Applications of Abstract Interpretation

Patrick Cousot
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
http://www.di.ens.fr/~cousot

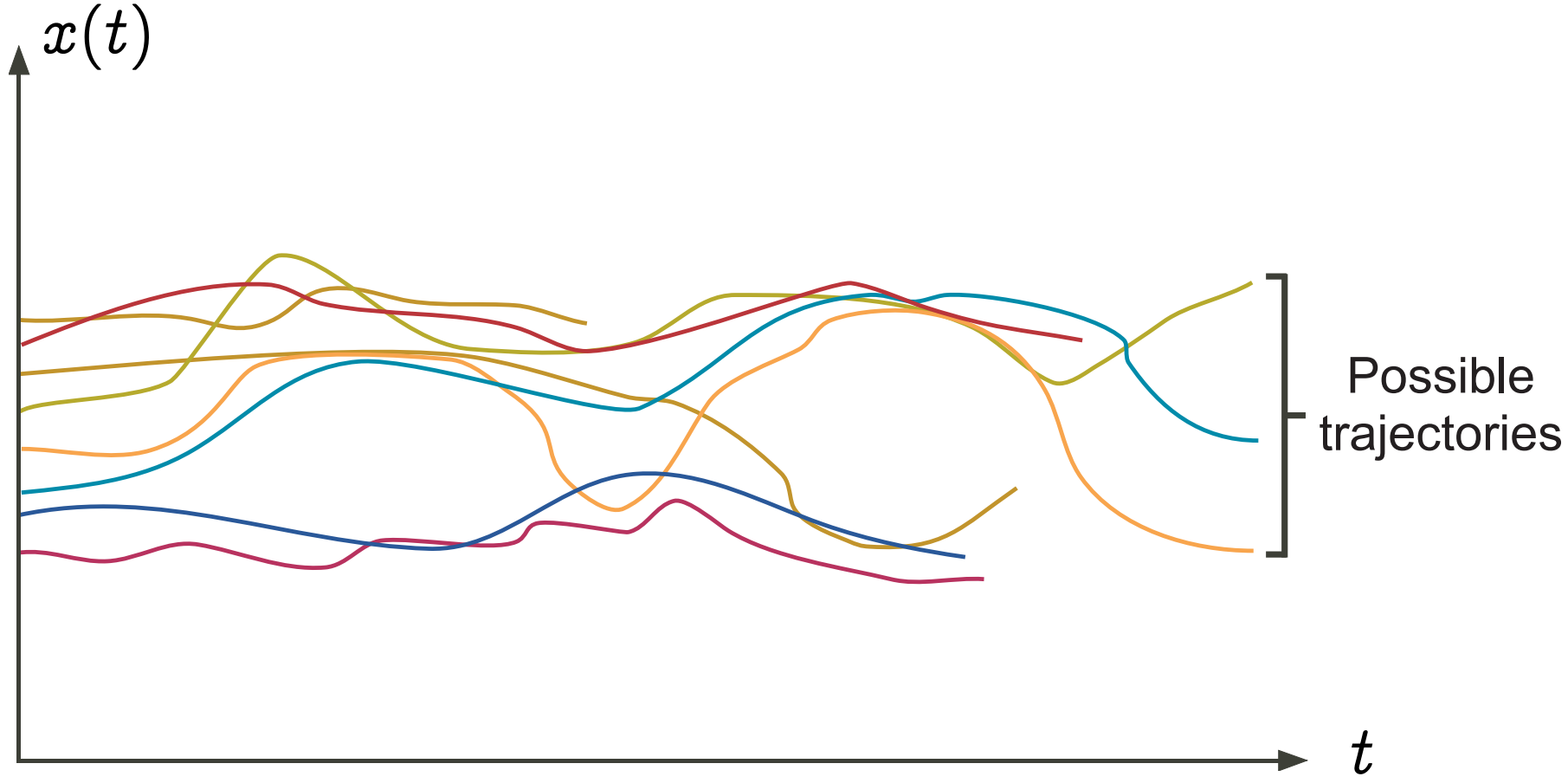
Content

– A brief introduction to Abstract Interpretation (in pictures [1]);
– A rapid survey of the main applications of Abstract Interpretation;
– A more detailled description of the Astrée [2] static analyzer.

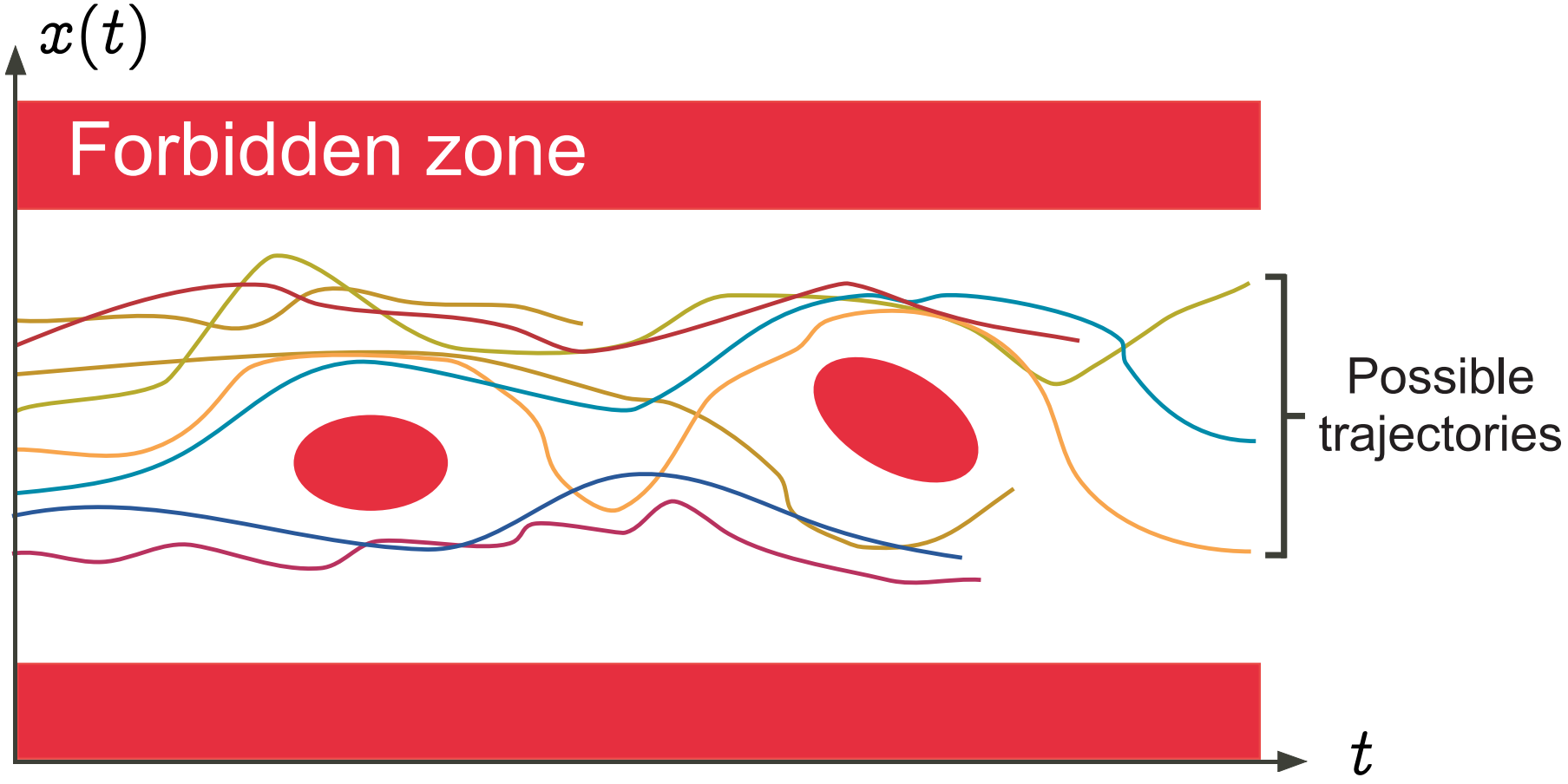
[1] For those attendees that missed the morning session!
[2] `www.astree.ens.fr`

