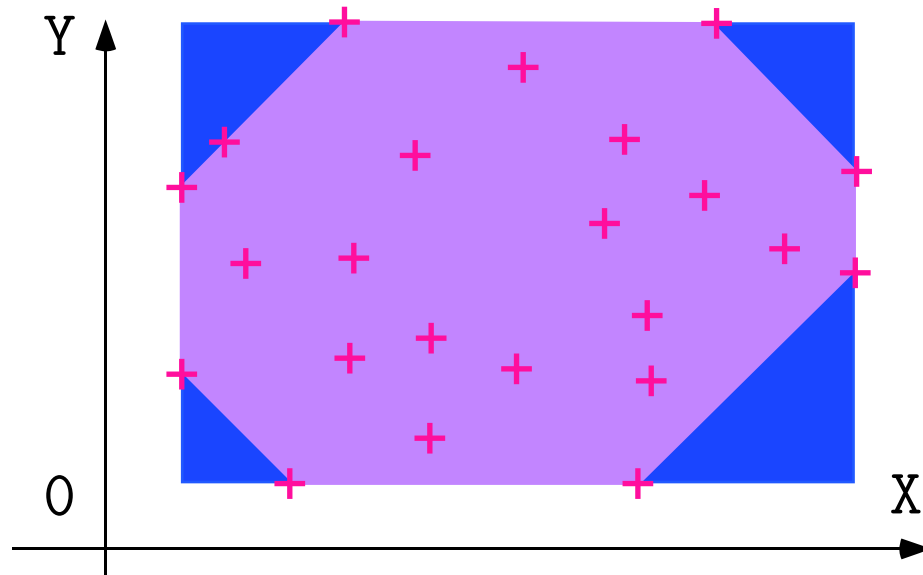
---

<sup>6</sup> We are still in a phase where we favour precision rather than computation costs, and this should go down. For example, the A340 analysis went up to 5 h, before being reduced by requiring less precision while still getting no false alarm.



# Examples of Abstractions

# General-Purpose Abstract Domains: Intervals and Octagons



Intervals:

$$\begin{cases} 1 \leq x \leq 9 \\ 1 \leq y \leq 20 \end{cases}$$

Octagons [11]:

$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 20 \\ x - y \leq 04 \end{cases}$$

