

# Static Analysis of Embedded Control/Command Software by Abstract Interpretation

Patrick Cousot

École normale supérieure  
Paris, France

cousot@ens.fr

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

Radhia Cousot

CNRS & École polytechnique  
Palaiseau, France

Radhia.Cousot@polytechnique.edu

[www.polytechnique.fr/enseignants/rcousot](http://www.polytechnique.fr/enseignants/rcousot)

Kestrel Technology, Palo Alto, CA, Nov. 7<sup>th</sup>, 2005

## Motivation

— 3 —

### Talk Outline

|   |    |
|---|----|
| – Motivation (2 mn) .....                                       | 4  |
| – Abstract interpretation, reminder (12 mn) .....               | 8  |
| – Applications of abstract interpretation (2 mn) .....          | 24 |
| – A practical application to the ASTRÉE static analyzer (18 mn) | 24 |
| – Examples of abstractions in ASTRÉE (12 mn) .....              | 44 |
| – Static analysis of systems (6 mn) .....                       | 58 |
| – Conclusion (2 mn) .....                                       | 66 |

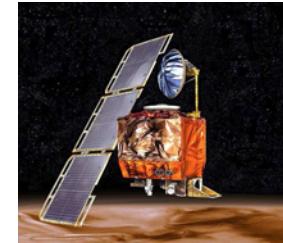
### All Computer Scientists Have Experienced Bugs



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 →

**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.

— 5 —

---

### Abstract Interpretation, Reminder using a simple example

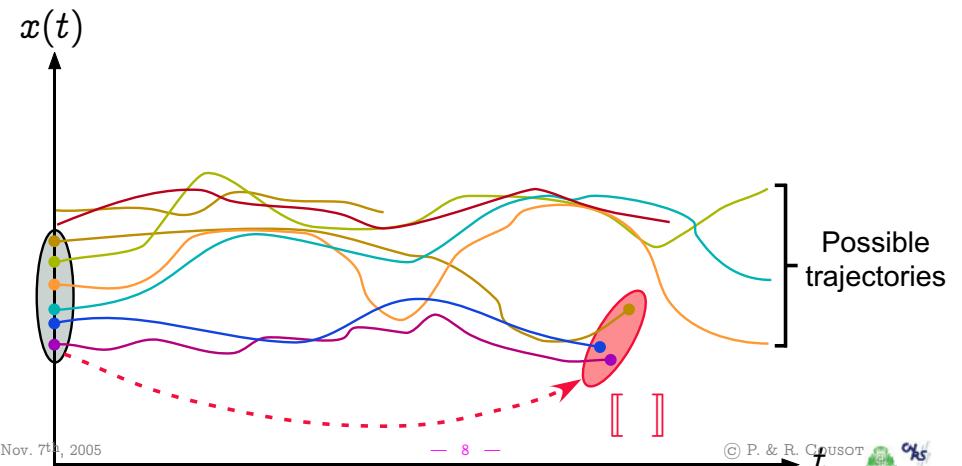
## Syntax of programs

|  |   |
|--|---|
| $X$  | variables $X \in \mathbb{X}$              |
| $T$  | types $T \in \mathbb{T}$                  |
| $E$  | arithmetic expressions $E \in \mathbb{E}$ |
| $B$  | boolean expressions $B \in \mathbb{B}$    |
| $D ::= T\ X;$<br>  $\quad T\ X\ ;\ D'$   |   |
| $C ::= X = E;$<br>  $\quad \text{while } B\ C'$<br>  $\quad \text{if } B\ C'\ \text{else } C''$<br>  $\quad \{ C_1 \dots C_n \}, (n \geq 0)$ | commands $C \in \mathbb{C}$               |
| $P ::= D\ C$   | program $P \in \mathbb{P}$                |

— 7 —

---

## Postcondition semantics



### 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.

## 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>:

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

— 9 —

---

## 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,  $\sqcup$  is disjunction.

## Concrete Reachability Semantics of Programs

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

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

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

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

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

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

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

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

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

~~Not computable (undecidability).~~

## Abstract Semantic Domain of Programs

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

such that:

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

i.e.