1. Principles of Abstract Interpretation

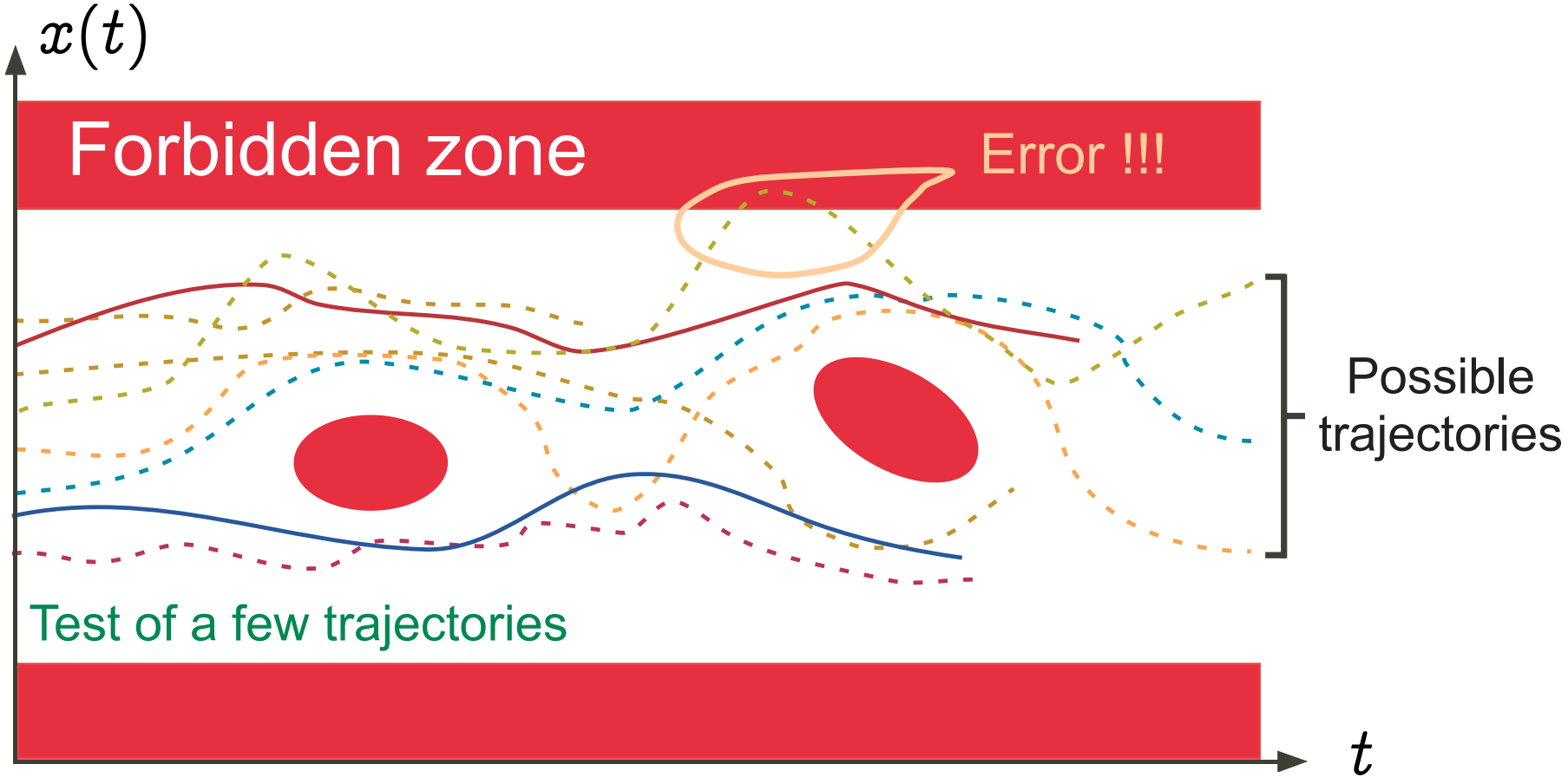
Operational semantics



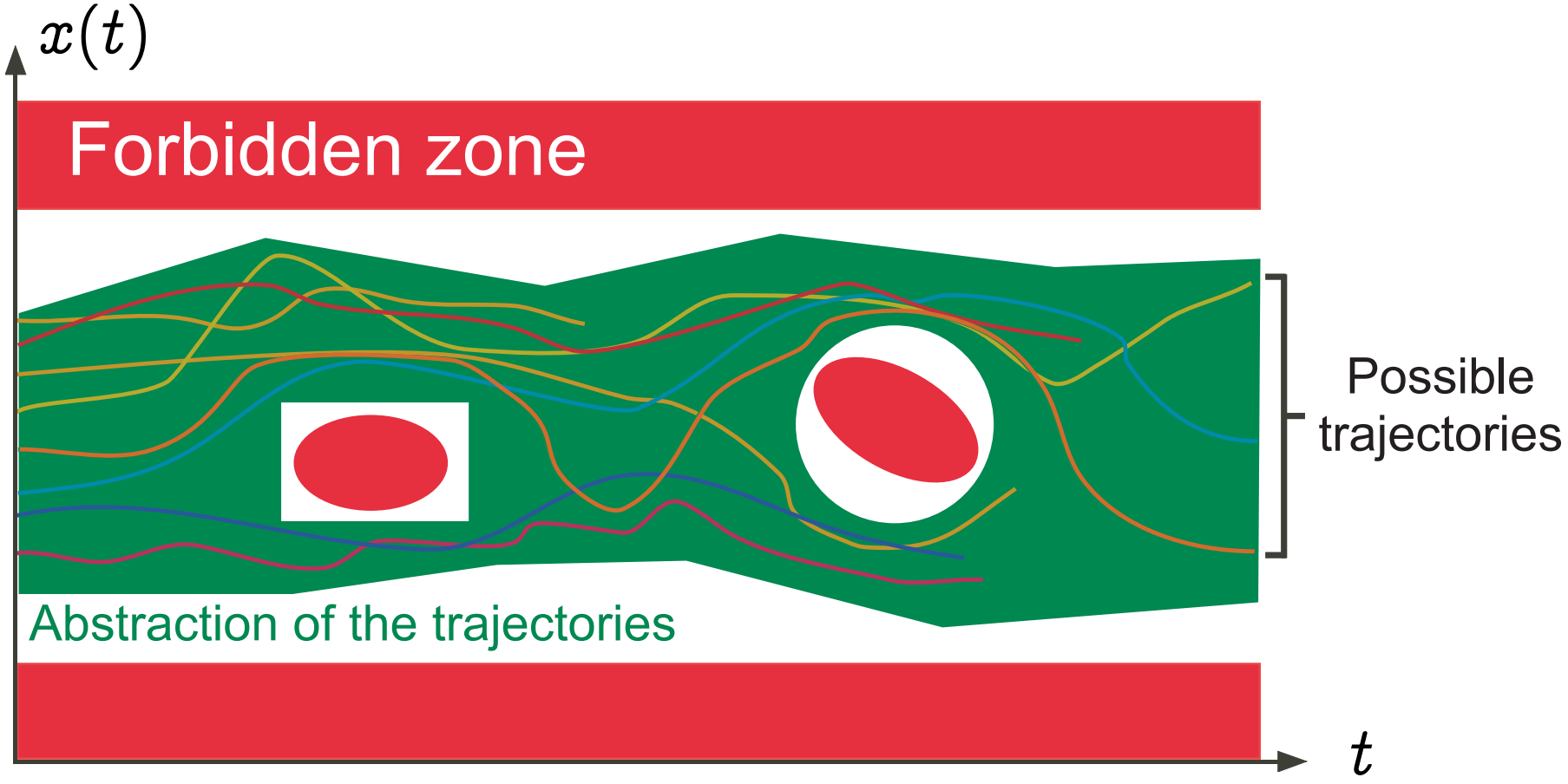
Safety property



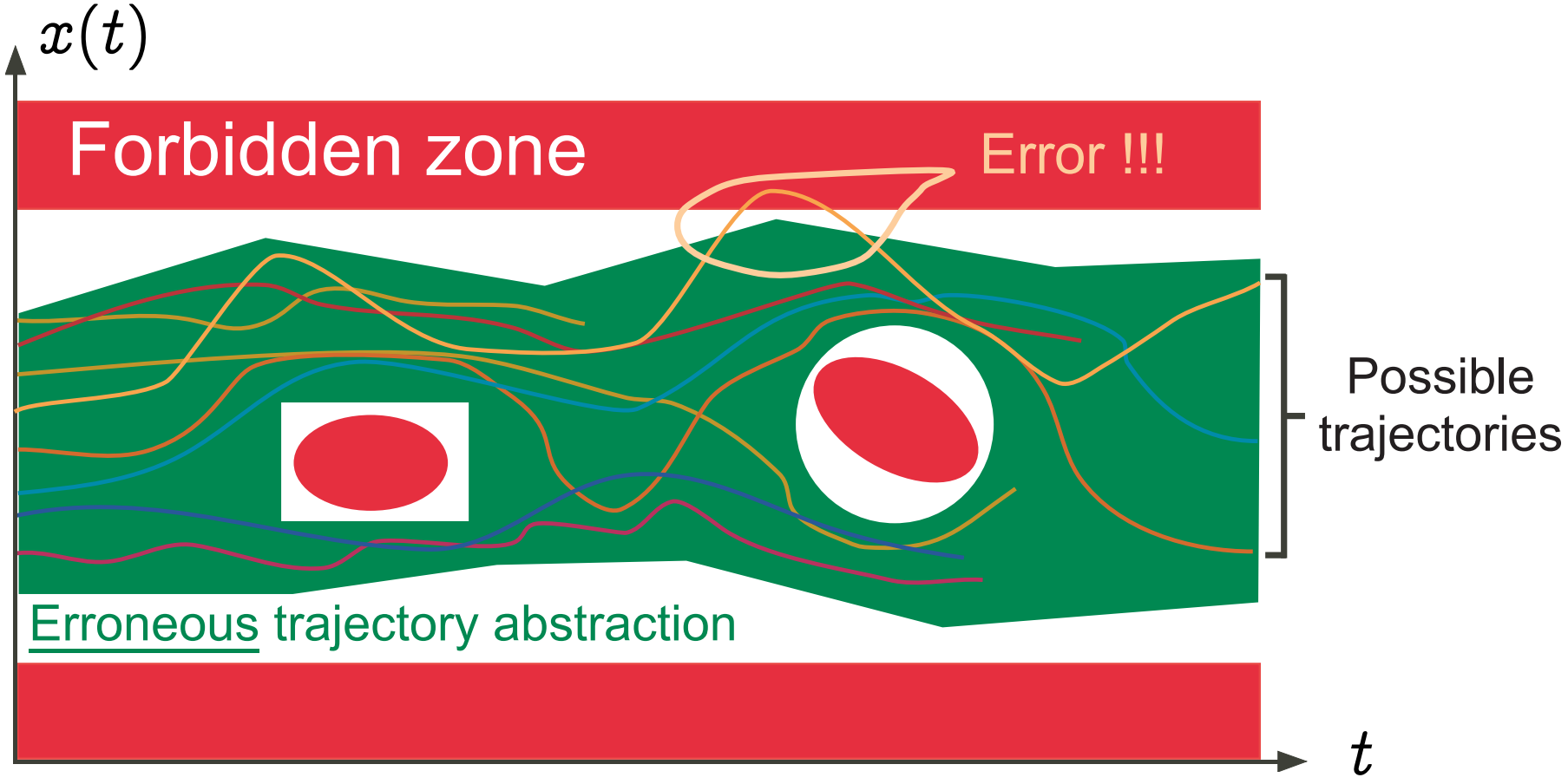
Test/debugging is unsafe



Abstract interpretation is safe

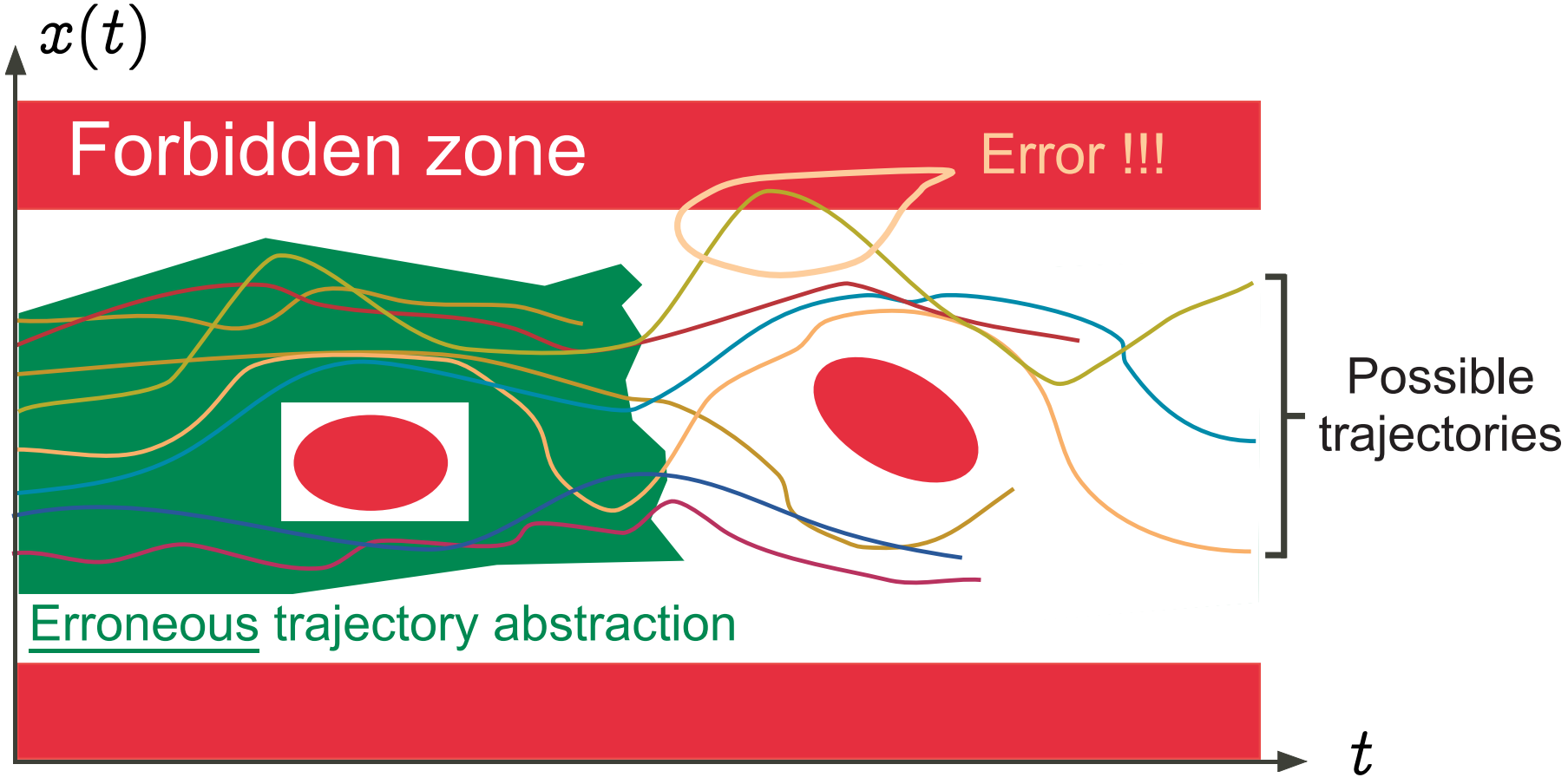


Soundness requirement: erroneous abstraction [3]