**Difficulties:** many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [POPL '77, 11, 12]

# Floating-Point Computations

```
/* 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
```

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

# Floating-Point Computations

```
/* 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
```

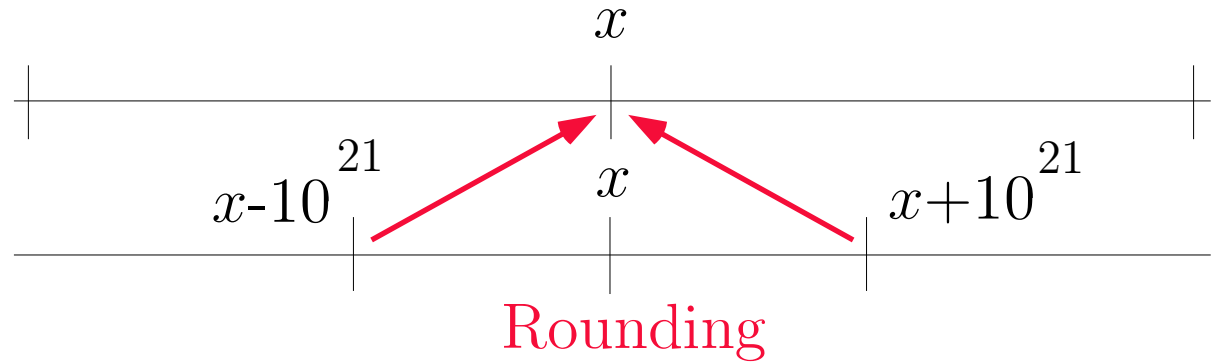
$$(x + a) - (x - a) \neq 2a$$



# Explanation of the huge rounding error

(1) Floats

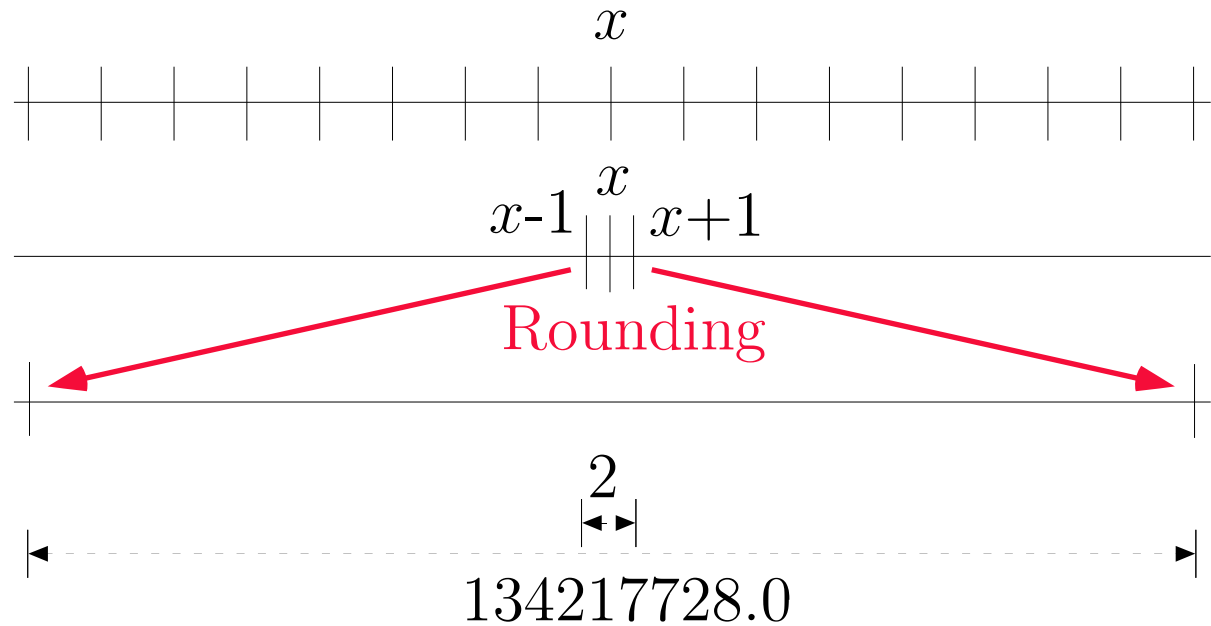
Reals



(2) Doubles

Reals

Floats



## Floating-point linearization [12, 13]

- Approximate arbitrary expressions in the form

$$[a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$$

- Example:

$Z = X - (0.25 * X)$  is linearized as

$$Z = ([0.749 \dots, 0.750 \dots] \times x) + (2.35 \dots 10^{-38} \times [-1, 1])$$

- Allows **simplification** even in the interval domain

if  $X \in [-1, 1]$ , we get  $|Z| \leq 0.750 \dots$  instead of  $|Z| \leq 1.25 \dots$

- Allows using a **relational abstract domain** (octagons)

- Example of good compromise between cost and precision

## Symbolic abstract domain [12, 13]

- Interval analysis: if  $x \in [a, b]$  and  $y \in [c, d]$  then  $x - y \in [a - d, b - c]$  so if  $x \in [0, 100]$  then  $x - x \in [-100, 100]$ !!!
- The symbolic abstract domain propagates the symbolic values of variables and performs simplifications;
- Must maintain the maximal possible rounding error for float computations (overestimated with intervals);

```
% cat -n x-x.c
```

```
1 void main () { int X, Y;  
2     __ASTREE_known_fact(((0 <= X) && (X <= 100)));  
3     Y = (X - X);  
4     __ASTREE_log_vars((Y));  
5 }
```

```
astree -exec-fn main -no-relational x-x.c
```

```
Call main@x-x.c:1:5-x-x.c:1:9:
```

```
<interval: Y in [-100, 100]>
```

```
astree -exec-fn main x-x.c
```

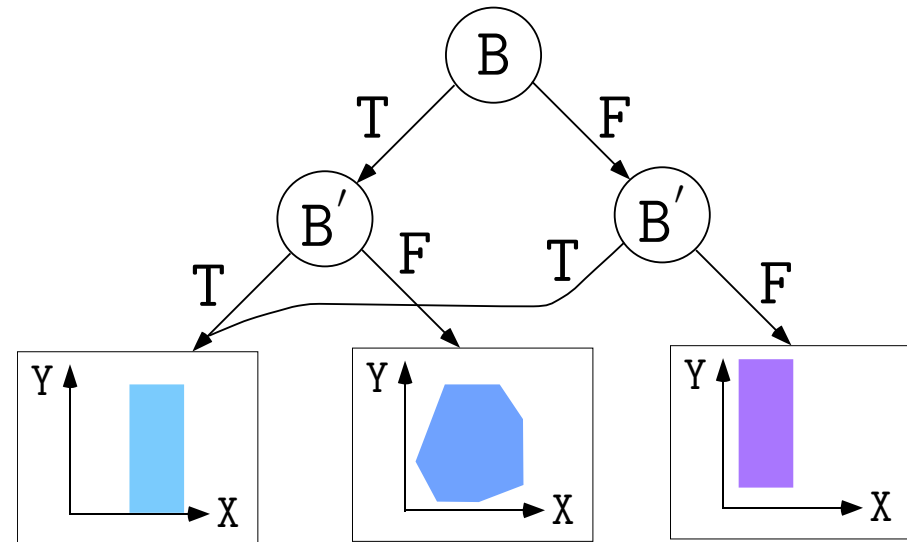
```
Call main@x-x.c:1:5-x-x.c:1:9:
```

```
<interval: Y in {0}> <symbolic: Y = (X -i X)>
```