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

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

---

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

## Example 1 of Abstraction

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

$\alpha$  **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,  $\subseteq$  pointwise):

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

---

— 13 —

## 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)

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

$\alpha_3$  **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>

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

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

---

— 15 —

## Example 3: Reduced Product of Abstract Domains

To combine abstractions

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

the reduced product is

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

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

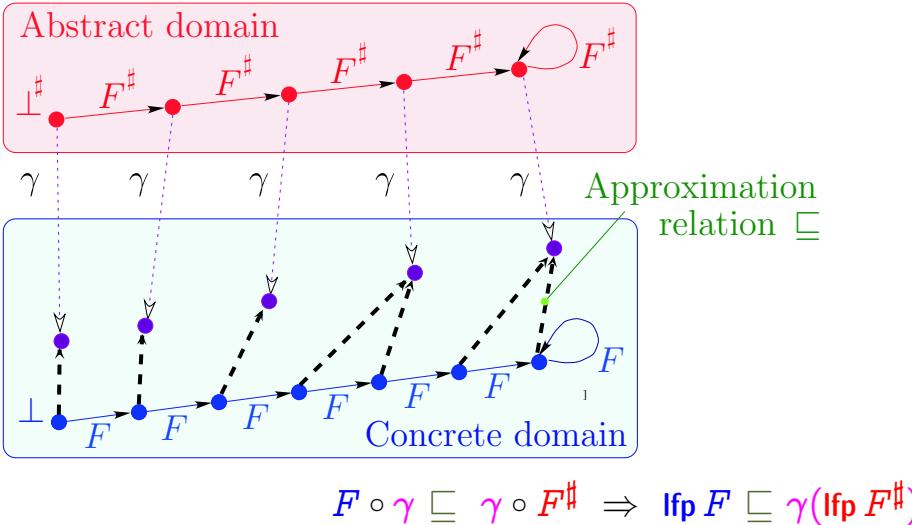
$$\langle \mathcal{D}, \subseteq \rangle \xrightleftharpoons[\alpha]{\gamma_1 \times \gamma_2} \langle \alpha(\mathcal{D}), \sqsubseteq \rangle$$

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

---

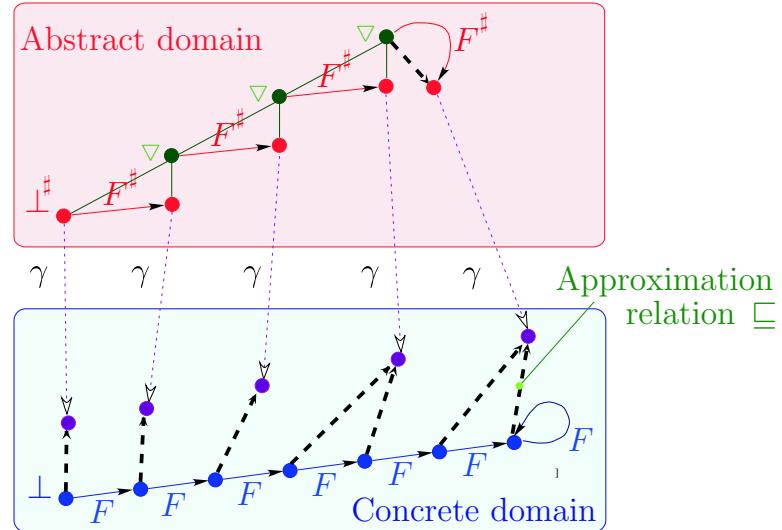
<sup>2</sup> assuming these values to be totally ordered.

## Approximate Fixpoint Abstraction



— 17 —

## Convergence Acceleration with Widening



— 19 —

## Abstract Reachability Semantics of Programs

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

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

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

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

## Applications of Abstract Interpretation

— 21 —

### 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**

— 23 —

### 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)

---

— 25 —

– without

- union (new memory model in progress<sup>4</sup>)
- 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. encoding 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. volatile environment specified by a *trusted configuration file*, assert, execution stops on first runtime error<sup>5</sup> ,)

---

— 27 —

## 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, no float 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).