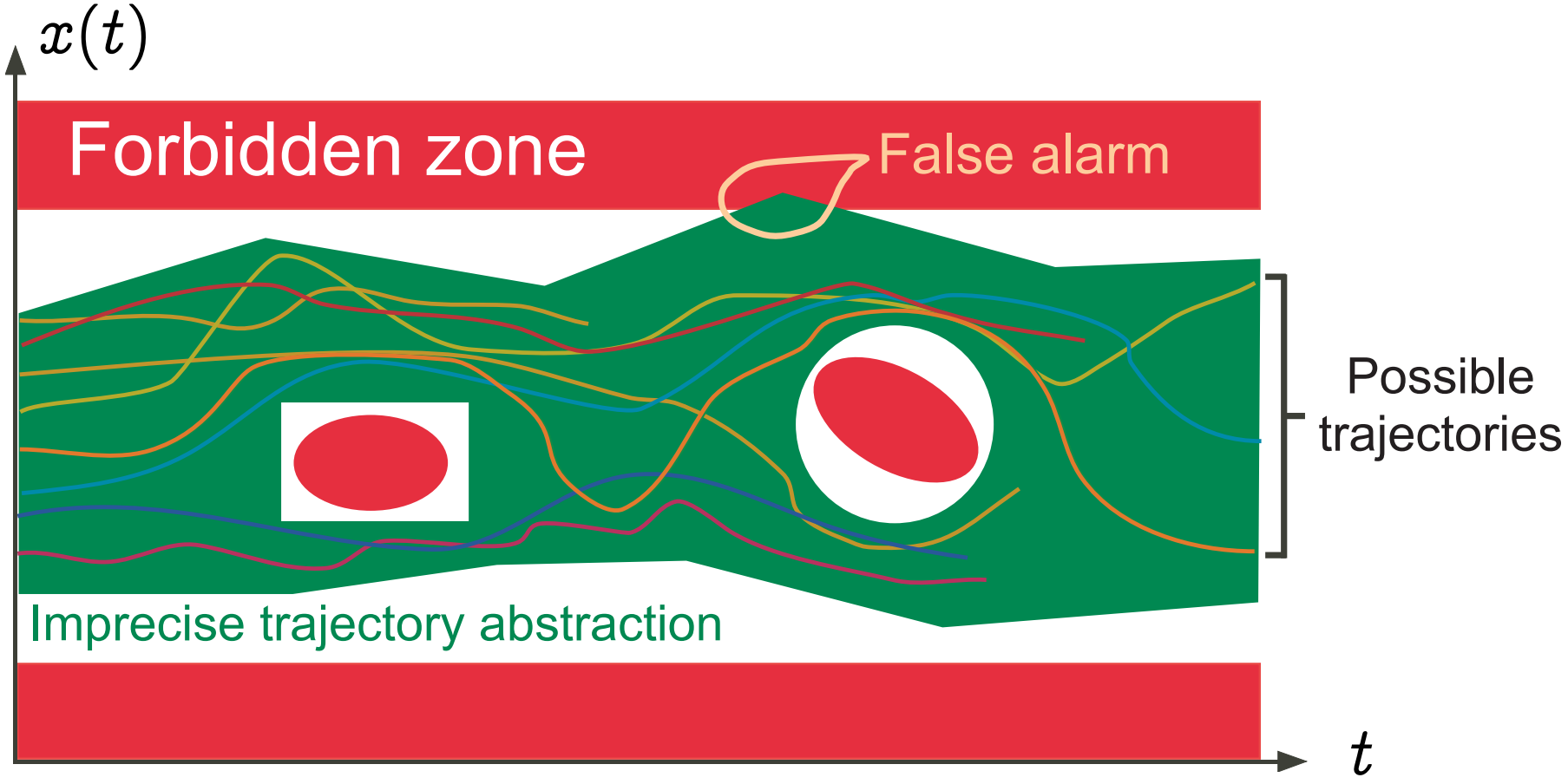


[3] This situation is always excluded in static analysis by abstract interrpetation.

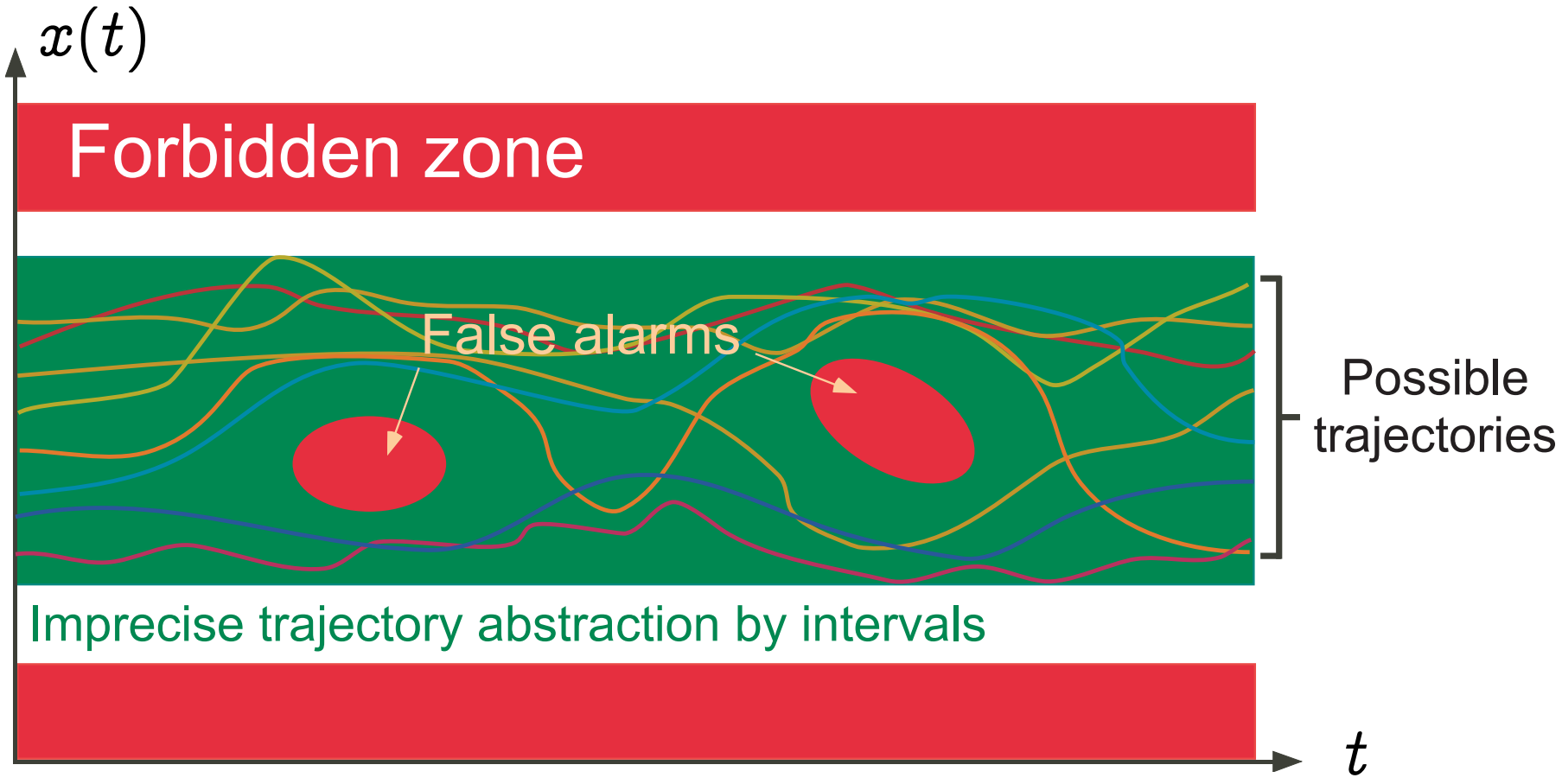
Soundness requirement: erroneous abstraction