# Boolean Relations for Boolean Control

## – Code Sample:

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



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

# Control Partitionning for Case Analysis

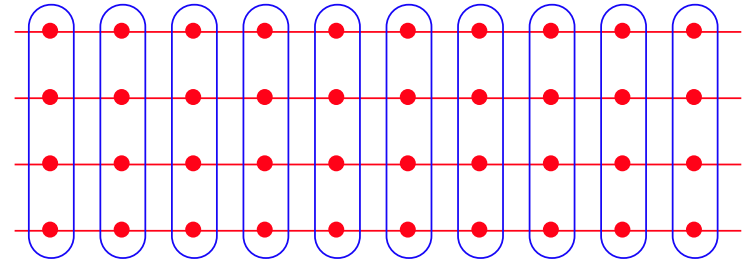
## —Code Sample:

```
/* trace_partitionning.c */
void main() {
  float t[5] = {-10.0, -10.0, 0.0, 10.0, 10.0};
  float c[4] = {0.0, 2.0, 2.0, 0.0};
  float d[4] = {-20.0, -20.0, 0.0, 20.0};
  float x, r;
  int i = 0;

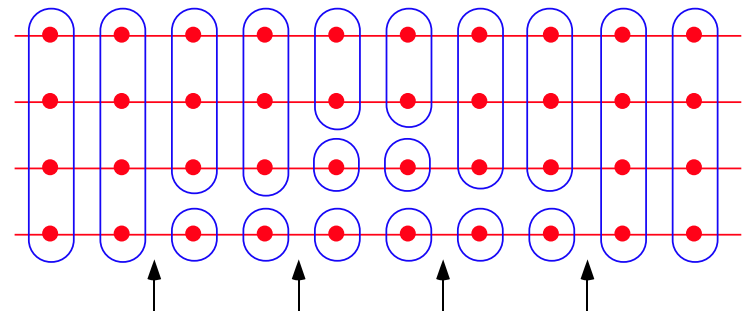
  ... found invariant  $-100 \leq x \leq 100$  ...

  while ((i < 3) && (x >= t[i+1])) {
    i = i + 1;
  }
  r = (x - t[i]) * c[i] + d[i];
}
```

## Control point partitionning:

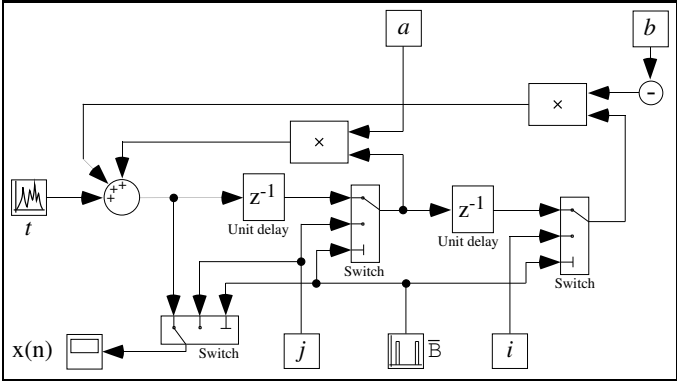


## Trace partitionning:



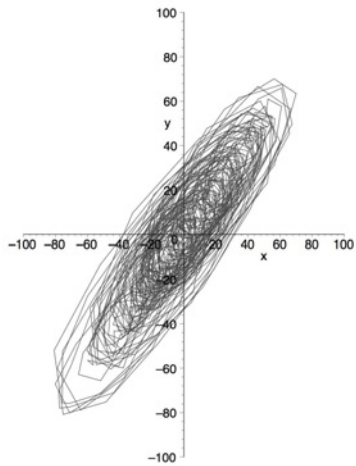
Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

## 2<sup>d</sup> Order Digital Filter:

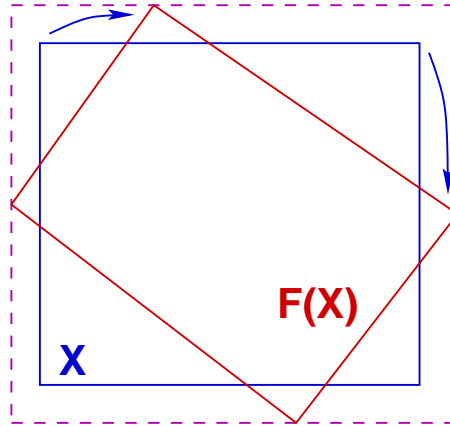


# Ellipsoid Abstract Domain for Filters

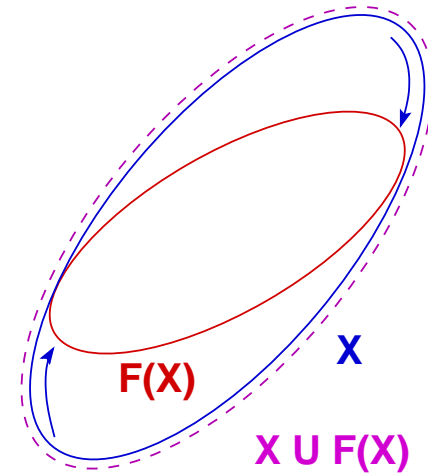
- Computes  $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is **bounded**, which must be proved in the abstract.
- There is **no stable interval or octagon**.
- The simplest stable surface is an **ellipsoid**.