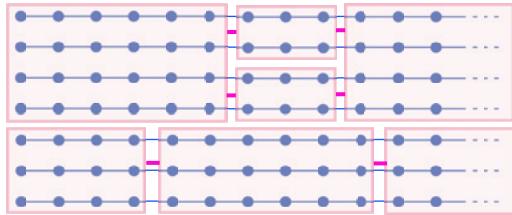
---

<sup>4</sup> Thanks A. Miné

<sup>5</sup> semantics of C unclear after an error, equivalent if no alarm

## Abstraction

- Set of traces of relational state abstractions of subtraces for the concrete trace operational semantics



— 29 —

## Requirements on the Abstract Semantics

- **Soundness:** absolutely essential for verification
- **Precision:** few or no false alarm<sup>6</sup> (full certification)
- **Efficiency:** rapid analyses and fixes during development

## Example of Industrial applications

- 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: × 3/7 (up to 1.000.000 LOCs)

— 31 —

## 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].

<sup>6</sup> Potential runtime error signaled by the analyzer due to overapproximation but impossible in any actual program run compatible with the configuration file.

## 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

— 33 —

## 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)

— 35 —

## 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)

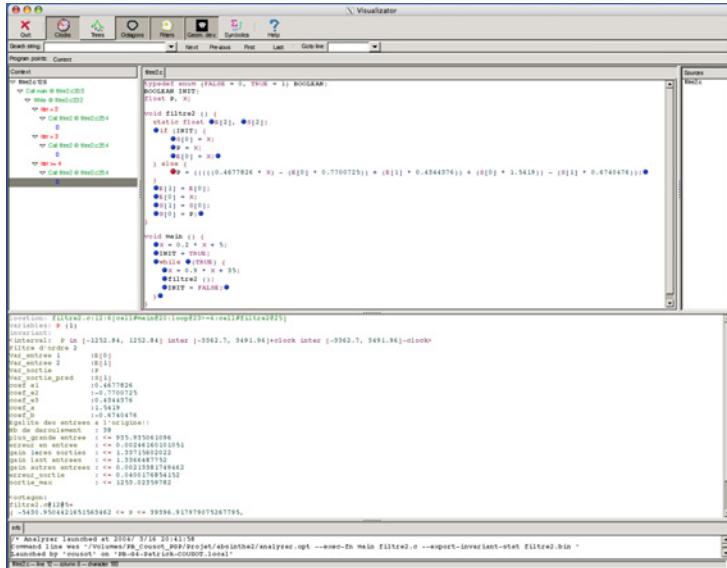
**Automatic Parametrization:** the generation of parametric directives in the code can be programmed (to be specialized for a specific application domain)