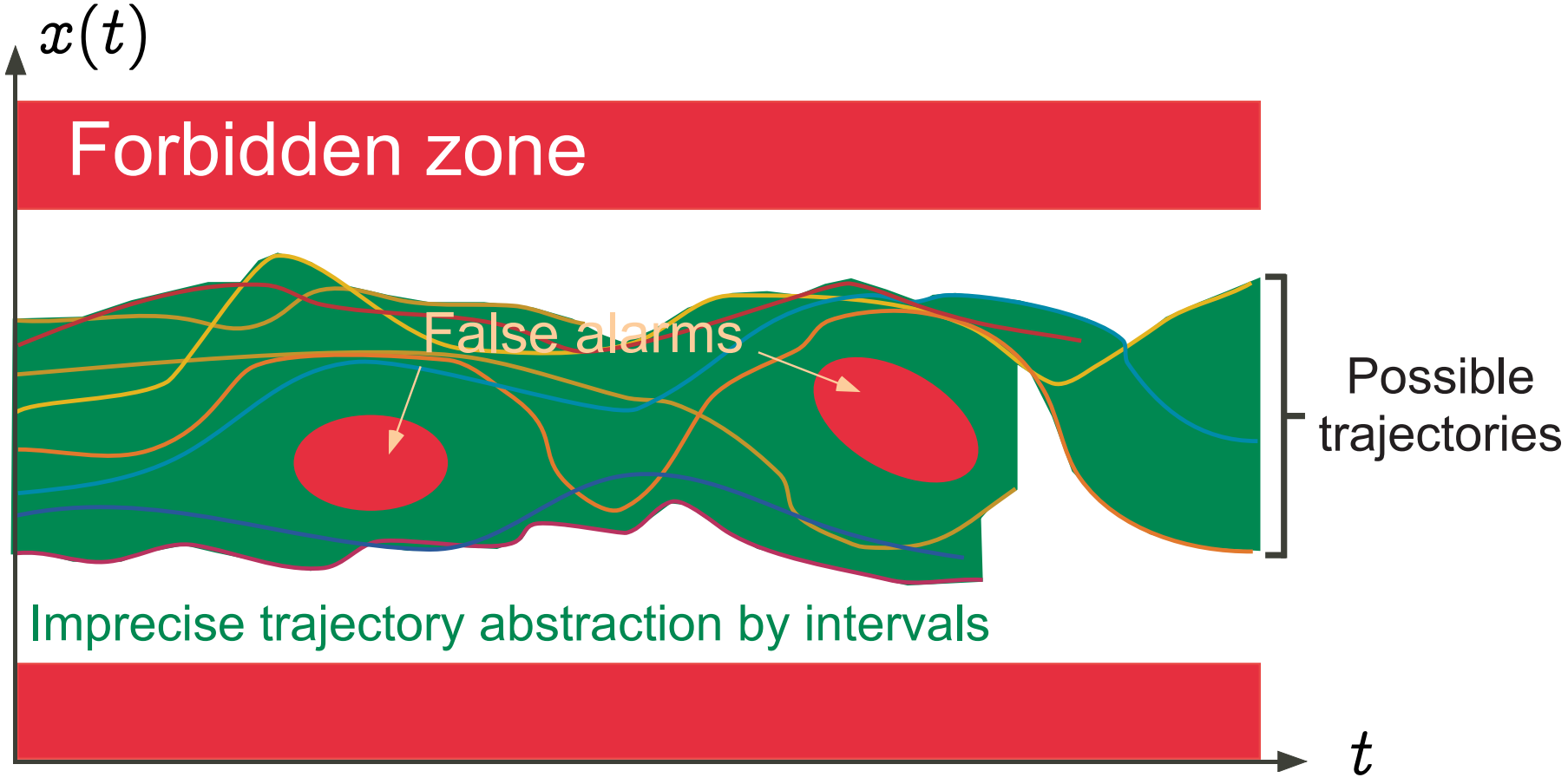
Imprecision $\Rightarrow$ false alarms



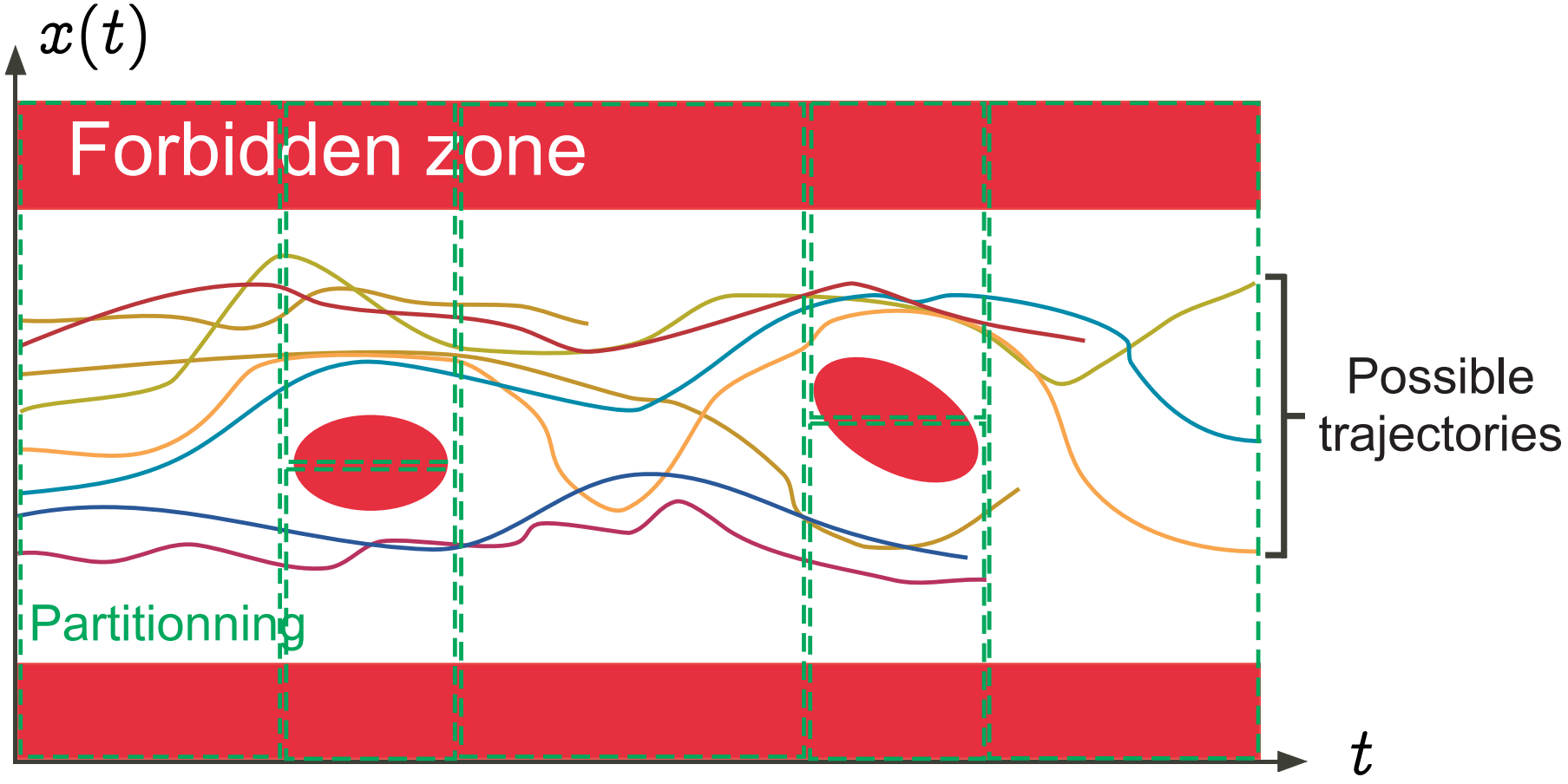
Global interval abstraction $\rightarrow$ false alarms



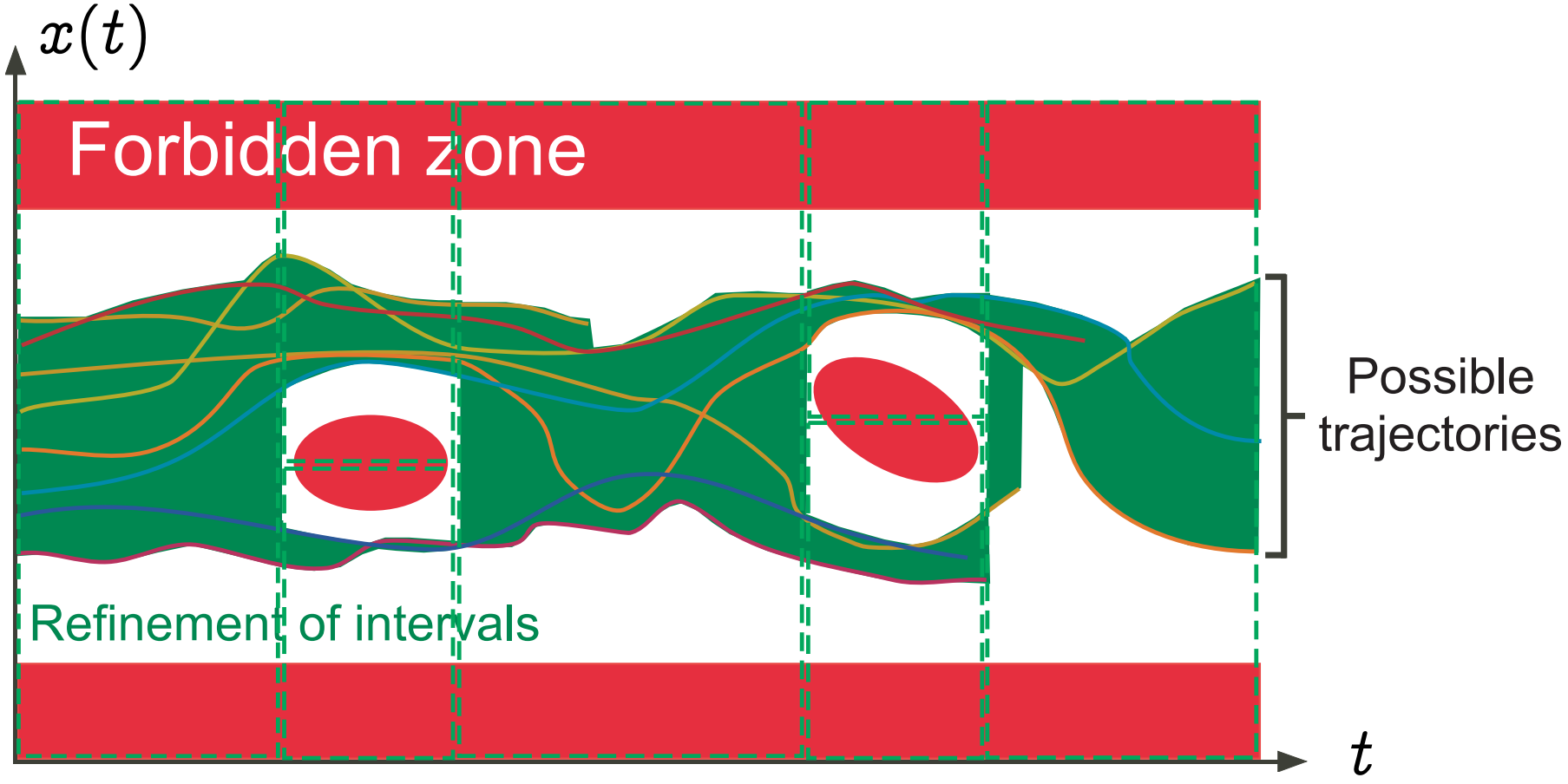
Local interval abstraction $\rightarrow$ false alarms



Refinement by partitionning



Intervals with partitionning



2. Applications of Abstract Interpretation

Applications of Abstract Interpretation

- **Static Program Analysis** [CC77], [CH78], [CC79] including
  - **Dataflow Analysis** [CC79], [CC00],
  - **Set-based Analysis** [CC95],
  - **Predicate Abstraction** [Cou03],
  - ...
- **Grammar Analysis and Parsing** [CC03], [CC06]

Applications of Abstract Interpretation (Cont'd)

- **Hierarchies of Semantics (including Proofs)** [CC92], [Cou02]
- **Typing & Type Inference** [CC97]
- **(Abstract) Model Checking** [CC00]
- **Bisimulations** [RT04]
- **Non-interference** [GM04]
- **Models of Security Protocols** [Bla05]

Applications of Abstract Interpretation (Cont'd)

- **Program Transformation** [CC02] including
  - **Software Watermarking** [CC04]
  - **(Semantic/Abstract) Slicing** [Riv05]
  - **Code Obfuscation** [PG05a, PG05b]
- **Malware Detection** [PCJD07]
- **Computational biology** [Dan07]
- **Quantum computing** [JP06, Per06]
- ...

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

3. The ASTRÉE static analyzer

Project Members

Bruno Blanchet [1]

Patrick Cousot

Radhia Cousot

Jérôme Feret

Laurent Mauborgne

Antoine Miné

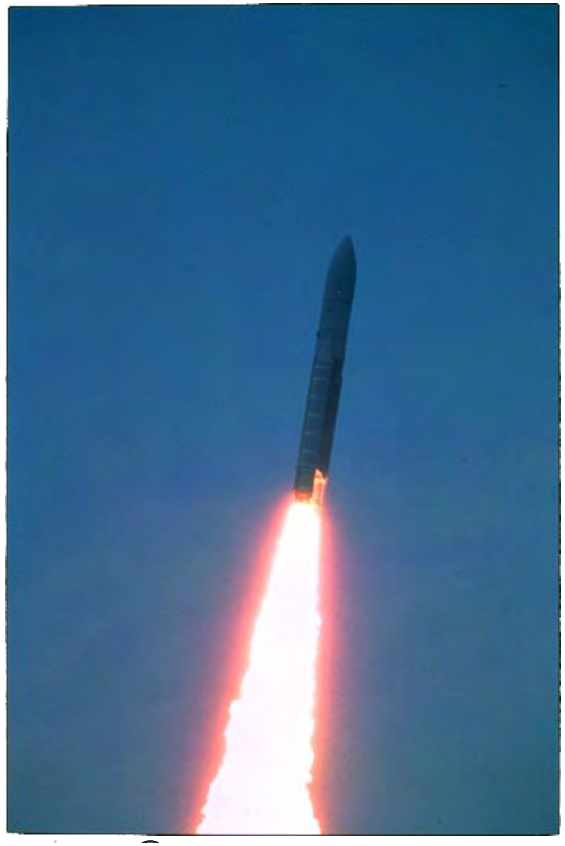
David Monniaux

Xavier Rival

[1] Nov. 2001 — Nov. 2003.

How we got started

A dramatic realization



Software safety is important

June 4, 1996: Ariane 5 explodes on its maiden flight due to a bug

Bug not detected by testing

Renewed interest in formal methods

PolySpace.com produces ADA verifier (now other tools available)

Other industries become interested in static analysis.

Airbus and Astréeproject

Airbus has producted digital fly-by-wire controls since A320 (1988).

Avionics division interested in formal methods.

Uses Hoare logic theorem prover (Caveat, from CEA), and static analyzers: timing validation + stack usage, by Absint GmbH
Astréeby ENS/CNRS (started in 2001)

Introduction

The Astrée static analyzer

Astrée `http://www.astree.ens.fr` is a static analyzer based on abstract interpretation.

- Analyzes a subset of the C language.
- Machine integers and floating-point numbers, not "mathematical" integers and real numbers.
- Tuned for large-scale control/command codes, automatically generated from high-level specifications.
- Precise domains for numerical computations.
- Detects runtime errors.

Challenges

Has to analyze the original source code, not a derived "model".

Has to be sound (i.e. not say "there is no runtime error possible" when there are)

Has to be precise (i.e. not warn about many *possible* alarms that can't happen — false alarms)

Handle floating-point well, including digital filtering algorithms

The biggest challenge

Very large software ($\gg$ 700,000 LOC) $\Rightarrow$ efficiency questions !

Commodity PC hardware $\Rightarrow$ keep memory requirements low

$\Rightarrow$ keep analysis times low

Efficiency considerations

False "good idea" For final "certification" of the system, only need a single pass of analysis, even if it is slow.

