

# Application of Abstract Interpretation to the Static Verification of Safety Critical Code

Patrick Cousot

École normale supérieure, Paris

cousot@ens.fr www.di.ens.fr/~cousot

IBM Research Seminar, Thomas J. Watson Research Center,  
Hawthorne, NY, January 20, 2006

## Talk Outline

- Motivation (1 mn) ..... 3
- Abstract interpretation, reminder (10 mn) ..... 6
- Applications of abstract interpretation (2 mn) ..... 21
- A practical application to the ASTRÉE static analyzer (15 mn) 24
- Examples of abstractions in ASTRÉE (15 mn) ..... 40
- Conclusion (2 mn) ..... 56



IBM Research January 20, 2006

— 2 —

© P. Cousot 

## Motivation

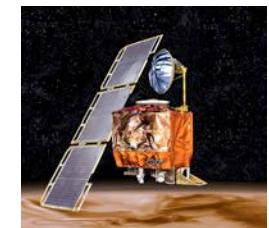
## All Computer Scientists Have Experienced Bugs



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.



IBM Research January 20, 2006

— 3 —

© P. Cousot 



IBM Research January 20, 2006

— 4 —

© P. Cousot 

## Static Analysis by Abstract Interpretation

**Static analysis:** analyze the program at compile-time to verify a program runtime property (e.g. the absence of some categories of bugs)

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

### 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*.

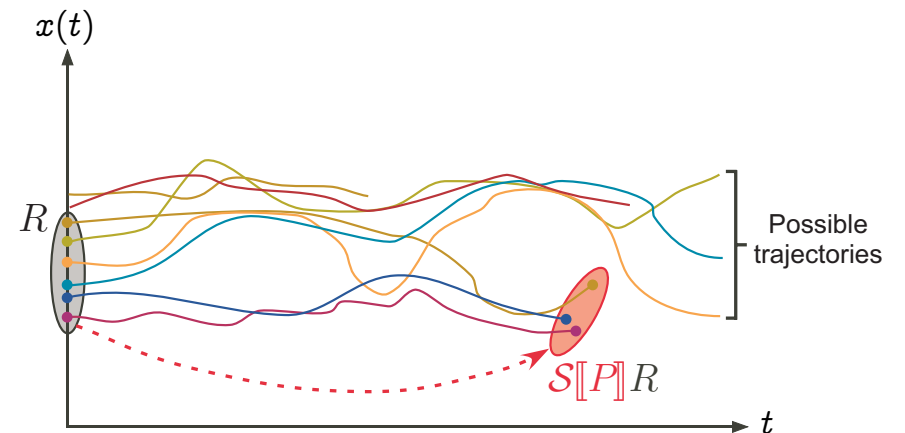


## Syntax of programs

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

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

<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  $\sqsubseteq$  is implication,  $\emptyset$  is false,  $\sqcup$  is disjunction.



## Concrete Reachability Semantics of Programs

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

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

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

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

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

$$S[\text{while } B \ C']R \stackrel{\text{def}}{=} \text{let } \mathcal{W} = \text{fp}_0^{\subseteq} \lambda \mathcal{X}. R \cup S[C'](\mathcal{B}[B]\mathcal{X}) \\ \text{in } (\mathcal{B}[\neg B]\mathcal{W})$$

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

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

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

Not computable (undecidability).



## Abstract Semantic Domain of Programs

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

such that:

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

i.e.

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

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



### Example 1 of Abstraction

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

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

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

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

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



### Example 2 of Abstraction

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

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

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

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

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



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

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

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

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

$\alpha_1, \alpha_2, \alpha_3$  and  $\alpha_4$ , whence  $\alpha_4 \circ \alpha_3 \circ \alpha_2 \circ \alpha_1$  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