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

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

— 37 —

### Example of Analysis Session



The screenshot shows the ASTRÉE Visualizer interface. The top window is a code editor displaying C-like pseudocode with annotations. The bottom window is a visualization pane showing a state transition graph or a similar analysis result.

```
/* Main.c */
void main() {
    static float x[5];
    static float y[5];
    static float z[5];
    static float w[5];
    static float v[5];
    static float t[5];
    static float u[5];
    static float s[5];
    static float r[5];
    static float q[5];
    static float p[5];
    static float o[5];
    static float n[5];
    static float m[5];
    static float l[5];
    static float k[5];
    static float j[5];
    static float i[5];
    static float h[5];
    static float g[5];
    static float f[5];
    static float e[5];
    static float d[5];
    static float c[5];
    static float b[5];
    static float a[5];
    static float b1[5];
    static float b2[5];
    static float b3[5];
    static float b4[5];
    static float b5[5];
    static float b6[5];
    static float b7[5];
    static float b8[5];
    static float b9[5];
    static float b10[5];
    static float b11[5];
    static float b12[5];
    static float b13[5];
    static float b14[5];
    static float b15[5];
    static float b16[5];
    static float b17[5];
    static float b18[5];
    static float b19[5];
    static float b20[5];
    static float b21[5];
    static float b22[5];
    static float b23[5];
    static float b24[5];
    static float b25[5];
    static float b26[5];
    static float b27[5];
    static float b28[5];
    static float b29[5];
    static float b30[5];
    static float b31[5];
    static float b32[5];
    static float b33[5];
    static float b34[5];
    static float b35[5];
    static float b36[5];
    static float b37[5];
    static float b38[5];
    static float b39[5];
    static float b40[5];
    static float b41[5];
    static float b42[5];
    static float b43[5];
    static float b44[5];
    static float b45[5];
    static float b46[5];
    static float b47[5];
    static float b48[5];
    static float b49[5];
    static float b50[5];
    static float b51[5];
    static float b52[5];
    static float b53[5];
    static float b54[5];
    static float b55[5];
    static float b56[5];
    static float b57[5];
    static float b58[5];
    static float b59[5];
    static float b60[5];
    static float b61[5];
    static float b62[5];
    static float b63[5];
    static float b64[5];
    static float b65[5];
    static float b66[5];
    static float b67[5];
    static float b68[5];
    static float b69[5];
    static float b70[5];
    static float b71[5];
    static float b72[5];
    static float b73[5];
    static float b74[5];
    static float b75[5];
    static float b76[5];
    static float b77[5];
    static float b78[5];
    static float b79[5];
    static float b80[5];
    static float b81[5];
    static float b82[5];
    static float b83[5];
    static float b84[5];
    static float b85[5];
    static float b86[5];
    static float b87[5];
    static float b88[5];
    static float b89[5];
    static float b90[5];
    static float b91[5];
    static float b92[5];
    static float b93[5];
    static float b94[5];
    static float b95[5];
    static float b96[5];
    static float b97[5];
    static float b98[5];
    static float b99[5];
    static float b100[5];
    static float b101[5];
    static float b102[5];
    static float b103[5];
    static float b104[5];
    static float b105[5];
    static float b106[5];
    static float b107[5];
    static float b108[5];
    static float b109[5];
    static float b110[5];
    static float b111[5];
    static float b112[5];
    static float b113[5];
    static float b114[5];
    static float b115[5];
    static float b116[5];
    static float b117[5];
    static float b118[5];
    static float b119[5];
    static float b120[5];
    static float b121[5];
    static float b122[5];
    static float b123[5];
    static float b124[5];
    static float b125[5];
    static float b126[5];
    static float b127[5];
    static float b128[5];
    static float b129[5];
    static float b130[5];
    static float b131[5];
    static float b132[5];
    static float b133[5];
    static float b134[5];
    static float b135[5];
    static float b136[5];
    static float b137[5];
    static float b138[5];
    static float b139[5];
    static float b140[5];
    static float b141[5];
    static float b142[5];
    static float b143[5];
    static float b144[5];
    static float b145[5];
    static float b146[5];
    static float b147[5];
    static float b148[5];
    static float b149[5];
    static float b150[5];
    static float b151[5];
    static float b152[5];
    static float b153[5];
    static float b154[5];
    static float b155[5];
    static float b156[5];
    static float b157[5];
    static float b158[5];
    static float b159[5];
    static float b160[5];
    static float b161[5];
    static float b162[5];
    static float b163[5];
    static float b164[5];
    static float b165[5];
    static float b166[5];
    static float b167[5];
    static float b168[5];
    static float b169[5];
    static float b170[5];
    static float b171[5];
    static float b172[5];
    static float b173[5];
    static float b174[5];
    static float b175[5];
    static float b176[5];
    static float b177[5];
    static float b178[5];
    static float b179[5];
    static float b180[5];
    static float b181[5];
    static float b182[5];
    static float b183[5];
    static float b184[5];
    static float b185[5];
    static float b186[5];
    static float b187[5];
    static float b188[5];
    static float b189[5];
    static float b190[5];
    static float b191[5];
    static float b192[5];
    static float b193[5];
    static float b194[5];
    static float b195[5];
    static float b196[5];
    static float b197[5];
    static float b198[5];
    static float b199[5];
    static float b200[5];
    static float b201[5];
    static float b202[5];
    static float b203[5];
    static float b204[5];
    static float b205[5];
    static float b206[5];
    static float b207[5];
    static float b208[5];
    static float b209[5];
    static float b210[5];
    static float b211[5];
    static float b212[5];
    static float b213[5];
    static float b214[5];
    static float b215[5];
    static float b216[5];
    static float b217[5];
    static float b218[5];
    static float b219[5];
    static float b220[5];
    static float b221[5];
    static float b222[5];
    static float b223[5];
    static float b224[5];
    static float b225[5];
    static float b226[5];
    static float b227[5];
    static float b228[5];
    static float b229[5];
    static float b230[5];
    static float b231[5];
    static float b232[5];
    static float b233[5];