In reality... You want fast analysis

- for debugging the analyzer
- for using it while you develop the analyzed code
- for debugging input specifications (i.e. bounds on the inputs)

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

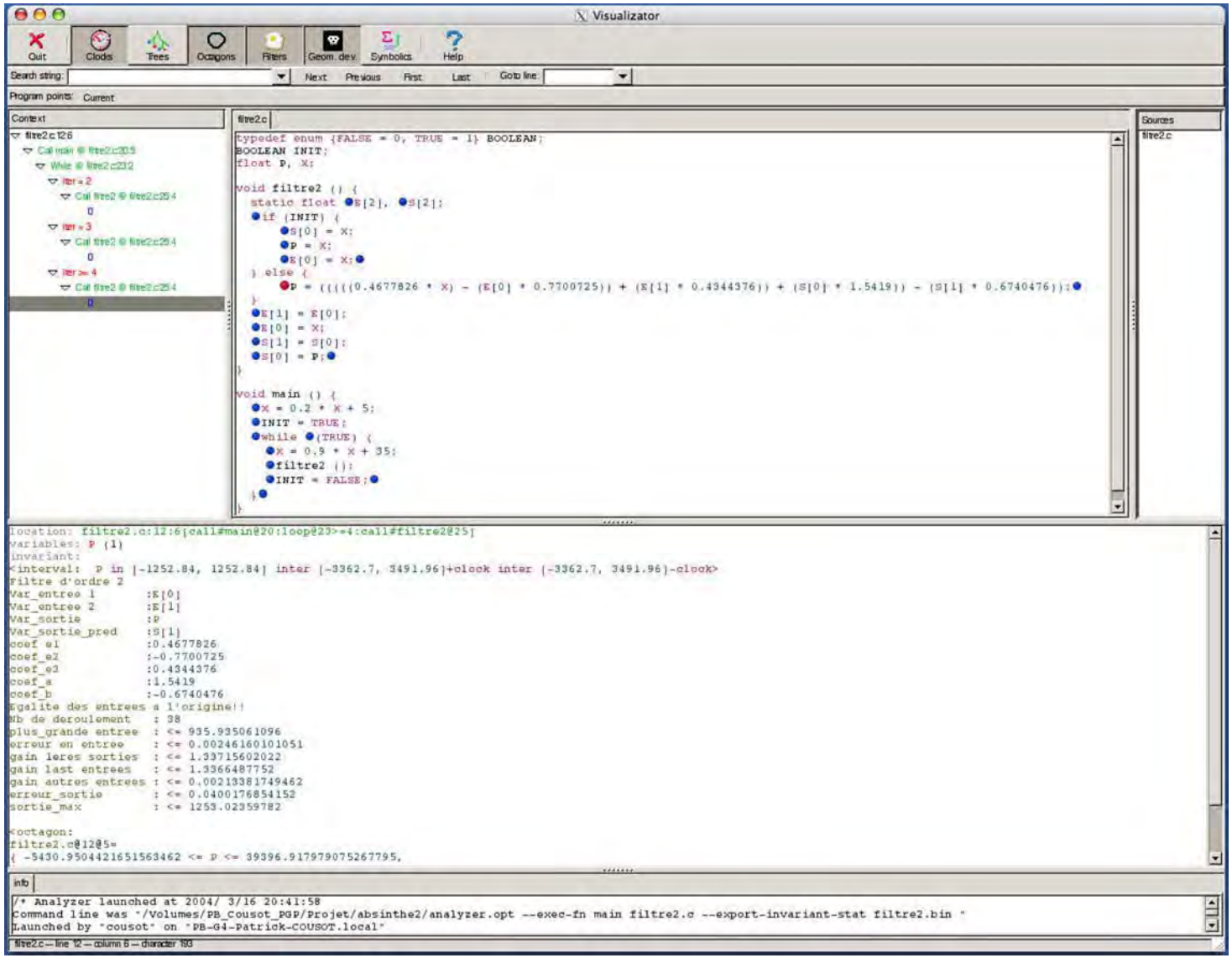
Characteristics of the ASTRÉE Analyzer (Cont'd)

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 OCAML 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



Initial domain of application of ASTRÉE

Electric Flight Control Command

- Before: assistance to the pilot
  - Boeing 747
  - Airbus A300/310
- Now: control of the plane (and the pilot :-))
  - Airbus A320 / A330 / A340
  - Boeing 777
  - Airbus A380

Manual (assisted) flight command

Boeing 747-400

Crédit photo : Adrian Pingstone (domaine public) / Wikimedia Commons

Manual (assisted) flight command

Boeing 747

Crédit photo : Snowdog / domaine public / Wikimedia Commons

Manual (assisted) flight command

Airbus A300 / A310

End of the 70s – Beginng of the 80s

Crédit photo : Airbus

Manual (assisted) flight command

The pilot directly controls the active surfaces via mechanical links (with computer assistance)

Digital fly-by-wire avionics

Airbus A330

Crédit photo : Airbus

Digital fly-by-wire avionics

The *sidestick*

The pilot operates on a command linked to a computer which activates the active surfaces via electric transmissions

No mechanical link (even in case of malfunctioning)

Digital fly-by-wire avionics

(Simulating manual commands)

Boeing 777

Crédit photo : Boeing [photograph used under the fair use doctrine of US law]

Digital fly-by-wire avionics

Pilot sends orders to computer via sidestick (Airbus) or "traditional looking" control yoke (Boeing)

Computer sends order to surfaces, receives feedback from sensors

Allows simplified piloting (computer does all compensations internally)

Not a single computer (multiple redundant computers, cross-checking, separate architectures, multi-level redundancy)

Airbus A380

Crédit photo : David Monniaux (CC-BY-SA), Wikimedia Commons

Considered Programs and Semantics

Which Programs are Considered ?

- Embedded avionic programs;
- Automatically generated from a proprietary graphical system control language (à la Simulink);
- Synchronous real-time critical programs:

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

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`)

- with (cont'd)
  - `union`
  - pointer arithmetics
- without
  - dynamic memory allocation
  - recursive function calls
  - backward branching
  - conflicting side effects
  - C libraries, system calls (parallelism)

Main Characteristics of the Programs

Difficulties:

- **Many global variables and arrays** ($> 5\,0\,000$);
- A **huge loop** ($> 700\,000$ lines after simplification);
- Each iteration depends on the state of the previous iterations (state variables);
- **Floating-point** computations
  (**80% of the code** implements non-linear control with feed-back);
- Everything is **interdependent** (live variables analysis, slicing ineffective);
- Abstraction by elimination of any variable is too imprecise.

Simplicities:

- All data is **statically allocated**;
- Pointers are restricted to call-by-reference, **no pointer arithmetics**;
- Structured, **recursion-free** control flow.

ASTRÉE has been extended to cope with union and pointer arithmetic to handle other embedded programs, like hand written communication programs.

Semantics

- The standard **ISO C99 semantics**:
  - arrays should not be accessed out of their bounds, ...

  restricted by:

- The **machine semantics:**
  - integer arithmetics is 2's complement,
  - floating point arithmetics is IEEE 754-1985,
  - `int` and `float` are 32-bit, `short` is 16-bit, ...

  restricted by:

- The **user's semantics:**
  - integer arithmetics should not wrap-around,
  - some IEEE exceptions (invalid operation, overflow, division by zero) should not occur, ...

Trace semantics

– From this small-step semantics we deribe a discrete-time complete trace semantics [4];

– This trace semantics is abstracted into many different abstract properties as implemented by various abstract domains;

– ASTRÉE computes a weak reduced product for these abstractions [5].

[4] posibly limited, for synchronous control/command programs, to a maximum number of clock ticks (`__ASTREE_wait_for_clock(())`, as specified by a configuration file.

[5] for efficiency, only a number of useful reductions are performed amongst all possible ones, via communications between abstract domains.

Examples of abstractions

– Set of complete traces $\xrightarrow{\alpha}$ reachable states $\xrightarrow{\alpha}$ [6] possible values of each variable at a given program point $\xrightarrow{\alpha}$ [7] [8] interval of possible values of each variable at a given program point;

– ... idem ... $\xrightarrow{\alpha}$ simple congruence

– Set of complete traces $\xrightarrow{\alpha}$ reachable states $\xrightarrow{\alpha}$ set of vectors of possible values of all variables at a given program point $\xrightarrow{\alpha}$ octagonal relations between pairs of variables at a given program point [9];

[6] cartesian abstraction

[7] interval abstraction

[8] see page 66–67

[9] see page 69–70

Examples of abstractions(Cont'd)

– Set of complete traces $\xrightarrow{\alpha}$ set of reachable partial traces of a loop for 1, 2, ..., n and $> n$ iterations [10] $\xrightarrow{\alpha}$ ... [11]

– Set of complete traces $\xrightarrow{\alpha}$ for each trace, a map of discrete time i in the trace to thevalue $\mathtt{X}_i$ of a variable $\mathtt{X}$ at that time i along that trace $\xrightarrow{\alpha}$ a map of dicrete time i in the traces to the maximum value $\bar{\mathtt{X}}_i$ of a variable $\mathtt{X}$ at that time i in all traces [12]

[10] n is a parameter of ASTRÉE

[11] see pages 108–110

[12] see slides 74 to 95

Goal of the Program Static Analyzer

- **Correctness verification**.

- Nothing can go wrong at execution:
  - no integer overflow or division by zero,
  - no exception, NaN, or $\pm\infty$ generated by IEEE floating-point arithmetics,
  - no out of bounds array access,
  - no erroneous type conversion.

- The execution semantics on the machine **never reaches an indetermination or an error case** in the standard / machine / user semantics.

Different Classes of Run-time Errors

1. Errors terminating the execution [6]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.
2. Errors not terminating the execution with predictable outcome [7]. ASTRÉE warns and continues with worst-case assumptions.
3. Errors not terminating the execution with unpredictable outcome [8]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.

3. is sound with respect to C standard, unsound with respect to C implementation, unless no false alarm.

[6] floating-point exceptions e.g. (invalid operations, overflows, etc.) when traps are activated

[7] e.g. overflows over signed integers resulting in some signed integer.

[8] e.g. memory corruptionss.

Information about the Program Execution Automatically Inferred by the Analyzer

- The analyzer effectively computes a **finitely represented**, **compact** over-approximation of the **immense** reachable state space.
- The information is **valid for any execution** interacting with **any possible environment** (through undetermined volatiles).
- It is inferred **automatically** by abstract interpretation of the collecting semantics and convergence acceleration (∇, Δ).

Automatic Program Verification by Abstract Interpretation

Result:

- Can produce **zero or very few false alarms** while checking **non-trivial properties** (absence of Run-Time Error);
- **Does scale up**.

How ?

- We **specialize** the abstract interpreter for a **family of programs** (which correctness proofs would be similar).
- The abstract domains are **generic** invariants **automatically** instantiated by the analyzer (to make these proofs).

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: $\times$ 5/7

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

$\longrightarrow$ A world première!