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

the **reduced product** is

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

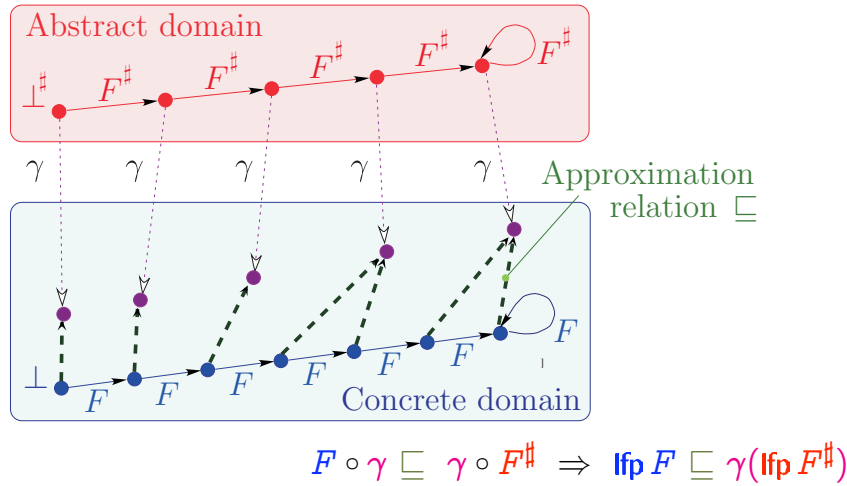
such that  $\sqsubseteq \stackrel{\text{def}}{=} \sqsubseteq_1 \times \sqsubseteq_2$  and

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

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



## Approximate Fixpoint Abstraction

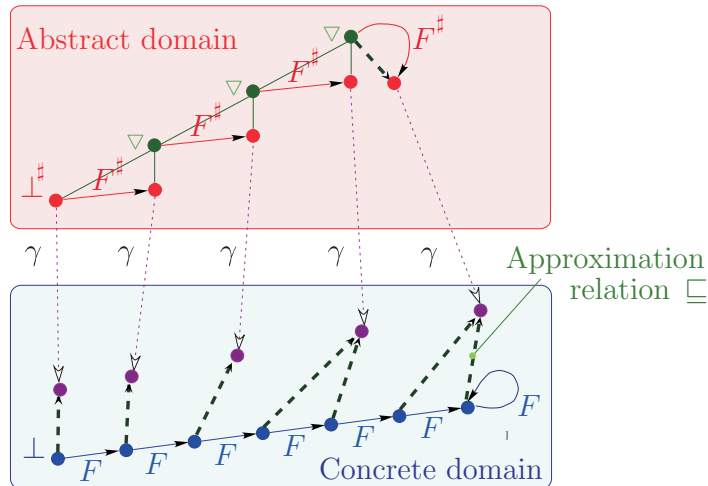


## Abstract Reachability Semantics of Programs

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



## Convergence Acceleration with Widening



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

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

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



## Applications of Abstract Interpretation



## Applications of Abstract Interpretation

- **Static Program Analysis** [POPL '77], [POPL '78], [POPL '79] including **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]



## 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/>



## 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:  $\times 3$



## 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** [EMSOF'T'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  $\rightarrow$  it is a **VERIFIER!**



## Example of Analysis Session



## 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!

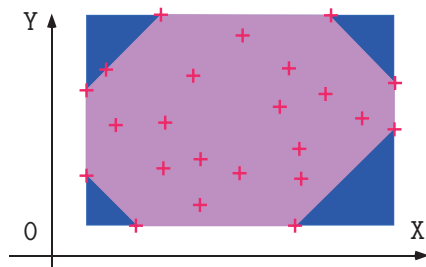
<sup>6</sup> We are still in a phase where we favour precision rather than computation costs, and this should go down. For example, the A340 analysis went up to 5 h, before being reduced by requiring less precision while still getting no false alarm.