    static float b234[5];
    static float b235[5];
    static float b236[5];
    static float b237[5];
    static float b238[5];
    static float b239[5];
    static float b240[5];
    static float b241[5];
    static float b242[5];
    static float b243[5];
    static float b244[5];
    static float b245[5];
    static float b246[5];
    static float b247[5];
    static float b248[5];
    static float b249[5];
    static float b250[5];
    static float b251[5];
    static float b252[5];
    static float b253[5];
    static float b254[5];
    static float b255[5];
    static float b256[5];
    static float b257[5];
    static float b258[5];
    static float b259[5];
    static float b260[5];
    static float b261[5];
    static float b262[5];
    static float b263[5];
    static float b264[5];
    static float b265[5];
    static float b266[5];
    static float b267[5];
    static float b268[5];
    static float b269[5];
    static float b270[5];
    static float b271[5];
    static float b272[5];
    static float b273[5];
    static float b274[5];
    static float b275[5];
    static float b276[5];
    static float b277[5];
    static float b278[5];
    static float b279[5];
    static float b280[5];
    static float b281[5];
    static float b282[5];
    static float b283[5];
    static float b284[5];
    static float b285[5];
    static float b286[5];
    static float b287[5];
    static float b288[5];
    static float b289[5];
    static float b290[5];
    static float b291[5];
    static float b292[5];
    static float b293[5];
    static float b294[5];
    static float b295[5];
    static float b296[5];
    static float b297[5];
    static float b298[5];
    static float b299[5];
    static float b300[5];
    static float b301[5];
    static float b302[5];
    static float b303[5];
    static float b304[5];
    static float b305[5];
    static float b306[5];
    static float b307[5];
    static float b308[5];
    static float b309[5];
    static float b310[5];
    static float b311[5];
    static float b312[5];
    static float b313[5];
    static float b314[5];
    static float b315[5];
    static float b316[5];
    static float b317[5];
    static float b318[5];
    static float b319[5];
    static float b320[5];
    static float b321[5];
    static float b322[5];
    static float b323[5];
    static float b324[5];
    static float b325[5];
    static float b326[5];
    static float b327[5];
    static float b328[5];
    static float b329[5];
    static float b330[5];
    static float b331[5];
    static float b332[5];
    static float b333[5];
    static float b334[5];
    static float b335[5];
    static float b336[5];
    static float b337[5];
    static float b338[5];
    static float b339[5];
    static float b340[5];
    static float b341[5];
    static float b342[5];
    static float b343[5];
    static float b344[5];
    static float b345[5];
    static float b346[5];
    static float b347[5];
    static float b348[5];
    static float b349[5];
    static float b350[5];
    static float b351[5];
    static float b352[5];
    static float b353[5];
    static float b354[5];
    static float b355[5];
    static float b356[5];
    static float b357[5];
    static float b358[5];
    static float b359[5];
    static float b360[5];
    static float b361[5];
    static float b362[5];
    static float b363[5];
    static float b364[5];
    static float b365[5];
    static float b366[5];
    static float b367[5];
    static float b368[5];
    static float b369[5];
    static float b370[5];
    static float b371[5];
    static float b372[5];
    static float b373[5];
    static float b374[5];
    static float b375[5];
    static float b376[5];
    static float b377[5];
    static float b378[5];
    static float b379[5];
    static float b380[5];
    static float b381[5];
    static float b382[5];
    static float b383[5];
    static float b384[5];
    static float b385[5];
    static float b386[5];
    static float b387[5];
    static float b388[5];
    static float b389[5];
    static float b390[5];
    static float b391[5];
    static float b392[5];
    static float b393[5];
    static float b394[5];
    static float b395[5];
    static float b396[5];
    static float b397[5];
    static float b398[5];
    static float b399[5];
    static float b400[5];
    static float b401[5];
    static float b402[5];
    static float b403[5];
    static float b404[5];
    static float b405[5];
    static float b406[5];
    static float b407[5];
    static float b408[5];
    static float b409[5];
    static float b410[5];
    static float b411[5];
    static float b412[5];
    static float b413[5];
    static float b414[5];
    static float b415[5];
    static float b416[5];
    static float b417[5];
    static float b418[5];
    static float b419[5];
    static float b420[5];
    static float b421[5];
    static float b422[5];
    static float b423[5];
    static float b424[5];
    static float b425[5];
    static float b426[5];
    static float b427[5];
    static float b428[5];
    static float b429[5];
    static float b430[5];
    static float b431[5];
    static float b432[5];
    static float b433[5];
    static float b434[5];
    static float b435[5];
    static float b436[5];
    static float b437[5];
    static float b438[5];
    static float b439[5];
    static float b440[5];
    static float b441[5];
    static float b442[5];
    static float b443[5];
    static float b444[5];
    static float b445[5];
    static float b446[5];
    static float b447[5];
    static float b448[5];
    static float b449[5];
    static float b450[5];
    static float b451[5];
    static float b452[5];
    static float b453[5];
    static float b454[5];
    static float b455[5];
    static float b456[5];
    static float b457[5];
    static float b458[5];
    static float b459[5];
    static float b460[5];
    static float b461[5];
    static float b462[5];
    static float b463[5];
    static float b464[5];
    static float b465[5];
    static float b466[5];
    static float b467[5];
    static float b468[5];
    static float b469[5];
    static float b470[5];
    static float b471[5];
    static float b472[5];
    static float b473[5];
    static float b474[5];
    static float b475[5];
    static float b476[5];
    static float b477[5];
    static float b478[5];
    static float b479[5];
    static float b480[5];
    static float b481[5];
    static float b482[5];
    static float b483[5];
    static float b484[5];
    static float b485[5];
    static float b486[5];
    static float b487[5];
    static float b488[5];
    static float b489[5];
    static float b490[5];
    static float b491[5];
    static float b492[5];
    static float b493[5];
    static float b494[5];
    static float b495[5];
    static float b496[5];
    static float b497[5];
    static float b498[5];
    static float b499[5];
    static float b500[5];
}
```