## execution trace



$X \cup F(X)$   
unstable interval



stable ellipsoid

## Filter Example [8]

```
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; }
}
```



## Arithmetic-geometric progressions<sup>7</sup> [9]

– Abstract domain:  $(\mathbb{R}^+)^5$

– Concretization:

$$\gamma \in (\mathbb{R}^+)^5 \longmapsto \wp(\mathbb{N} \mapsto \mathbb{R})$$

$$\gamma(M, a, b, a', b') = \{f \mid \forall k \in \mathbb{N} : |f(k)| \leq (\lambda x . ax + b \circ (\lambda x . a'x + b')^k)(M)\}$$

i.e. any function bounded by the arithmetic-geometric progression.

---

<sup>7</sup> here in  $\mathbb{R}$



# Arithmetic-Geometric Progressions (Example 1)

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
    R = 0;
    while (TRUE) {
        __ASTREE_log_vars((R));
        if (I) { R = R + 1; }
        else { R = 0; }
        T = (R >= 100);
        __ASTREE_wait_for_clock(());
    }
}
```

← potential overflow!

```
% cat count.config
__ASTREE_volatile_input((I [0,1]));
__ASTREE_max_clock((3600000));
% astree -exec-fn main -config-sem count.config count.c|grep '|R|'
|R| <= 0. + clock *1. <= 3600001.
```



## Arithmetic-geometric progressions (Example 2)

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

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

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock();
  }}

% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));

|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1
+ 1.19209290217e-07)^clock
- 5.87747175411e-39 /
1.19209290217e-07 <=
23.0393526881
```

## (Automatic) Parameterization

- All abstract domains of ASTRÉE are **parameterized**, e.g.
  - variable packing for octagones and decision trees,
  - partition/merge program points,
  - loop unrollings,
  - thresholds in widenings, ...;
- End-users can either **parameterize by hand** (analyzer options, directives in the code), or
- choose the **automatic parameterization** (default options, directives for pattern-matched predefined program schemata).

## The main loop invariant for the A340

A textual file over 4.5 Mb with

- 6,900 boolean interval assertions ( $x \in [0; 1]$ )
- 9,600 interval assertions ( $x \in [a; b]$ )
- 25,400 clock assertions ( $x + \text{clk} \in [a; b] \wedge x - \text{clk} \in [a; b]$ )
- 19,100 additive octagonal assertions ( $a \leq x + y \leq b$ )
- 19,200 subtractive octagonal assertions ( $a \leq x - y \leq b$ )
- 100 decision trees
- 60 ellipse invariants, etc ...

involving over 16,000 floating point constants (only 550 appearing in the program text)  $\times$  75,000 LOCs.

## Possible origins of imprecision and how to fix it

In case of false alarm, the imprecision can come from:

- **Abstract transformers** (not best possible)  $\longrightarrow$  improve algorithm;
- **Automatized parametrization** (e.g. variable packing)  $\longrightarrow$  improve pattern-matched program schemata;
- **Iteration strategy** for fixpoints  $\longrightarrow$  fix widening <sup>8</sup>;
- **Inexpressivity** i.e. indispensable local inductive invariant are inexpressible in the abstract  $\longrightarrow$  add a **new abstract domain** to the reduced product (e.g. filters).

---

<sup>8</sup> This can be very hard since at the limit only a precise infinite iteration might be able to compute the proper abstract invariant. In that case, it might be better to design a more refined abstract domain.