## 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 [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]



## 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

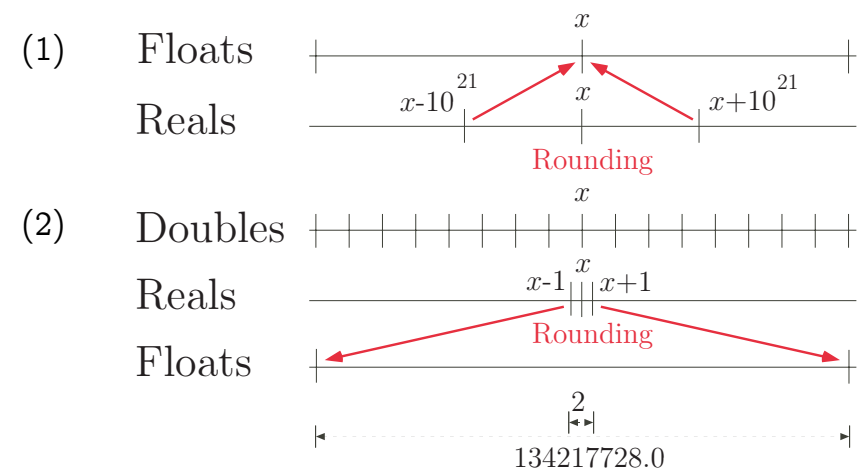
```
/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.000000019e+38;
    y = x + 1.0e21;
    z = x - 1.0e21;
    r = y - z;
    printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* 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



## 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



## 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 }

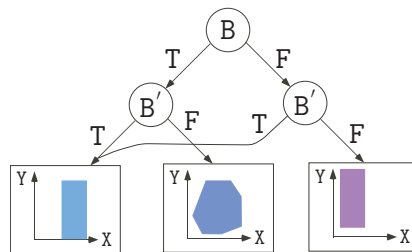
astree -exec-fn main -no-relational x-x.c      astree -exec-fn main x-x.c
Call main@x-x.c:1:5-x-x.c:1:9;                Call main@x-x.c:1:5-x-x.c:1:9;
<interval: Y in [-100, 100]>                   <interval: Y in {0}> <symbolic: Y = (X -i X)>
```



## 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



## Control Partitionning for Case Analysis

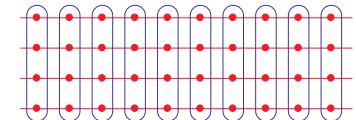
### Code Sample:

```
/* trace_partitioning.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;

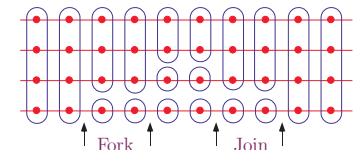
  ... found invariant -100 ≤ x ≤ 100 ...

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

### Control point partitioning:



### Trace partitioning:

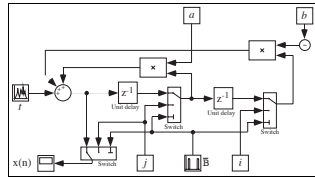


Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

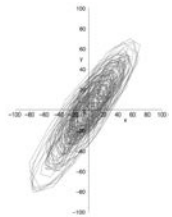


## 2<sup>d</sup> Order Digital Filter:

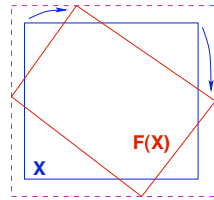
## Ellipsoid Abstract Domain for Filters



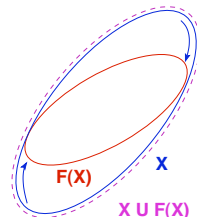
- Computes  $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is **bounded**, which must be proved in the abstract.
- There is **no stable interval or octagon**.
- The simplest stable surface is an **ellipsoid**.



execution trace



$X \cup F(X)$   
unstable interval



$X \cup F(X)$   
stable ellipsoid



**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>7</sup> [8]

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

– Concretization:

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

$$\gamma(M, a, b, a', b') =$$

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

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();
    }
}

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

|R| <= 0. + clock *1. <= 3600001.
```

← potential overflow!



## 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)  $\rightarrow$  improve algorithm;
- **Automatized parametrization** (e.g. variable packing)  $\rightarrow$  improve pattern-matched program schemata;
- **Iteration strategy** for fixpoints  $\rightarrow$  fix widening <sup>8</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>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.



## Conclusion

- Most applications of abstract interpretation **tolerate a small rate** (typically 5 to 15%) **of false alarms**:
  - Program transformation → do not optimize,
  - Typing → reject some correct programs, etc,
  - WCET analysis → overestimate;
- Some applications **require no false alarm** at all:
  - **Program verification**.
- **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.



## The Future & Grand Challenges

### Forthcoming (1 year):

- More general memory model (union)

### Future (5 years):

- **Asynchronous concurrency** (for less critical software)
- **Functional properties** (reactivity)
- **Industrialization**

### Grand challenge:

- **Verification from specifications to machine code** (verifying compiler)
- **Verification of systems** (quasi-synchrony, distribution)



# 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

- [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 19th 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 7th 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 24th 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.



- [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.