— 38 —

© P. & R. 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!

— 39 —

## (Airbus A380 Primary Flight Control Software)

— 350,000 lines

— 0 alarms (Nov. 2004),

7h<sup>7</sup> on 2.8 GHz PC,

1 Gigabyte

→ A world grand première!

<sup>7</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.

Nov. 7th, 2005

— 40 —

© P. & R. Cousot 

## Floating-Point Computations

### Examples of Abstractions

— 41 —

```
/* 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
```

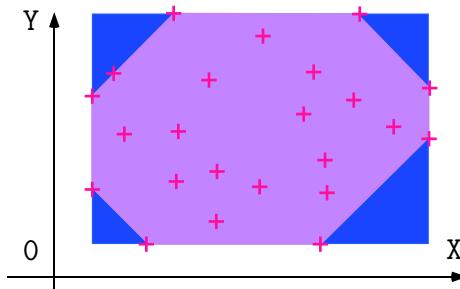
```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

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

— 43 —

## Floating-Point Computations

### General-Purpose Abstract Domains: Intervals and Octagons



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

Octagons [10]:  
 $\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 20 \\ x - y \leq 04 \end{cases}$

**Difficulties:** many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [POPL '77, 10, 11]

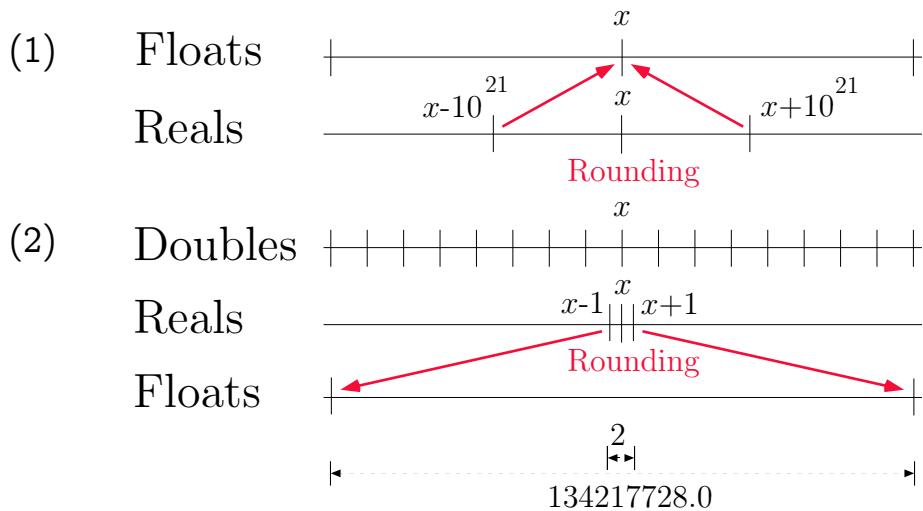
```
/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.000000019e+38;
    y = x + 1.0e21;
    z = x - 1.0e21;
    r = y - z;
    printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* 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
```

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

— 43 —

## Explanation of the huge rounding error



## Floating-point linearization [11, 12]

- 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

— 45 —

## Symbolic abstract domain [11, 12]

- 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);

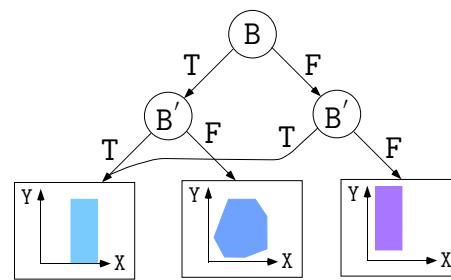
```
% cat -n x-x.c
 1 void main () { int X, Y;
 2     __ASTREE_known_fact(((0 <= X) && (X <= 100)));
 3     Y = (X - X);
 4     __ASTREE_log_vars((Y));
 5 }