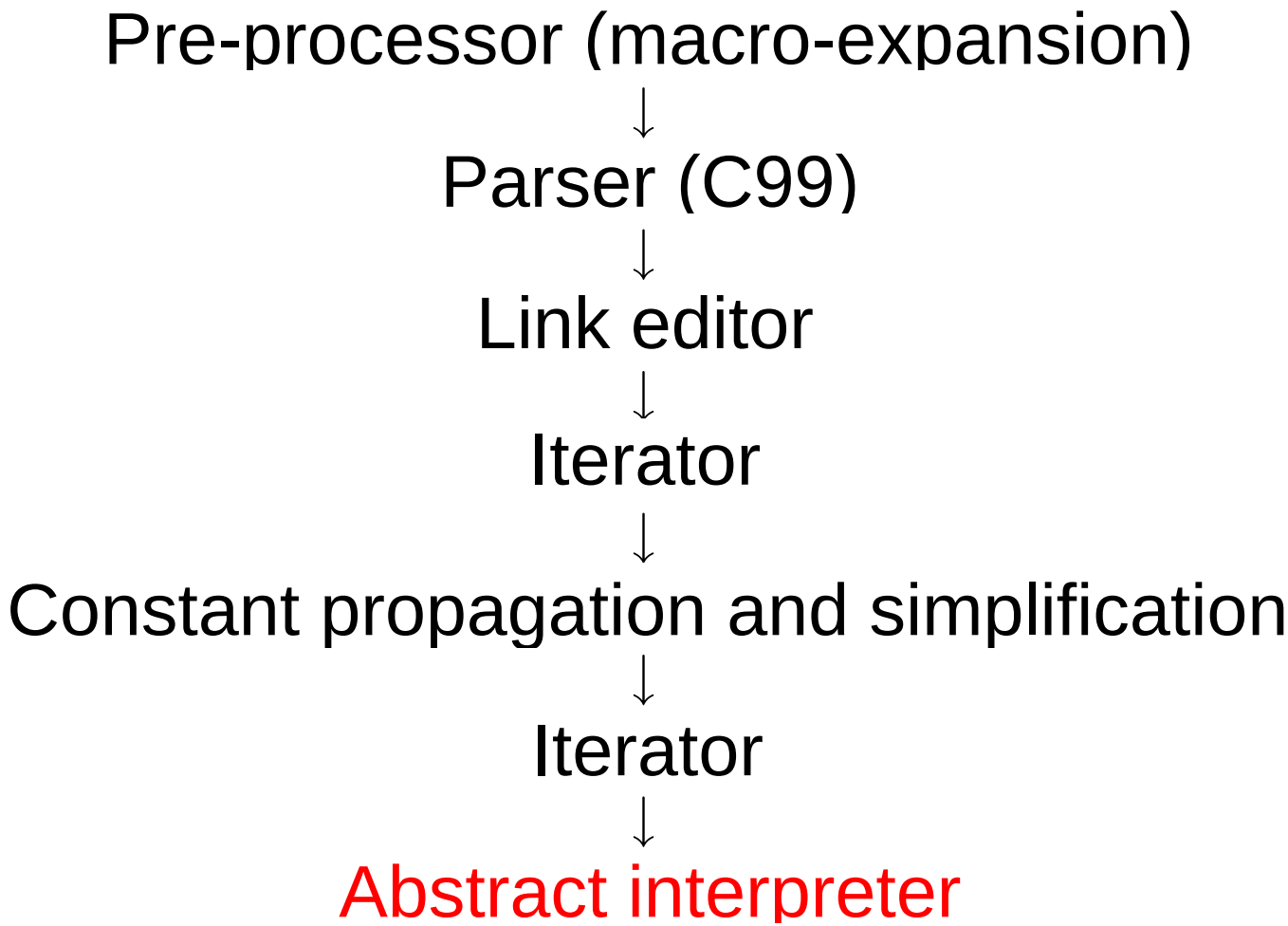
(Airbus A380 Primary Flight Control Software)

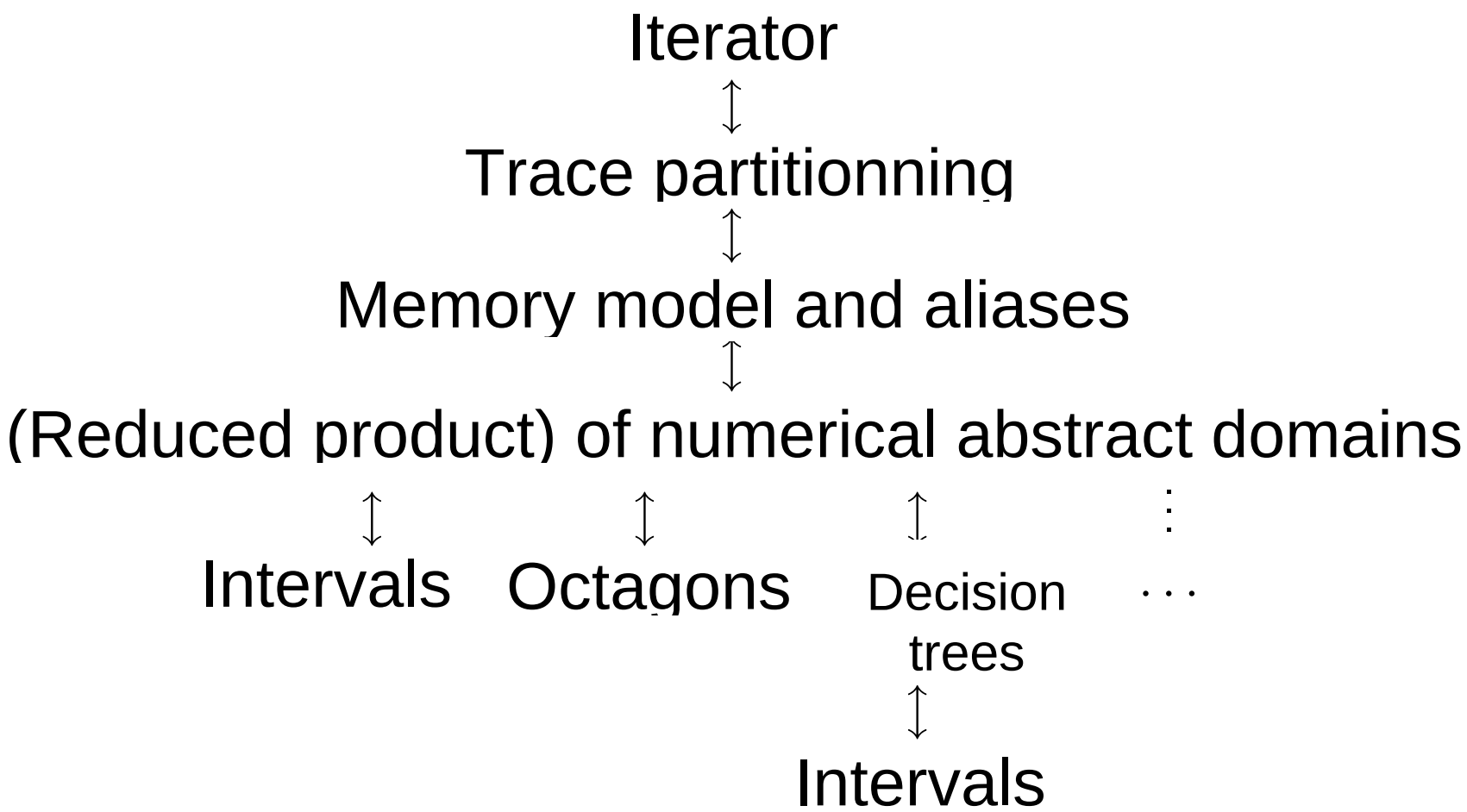
- 350,000 lines
- **0** alarms (Nov. 2004),
  7h [6] on 2.8 GHz PC,
  1 Gigabyte
  $\longrightarrow$ A world grand première!
- Now at 1,000,000 lines!

  34h, 8 Gb, still 0 false alarm
  (after some additional
  parametrisation)

[6] 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.

The abstract interpretor


Pre-processor (macro-expansion)
↓
Parser (C99)
↓
Link editor
↓
Iterator
↓
Constant propagation and simplification
↓
Iterator
↓
Abstract interpreter


Iterator
Trace partitionning
Memory model and aliases
(Reduced product) of numerical abstract domains
Intervals
Octagons
Decision trees
· · ·
Intervals

Abstractions and Abstract Domains

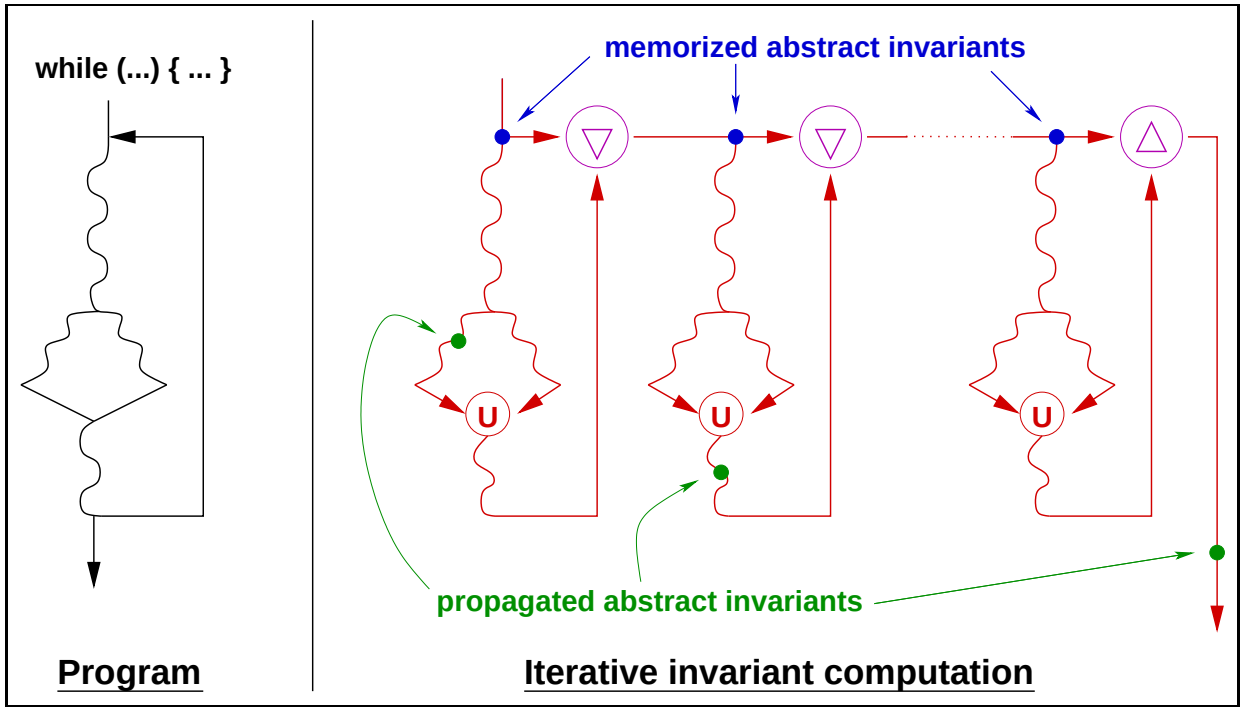
- What ASTRÉE computes:
  Invariant $I \in D^\sharp$: approximation for the set of traces $[\![P]\!]$
- Structure of the abstraction $I \in D^\sharp$:
  - For each control point l
  - For each execution context κ (e.g. calling stack)

  $\Rightarrow$ an approximation $I(l, \kappa) \in D^\sharp_{\mathtt{M}}$ for a set of memory states
- Layout of $D^\sharp_{\mathtt{M}}$:
  - Reduced product of a collection of abstract domains
  - Each domain:
    - Expresses a (generally infinite) family of predicates
    - Transfer functions: $\mathit{assign}, \mathit{guard}$, ... operators
    - Approximations for $\cup$: $\sqcup$ and ∇ (convergence acceleration)

The Abstract Interpreter

Principle: play all executions in a single, abstract computation

- Analysis of a basic statement $x = e$:
  - Transfer function $\mathit{assign}(x = e) : D^\sharp_{\mathtt{M}} \to D^\sharp_{\mathtt{M}}$
  - $D^\sharp_{\mathtt{M}}$: accounts for the new/removed constraints
- Analyzing compound programs, e.g. loops:



Abstract Domains

Choice of the Abstract Domains

Abstract Domain:

- Computer representation of a **class** of **program properties**;
- **Transformers** for propagation through expressions and commands;
- Primitives for **convergence acceleration**: ∇, Δ.