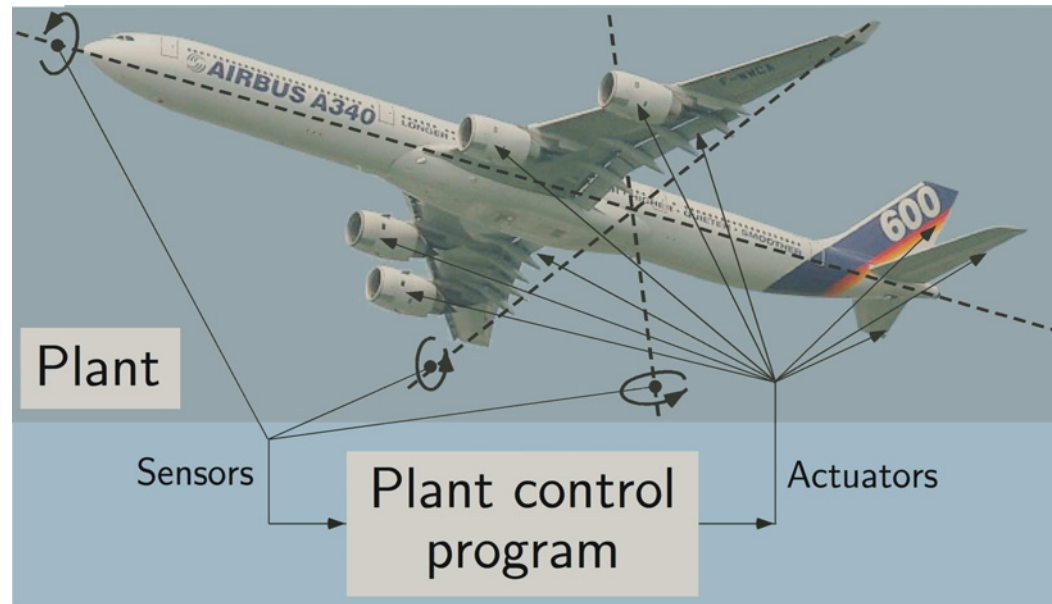
# Grand challenges in the static analysis of systems

---

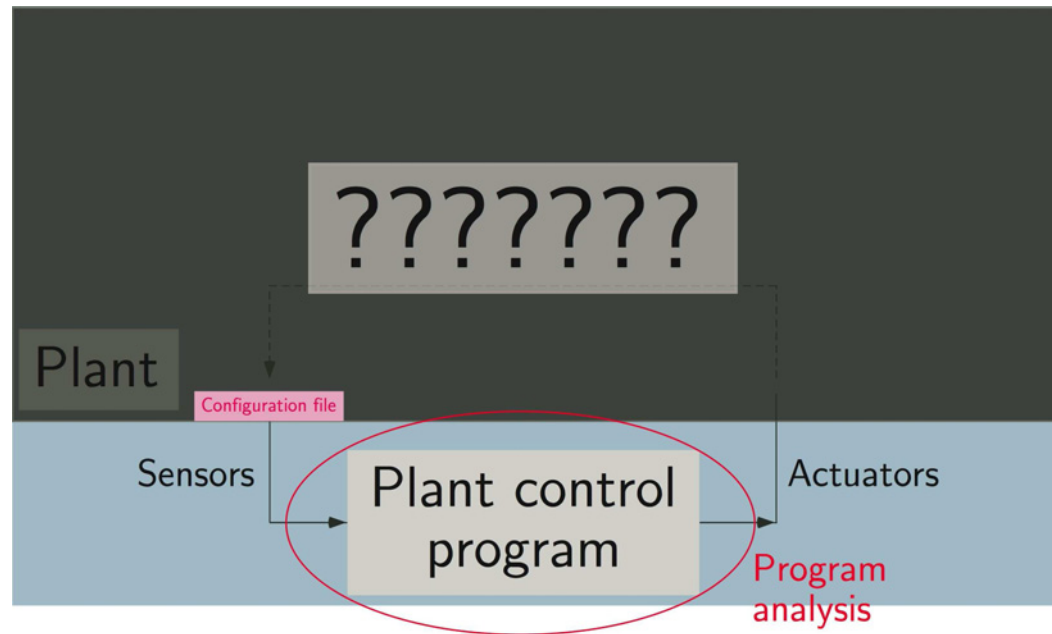
## Reference

- [2] P. Cousot. Integrating Physical Systems in the Static Analysis of Embedded Control Software APLAS'05, Tsukuba, Japan, November 4, 2004. LNCS, Springer.