astree -exec-fn main -no-relational x-x.c          astree -exec-fn main x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:                   Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in [-100, 100]>                  <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

— 47 —

## Control Partitionning for Case Analysis

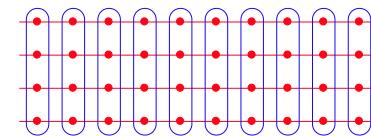
- Code Sample:

```
/* 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;

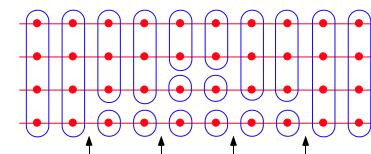
    ... found invariant  $-100 \leq x \leq 100$  ...

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

Control point partitionning:



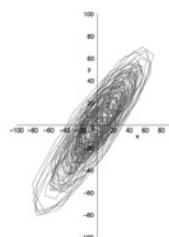
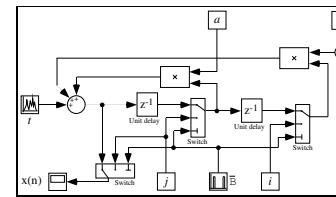
Trace partitionning:



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

— 50 —

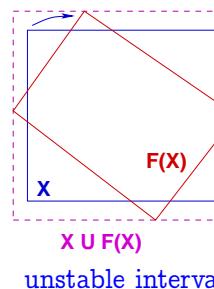
2<sup>d</sup> Order Digital Filter:



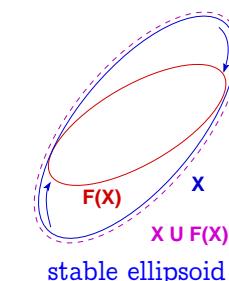
execution trace

## 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.



unstable interval



stable ellipsoid

— 49 —

## Filter Example [7]

```
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>8</sup> [8]

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

– Concretization:

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

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

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

## 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; }           ← potential overflow!
        else { R = 0; }
        T = (R >= 100);
        __ASTREE_wait_for_clock();
    }
}