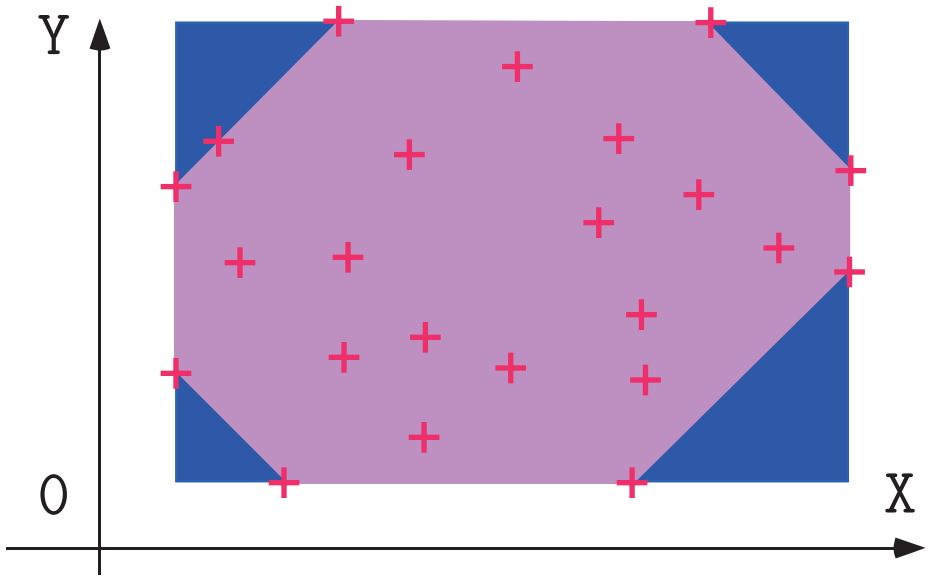
Composition of Abstract Domains:

- Essentially approximate **reduced product** (conjunction with simplification).

Design of Abstract Domains:

- Know-how;
- **Experimentation**.

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]

Interval Abstract Domain

- Classical domain [Cousot Cousot 76];
- Minimum information needed to check the correctness conditions;
- **Not precise enough** to express a useful inductive invariant (thousands of false alarms);
- $\Longrightarrow$ must be refined by:
  - combining with existing domains through reduced product,
  - designing **new domains**, until all false alarms are eliminated.

Clock Abstract Domain

Code Sample:

```
R = 0;
while (1) {
  if (I)
    { R = R+1; }
  else
    { R = 0; }
  T = (R>=n);
  wait_for_clock ();
}
```

This abstract domain need no longer to be used since it is subsumed by the arihmetric-geometric abstract domain (introduced to cope with cumulation of rounding errors)

- Output `T` is true iff the volatile input `I` has been true for the last `n` clock ticks.
- The clock ticks every `s` seconds for at most `h` hours, thus `R` **is bounded**.
- To prove that `R` **cannot overflow**, we must prove that `R` **cannot exceed the elapsed clock ticks** (impossible using only intervals).

Solution:

- We add a phantom variable `clock` in the concrete user semantics to track elapsed clock ticks.
- For each variable `X`, we abstract **three intervals**: `X`, `X+clock`, and `X-clock`.
- If `X+clock` or `X-clock` is bounded, so is `X`.

Octagons

```
assume(x ∈ [−10, 10])
if(x < 0){y = −x;}
else{y = x;}
①if(y ≤ 5)
  {②assert(−5 ≤ x ≤ 5);}
```

- Interval analysis:
  - At ①, $\mathtt{x} \in [-10, 10];\ \mathtt{y} \in [0, 10]$
  - At ②, $\mathtt{x} \in [-10, 10];\ \mathtt{y} \in [0, 5]$
  - Alarm (assert not proved)

- A relation between $\mathtt{x}$ and $\mathtt{y}$ is required:
  $\Rightarrow$ We need a relational abstraction

- Octagons:
  - Express constraints of the form $\pm x \pm y \leq c$.
    Above example:
    - At ①, $0 \leq \mathtt{y} - \mathtt{x} \leq 20;\ 0 \leq \mathtt{y} + \mathtt{x} \leq 20$
    - At ②, $\mathtt{y} \in [0, 5];\ 0 \leq \mathtt{y} - \mathtt{x} \leq 20;\ 0 \leq \mathtt{y} + \mathtt{x} \leq 20,$
      so we derive $\mathtt{x} \in [-5, 5]$
  - Reasonable cost: $\mathcal{O}(n^2)$ memory and $\mathcal{O}(n^3)$ time complexity

Using Octagons

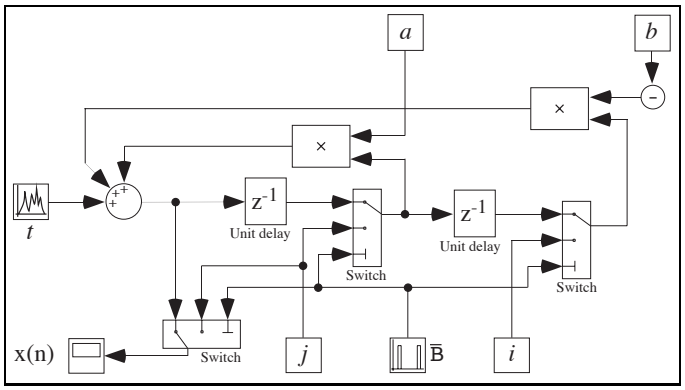
Several issues should be addressed:

- Scalability: $\mathcal{O}(n^3)$ time, $n \equiv 10\,000$ will not scale:
  $\Rightarrow$ Use many small octagons instead of a big one
  - Packs: small group of variables relations are required for
  - Strategy: determines packs, required relations represented
  - Complexity: linear in the number of packs
    Size of packs: bounded by a constant
    Number of packs: linear in the size of the code
    $\Rightarrow$ Linear complexity
- Floating point rounding errors in the concrete computations solved by linearization of expressions:
  - Expressions approximated with real interval linear forms
  - Relational domain: semantics in terms of real numbers
  - Rounding errors in concrete computations accounted for at linearization time

Application-specific abstract domains (for control-command)

Ellipsoid Abstract Domain for Filters

2^{d} Order Digital Filter:

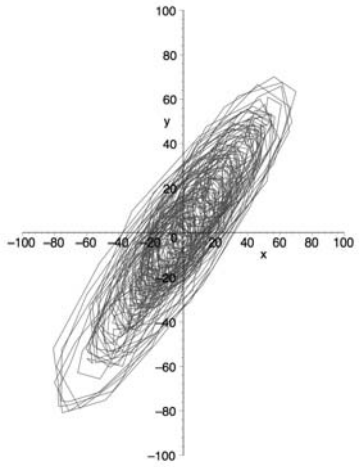


– Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$

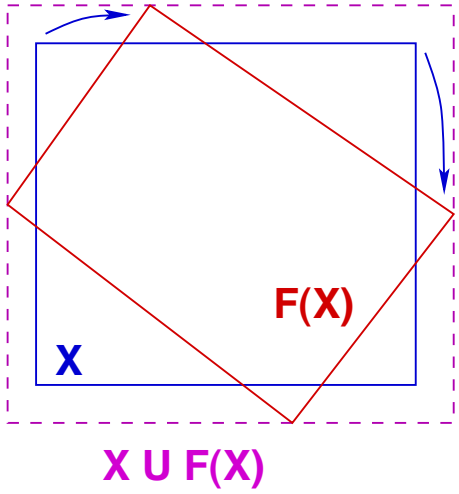
– The concrete computation is bounded, which must be proved in the abstract.

– There is no stable interval or octagon.

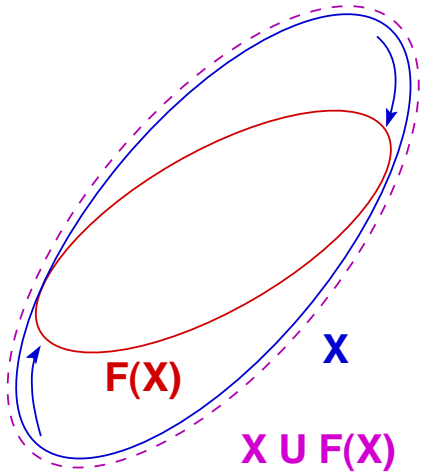
– The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

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 [7] [8]

– Abstract domain: $(\mathbb{R}^+)^5$

– Concretization:

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

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

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

[7] here in $\mathbb{R}$

Applications

Arithmetic-geometric progressions provide bounds for :

1. division by α followed by a multiplication by α:

   $\Longrightarrow$ our running example;

2. barycentric means:

   $\Longrightarrow$ at each loop iteration, the value of a variable X is computed as a barycentric mean of some previous values of X

   (not necessarily the last values);

3. bounded incremented variables:

   $\Longrightarrow$ it replaces the former domain that bounds the difference and the sum between each variable and the loop counter.

Arithmetic-geometric progressions (Example 1)

Running example (in $\mathbb{R}$)

```
1 : X := 0; k := 0;
2 : while (k < 1000) {
3 :     if (?) {X ∈ [−10; 10]};
4 :     X := X/3;
5 :     X := 3 × X;
6 :     k := k + 1;
7 :     }
```

Interval analysis: first loop iteration

$1:\ X := 0; k := 0;$ $\quad X = 0$

$2:\ \text{while}\ (k < 1000)\ \{$ $\quad X = 0$

$3:\quad \text{if}\ (?)\ \{X \in [-10; 10]\};$ $\quad |X| \leq 10$

$4:\quad X := X/3;$ $\quad |X| \leq \frac{10}{3}$

$5:\quad X := 3 \times X;$ $\quad |X| \leq 10$

$6:\quad k := k + 1;$

$7:\quad \}$

Interval analysis: Invariant

$1:\ X := 0; k := 0;$ $\quad X = 0$

$2:\ \textbf{while}\ (k < 1000)\ \{$ $\quad |X| \leq 10$

$3:\quad \textbf{if}\ (?)\ \{X \in [-10; 10]\};$ $\quad |X| \leq 10$

$4:\quad X := X/3;$ $\quad |X| \leq \frac{10}{3}$

$5:\quad X := 3 \times X;$ $\quad |X| \leq 10$

$6:\quad k := k + 1;$

$7:\quad \}$ $\quad |X| \leq 10$

Including rounding errors [Miné–ESOP'04]

$$
\begin{array}{ll}
1: & X := 0; k := 0; \\
2: & \text{while } (k < 1000)\ \{ \\
3: & \quad \text{if } (?)\ \{X \in [-10; 10]\}; \\
4: & \quad X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2]; \\
5: & \quad X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4]; \\
6: & \quad k := k + 1; \\
7: & \quad \}
\end{array}
$$

The constants ε_1, ε_2, ε_3, and ε_4 (≥ 0) are computed by other domains.

Interval analysis

Let $M \geq 0$ be a bound:

$1: \ X := 0; k := 0;$ $\qquad X = 0$

$2: \ \text{while } (k < 1000) \{$ $\qquad |X| \leq M$

$3: \quad \text{if } (?) \{X \in [-10; 10]\};$ $\qquad |X| \leq max(M, 10)$

$4: \quad X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$ $\qquad |X| \leq (\varepsilon_1 + \frac{1}{3}) \times max(M, 10) + \varepsilon_2$

$5: \quad X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$ $\qquad |X| \leq (1 + a) \times max(M, 10) + b$

$6: \quad k := k + 1;$

$7: \quad \}$

with $a = 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_3$ and $b = \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4$.

Ari.-geo. analysis: first iteration

$1:\ X := 0; k := 0;$

$X = 0,\ k = 0$

$2:\ \text{while}\ (k < 1000)\ \{$

$X = 0$

$3:\quad \text{if}\ (?)\ \{X \in [-10; 10]\};$

$|X| \leq 10$

$4:\quad X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$

$|X| \leq \left[v \mapsto \left(\frac{1}{3} + \varepsilon_1\right) \times v + \varepsilon_2\right](10)$

$5:\quad X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$

$|X| \leq f^{(1)}(10)$

$6:\quad k := k + 1;$

$|X| \leq f^{(k)}(10),\ k = 1$

$7:\quad \}$

with $f = \left[v \mapsto \left(1 + 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_3\right) \times v + \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4\right]$.