# Computer controlled systems



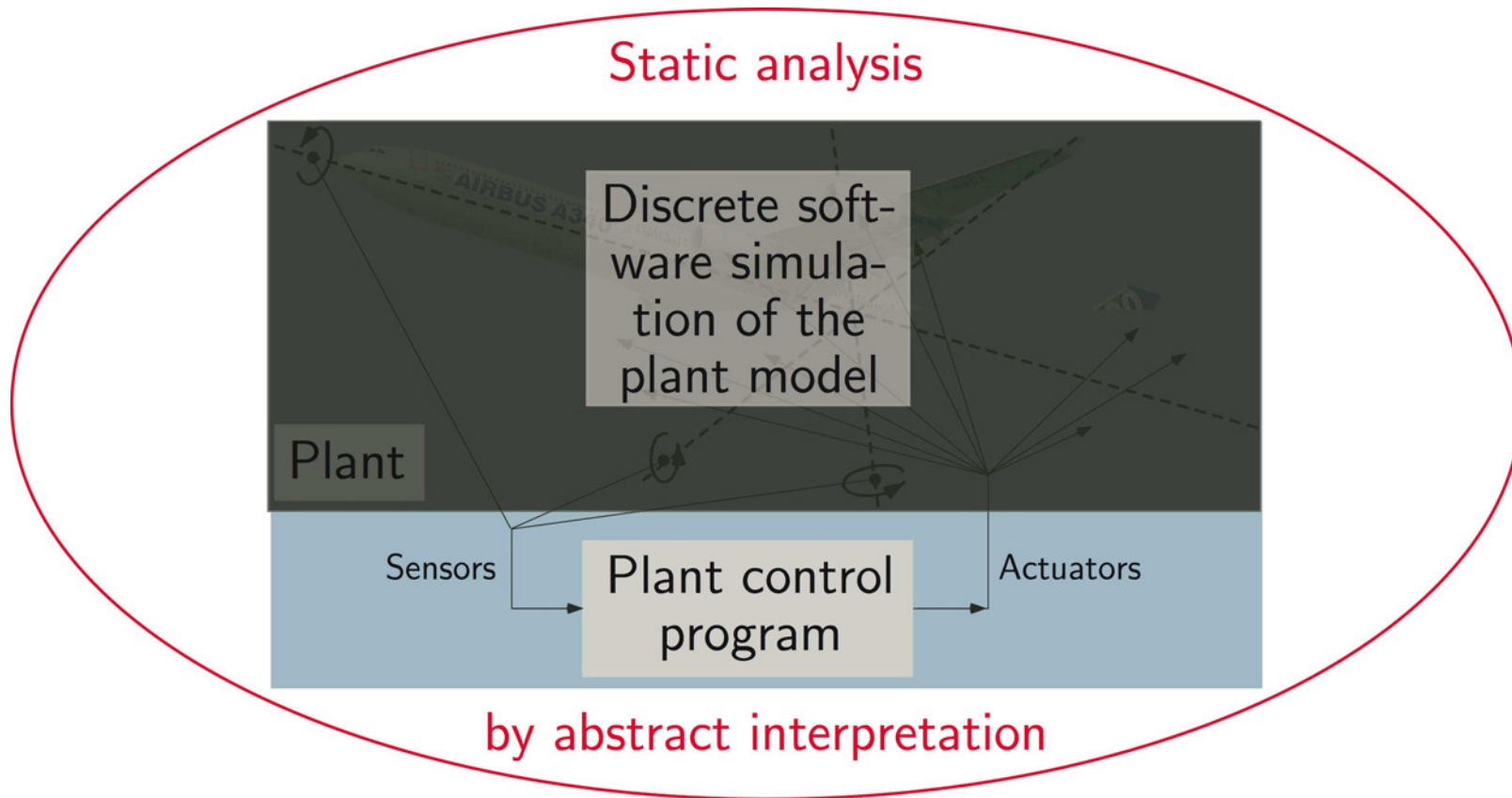
# Software analysis & verification with ASTRÉE



Abstractions: program  $\rightarrow$  precise, system  $\rightarrow$  coarse



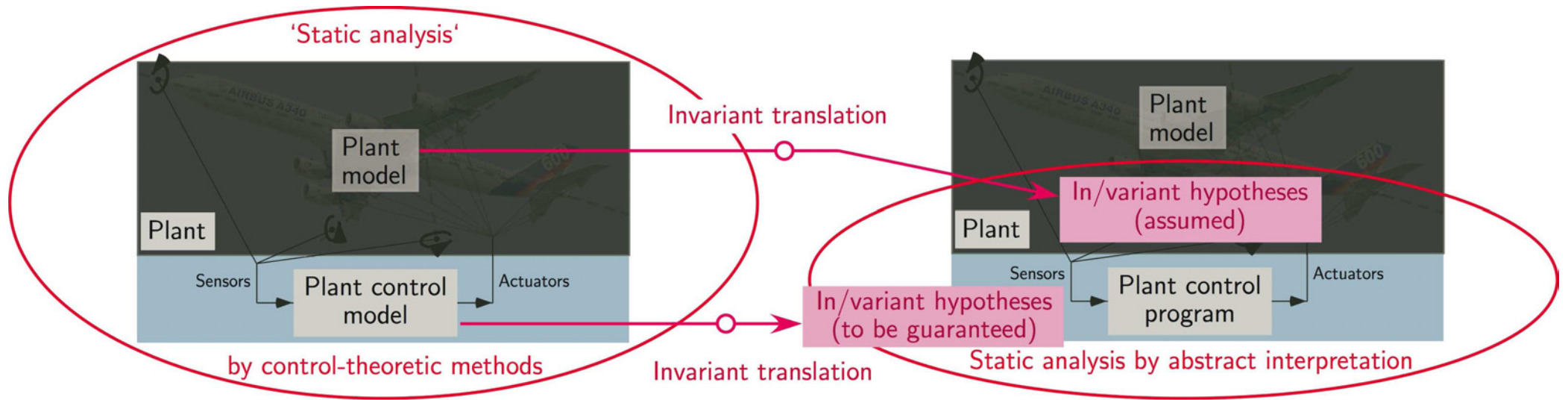
# System analysis & verification, Avenue 1