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

<sup>8</sup> here in  $\mathbb{R}$

## 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;}
}

% 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
```

— 53 —

## (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.

— 55 —

## Possible origins of imprecision and how to fix it

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

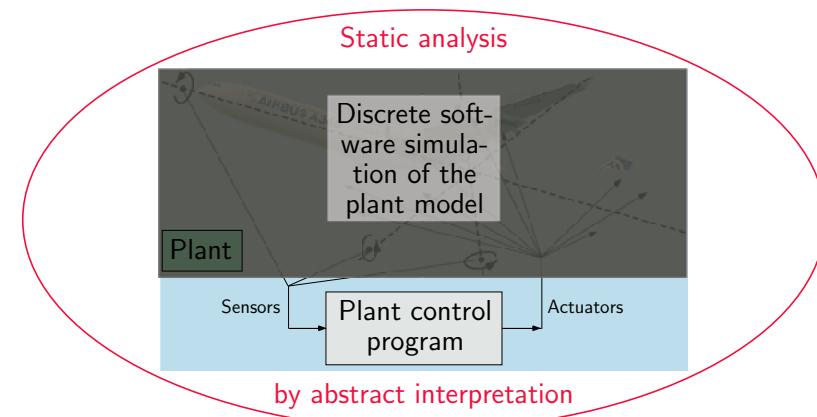
- **Abstract transformers** (not best possible)  $\rightarrow$  improve algorithm;
- **Automated parametrization** (e.g. variable packing)  $\rightarrow$  improve pattern-matched program schemata;
- **Iteration strategy** for fixpoints  $\rightarrow$  fix widening<sup>9</sup>;
- **Inexpressivity** i.e. indispensable local inductive invariant are inexpressible in the abstract  $\rightarrow$  add a **new abstract domain** to the reduced product (e.g. filters).

<sup>9</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.

## Static analysis of systems

— 57 —

## System analysis & verification, Avenue 1



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

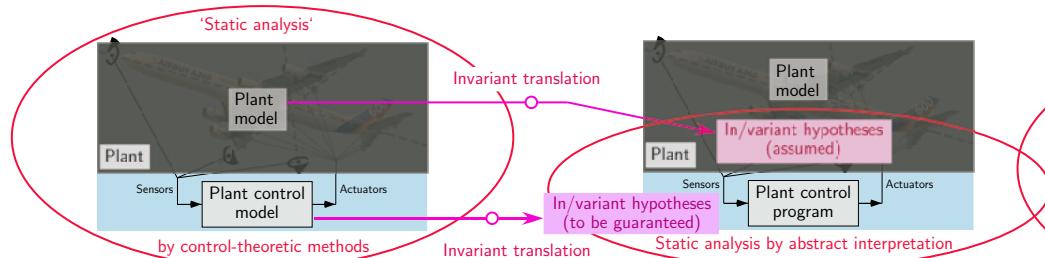
- Exhaustive (contrary to current simulations)
- The **plant model discretization errors** are similar to those of simulation methods (but for the use of the *actual* control program instead of a model!)
- In general, **polyhedral abstractions** are unstable or of very high complexity
- New abstractions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

- 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)

— 59 —

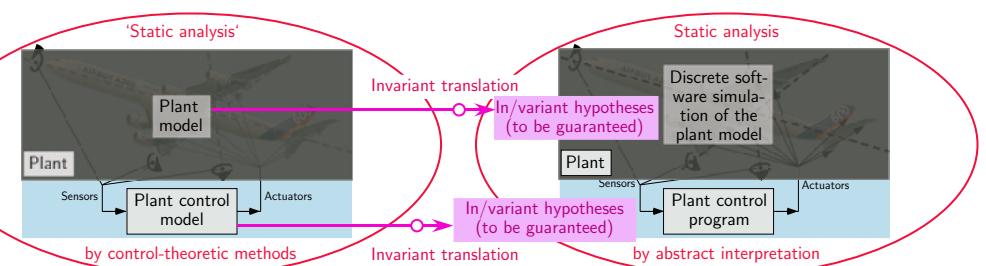
— 61 —

### System analysis & verification, Avenue 2



Abstractions: program → precise, system → precise

### System analysis & verification, Avenue 3



Abstractions: program → precise, system → 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>10</sup>)

---

— 63 —

## 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.

---

— 65 —

## 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

---

<sup>10</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

# THE END, THANK YOU

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

— 67 —

## References

- [2] [www.astree.ens.fr](http://www.astree.ens.fr) [4, 5, 6, 7, 8, 9, 10, 11, 12]
- [3] 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.
- [4] 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.
- [5] 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 Twentyseventh 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.
- [6] 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.
- [7] J. Feret. *Static analysis of digital filters*. ESOP'04, Barcelona, LNCS 2986, pp. 33–48, Springer, 2004.
- [8] J. Feret. *The arithmetic-geometric progression abstract domain*. In VMCAI'05, Paris, LNCS 3385, pp. 42–58, Springer, 2005.
- [9] Laurent Mauborgne & Xavier Rival. *Trace Partitioning in Abstract Interpretation Based Static Analyzers*. ESOP'05, Edinburgh, LNCS 3444, pp. 5–20, Springer, 2005.
- [10] A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices*. PADO'2001, LNCS 2053, Springer, 2001, pp. 155–172.
- [11] A. Miné. *Relational abstract domains for the detection of floating-point run-time errors*. ESOP'04, Barcelona, LNCS 2986, pp. 3–17, Springer, 2004.
- [12] A. Miné. *Weakly Relational Numerical Abstract Domains*. PhD Thesis, École Polytechnique, 6 december 2004.

— 69 —

[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.

[EMSOFT '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*. EMSOFT (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.