Abstractions: program  $\rightarrow$  precise, system  $\rightarrow$  precise

- **Exhaustive** (contrary to current simulations)
- The **plant model discretization errors** are similar to those of simulation methods (but for the use of the *actual* control program instead of a model!)
- In general, **polyhedral abstractions** are unstable or of very high complexity
- New abstractions have to be studied (e.g. **ellipsoidal abstractions**)!

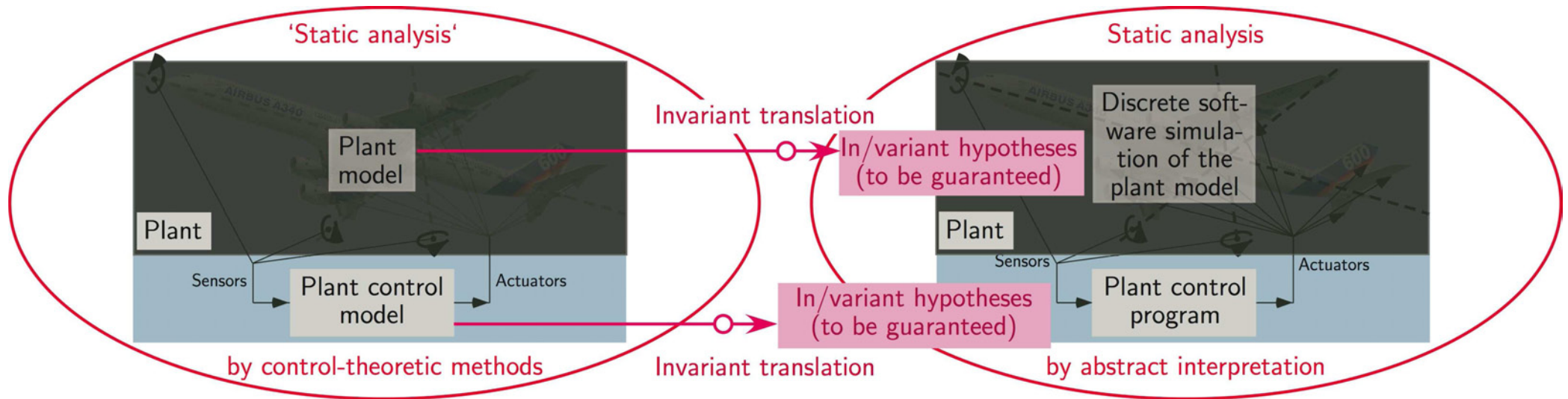
## System analysis & verification, Avenue 2



Abstractions: program  $\rightarrow$  precise, system  $\rightarrow$  precise

- The **control-theoretic ‘static analysis’** is easier on the plant/controller model using continuous optimization methods
- The **in/variant hypotheses** on the controlled plant are assumed to be true in the analysis of the plant control program
- It is now sufficient to perform the **analysis analysis control program** under these in/variant hypotheses
- The results can then be checked on the **whole system (plant simulation + control program)**

# System analysis & verification, Avenue 3



Abstractions: program  $\rightarrow$  precise, system  $\rightarrow$  precise

- The **translated in/variants** can be checked for the plant simulator/control program (easier than in/variant discovery)
- Should **scale up** (since these complex in/variants are relevant to a small part of the control program only<sup>9</sup>)

---

<sup>9</sup> e.g. the plant model assumes perfect sensors/actuators/computers whereas the control program must be made dependable by using redundant failing sensors/actuators/computers

# Conclusion

# Conclusions

1. On **soundness** and **completeness**:
  - **Software checking** (e.g. [abstract] testing): unsound
  - **Software static analysis** (for a language): sound but unprecise
  - **Software verification** (for a well-defined family of programs): **theoretically possible** [SARA '00], **practically feasible** [PLDI '03]

---

## Reference

- [SARA '00] P. Cousot. Partial Completeness of Abstract Fixpoint Checking, invited paper. In *4<sup>th</sup> Int. Symp. SARA '2000*, LNAI 1864, Springer, pp. 1–25, 2000.
- [PLDI '03] 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. PLDI'03, San Diego, June 7–14, ACM Press, 2003.



## Conclusions (cont'd)

### 2. On specifications for static verification:

- **Implicit**: e.g. from a language semantics (e.g. RTE) → extremely easy for engineers
- **Explicit**:
  - By a **logic** → very hard for engineers
  - By a **model** → easy for engineers / hard for static analysis
  - By a **program** automatically generated from a model → easy for engineers / easy for static analysis

# THE END, THANK YOU

More references at URL [www.di.ens.fr/~cousot](http://www.di.ens.fr/~cousot)  
[www.astree.ens.fr](http://www.astree.ens.fr).

# References

- [3] [www.astree.ens.fr](http://www.astree.ens.fr) [5, 6, 7, 8, 9, 10, 11, 12, 13]
- [4] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 March 1978.
- [5] 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](#). *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pp. 85–108. Springer, 2002.
- [6] 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](#). *PLDI'03*, San Diego, pp. 196–207, ACM Press, 2003.
- [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 *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY, USA.
- [PACJM '79] P. Cousot and R. Cousot. [Constructive versions of Tarski's fixed point theorems](#). *Pacific Journal of Mathematics* 82(1):43–57 (1979).
- [POPL '78] 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*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY, U.S.A.

- [POPL '79] 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*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY, U.S.A.
- [POPL '92] P. Cousot and R. Cousot. [Inductive Definitions, Semantics and Abstract Interpretation](#). In *Conference Record of the 19<sup>th</sup> ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages*, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.
- [FPCA '95] P. Cousot and R. Cousot. [Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation](#). In *SIGPLAN/SIGARCH/WG2.8 7<sup>th</sup> Conference on Functional Programming and Computer Architecture, FPCA'95*. La Jolla, California, U.S.A., pages 170–181. ACM Press, New York, U.S.A., 25–28 June 1995.
- [POPL '97] P. Cousot. [Types as Abstract Interpretations](#). In *Conference Record of the 24<sup>th</sup> ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages*, pages 316–331, Paris, France, 1997. ACM Press, New York, U.S.A.
- [POPL '00] P. Cousot and R. Cousot. [Temporal abstract interpretation](#). In *Conference Record of the Twentysixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.
- [POPL '02] 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*, pages 178–190, Portland, Oregon, January 2002. ACM Press, New York, NY.
- [TCS 277(1–2) 2002] P. Cousot. [Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation](#). *Theoretical Computer Science* 277(1–2):47–103, 2002.

- [TCS 290(1) 2002] P. Cousot and R. Cousot. [Parsing as abstract interpretation of grammar semantics](#). *Theoret. Comput. Sci.*, 290:531–544, 2003.
- [Manna's festschrift '03] P. Cousot. [Verification by Abstract Interpretation](#). *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, N. Dershowitz (Ed.), Taormina, Italy, June 29 – July 4, 2003. Lecture Notes in Computer Science, vol. 2772, pp. 243–268. © Springer-Verlag, Berlin, Germany, 2003.
- [7] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. [The ASTRÉE analyser](#). *ESOP 2005*, Edinburgh, LNCS 3444, pp. 21–30, Springer, 2005.
- [8] J. Feret. [Static analysis of digital filters](#). *ESOP'04*, Barcelona, LNCS 2986, pp. 33–48, Springer, 2004.
- [9] J. Feret. [The arithmetic-geometric progression abstract domain](#). In *VMCAI'05*, Paris, LNCS 3385, pp. 42–58, Springer, 2005.
- [10] Laurent Mauborgne & Xavier Rival. [Trace Partitioning in Abstract Interpretation Based Static Analyzers](#). *ESOP'05*, Edinburgh, LNCS 3444, pp. 5–20, Springer, 2005.
- [11] A. Miné. [A New Numerical Abstract Domain Based on Difference-Bound Matrices](#). *PADO'2001*, LNCS 2053, Springer, 2001, pp. 155–172.
- [12] A. Miné. [Relational abstract domains for the detection of floating-point run-time errors](#). *ESOP'04*, Barcelona, LNCS 2986, pp. 3–17, Springer, 2004.
- [13] A. Miné. [Weakly Relational Numerical Abstract Domains](#). *PhD Thesis*, École Polytechnique, 6 december 2004.

- [POPL '04] 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*, pages 173–185, Venice, Italy, January 14-16, 2004. ACM Press, New York, NY.
- [DPG-ICALP '05] M. Dalla Preda and R. Giacobazzi. [Semantic-based Code Obfuscation by Abstract Interpretation](#). In *Proc. 32nd Int. Colloquium on Automata, Languages and Programming (ICALP'05 – Track B)*. LNCS, 2005 Springer-Verlag. July 11-15, 2005, Lisboa, Portugal. To appear.
- [EMSOF T '01] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. [Reliable and precise WCET determination for a real-life processor](#). *EMSOF T (2001)*, LNCS 2211, 469–485.
- [RT-ESOP '04] F. Ranzato and F. Tapparo. [Strong Preservation as Completeness in Abstract Interpretation](#). *ESOP 2004*, Barcelona, Spain, March 29 - April 2, 2004, D.A. Schmidt (Ed), LNCS 2986, Springer, 2004, pp. 18–32.