Ari.-geo. analysis: Invariant

$1:\ X := 0; k := 0;$ $\qquad X = 0,\ k = 0$

$2:\ \text{while } (k < 1000)\ \{$ $\qquad |X| \leq f^{(k)}(10)$

$3:\quad \text{if } (?)\ \{X \in [-10; 10]\};$ $\qquad |X| \leq f^{(k)}(10)$

$4:\quad X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$ $\qquad |X| \leq (\frac{1}{3} + \varepsilon_1) \times \left(f^{(k)}(10)\right) + \varepsilon_2$

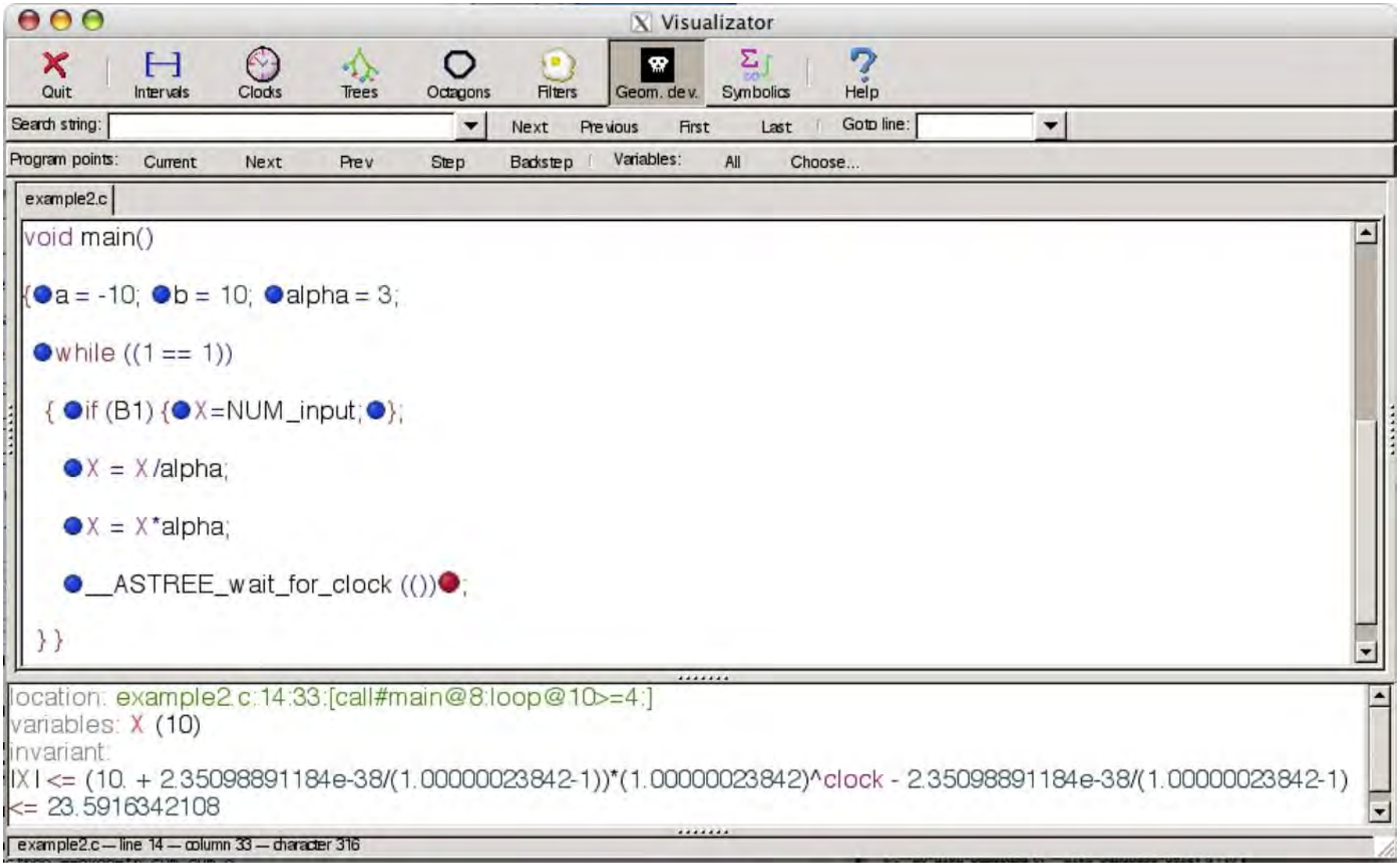
$5:\quad X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$ $\qquad |X| \leq f\left(f^{(k)}(10)\right)$

$6:\quad k := k + 1;$ $\qquad |X| \leq f^{(k)}(10)$

$7:\quad \}$ $\qquad |X| \leq f^{(1000)}(10)$

with $f = \left[v \mapsto \left(1 + 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_3\right) \times v + \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4\right]$.

Analysis session



Arithmetic-Geometric Progressions (Example 2)

```
% 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.
```


Arithmetic-geometric progressions (Example 3)

```
% 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
```


Benchmarks

We analyze three programs in the same family on a AMD Opteron 248, 8 Gb of RAM (analyses use only 2 Gb of RAM).

lines of C	70,000			216,000			379,000		
global variables	13,400			7,500			9,000		
iterations	80	63	**37**	229	223	**53**	253	286	**74**
time/iteration	1mn14s	1mn21s	**1mn16s**	4mn04s	5mn13s	**4mn40s**	7mn33s	9mn42s	**8mn17**
analysis time	2h18mn	2h05mn	**47mn**	15h34mn	19h24mn	**4h08mn**	31h53mn	43h51mn	**10h14m**
false alarms	625	24	**0**	769	64	**0**	1482	188	**0**

1. without using computation time; [no wait_for_clock(())]
2. with the former loop counter domain,
   (without the arithmetic-geometric domain);
3. with the arithmetic-geometric domain,
   (without the former loop counter domain).

Arithmetic-geometric progressions (in $\mathbb{R}$)

An arithmetic-geometric progression is a 5-tuple in $(\mathbb{R}^+)^5$.
An arithmetic-geometric progression denotes a function in $\mathbb{N} \to \mathbb{R}^+$:

$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) \stackrel{\Delta}{=} \left[\nu \mapsto a \times \nu + b\right]\left(\left[\nu \mapsto a' \times \nu + b'\right]^{(k)}(M)\right)$$

Thus,

- k is the loop counter;
- M is an initial value;
- $\left[\nu \mapsto a \times \nu + b\right]$ describes the current iteration;
- $\left[\nu \mapsto a' \times \nu + b'\right]^{(k)}$ describes the first k iterations.

A concretization $\gamma_{\mathbb{R}}$ maps each element $d \in (\mathbb{R}^+)^5$ to a set $\gamma_{\mathbb{R}}(d) \subseteq (\mathbb{N} \to \mathbb{R}^+)$ defined as:

$$\{f \mid \forall k \in \mathbb{N},\ |f(k)| \leq \beta_{\mathbb{R}}(d)(k)\}$$

Monotonicity

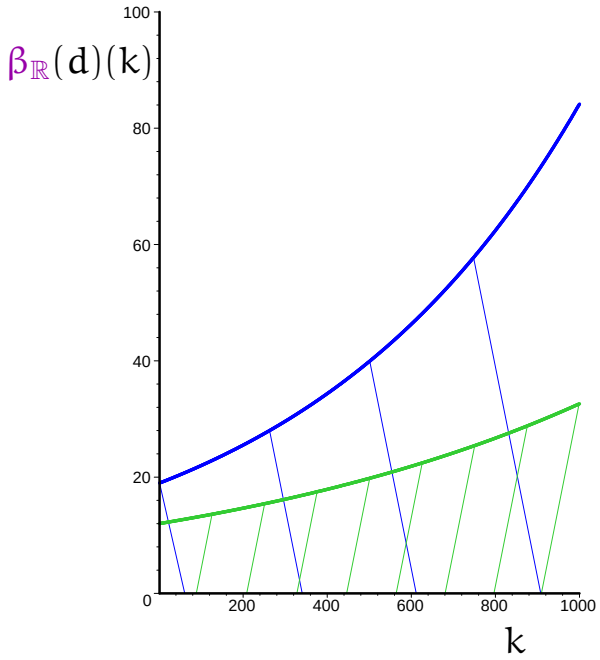
Let $d = (M, a, b, a', b')$ and $d = (M, a, b, a', b')$ be two arithmetic-geometric progressions.

If:

- $M \leq M$,
- $a \leq a$, $a' \leq a'$,
- $b \leq b$, $b' \leq b'$.

Then:

$$\forall k \in \mathbb{N},\ \beta_{\mathbb{R}}(d)(k) \leq \beta_{\mathbb{R}}(d)(k).$$



Disjunction

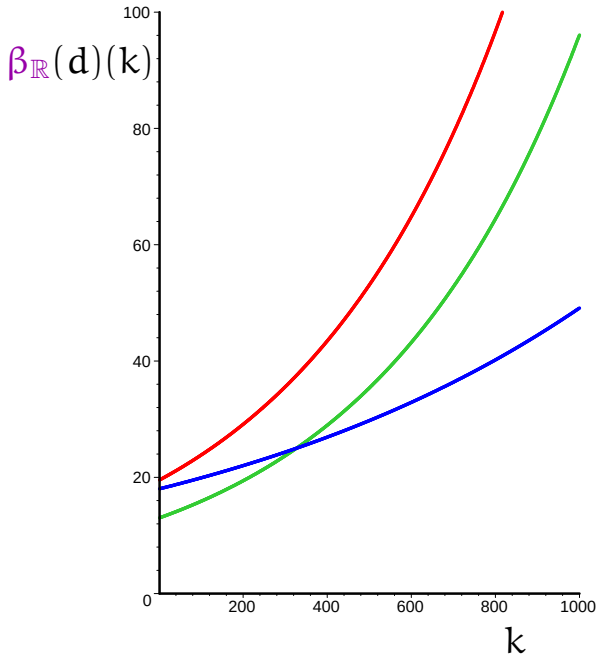
Let $d = (M, a, b, a', b')$ and $d = (M, a, b, a', b')$ be two arithmetic-geometric progressions.

We define:

$$d \sqcup_{\mathbb{R}} d \stackrel{\Delta}{=} (M, a, b, a', b')$$

where:

- $M \stackrel{\Delta}{=} max(M, M)$,
- $a \stackrel{\Delta}{=} max(a, a)$, $a' \stackrel{\Delta}{=} max(a', a')$,
- $b \stackrel{\Delta}{=} max(b, b)$, $b' \stackrel{\Delta}{=} max(b', b')$,



For any $k \in \mathbb{N}$, $\beta_{\mathbb{R}}(d \sqcup_{\mathbb{R}} d)(k) \geq max(\beta_{\mathbb{R}}(d)(k), \beta_{\mathbb{R}}(d)(k))$.

Conjunction

Let d and d be two arithmetic-geometric progressions.

1. If d and d are comparable (component-wise), we take the smaller one:

$$\mathrm{d} \sqcap_{\mathbb{R}} \mathrm{d} \stackrel{\Delta}{=} \mathit{Inf}_{\leq}^{\cdot}\{\mathrm{d}; \mathrm{d}\}.$$

2. Otherwise, we use a parametric strategy:

$$\mathrm{d} \sqcap_{\mathbb{R}} \mathrm{d} \in \{\mathrm{d}; \mathrm{d}\}.$$

For any $\mathrm{k} \in \mathbb{N}$, $\beta_{\mathbb{R}}(\mathrm{d} \sqcap_{\mathbb{R}} \mathrm{d})(\mathrm{k}) \geq \mathit{min}(\beta_{\mathbb{R}}(\mathrm{d})(\mathrm{k}), \beta_{\mathbb{R}}(\mathrm{d})(\mathrm{k}))$.

Assignment

We have:

$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) = a \times (M + b' \times k) + b \qquad \text{when } a' = 1$$
$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) = a \times \left((a')^k \times \left(M - \frac{b'}{1-a'}\right) + \frac{b'}{1-a'}\right) + b \quad \text{when } a' \neq 1.$$

Thus:

1. for any $a, a', M, b, b', \lambda \in \mathbb{R}^+$,

$$\lambda \times \left(\beta_{\mathbb{R}}\left(M, a, b, a', b'\right)(k)\right) = \beta_{\mathbb{R}}\left(\lambda \times M, a, \lambda \times b, a', \lambda \times b'\right)(k);$$

2. for any $a, a', M, b, b', M, b, b' \in \mathbb{R}^+$, for any $k \in \mathbb{N}$,

$$\beta_{\mathbb{R}}\left(M, a, b, a', b'\right)(k) + \beta_{\mathbb{R}}\left(M, a, b, a', b\right)(k) = \beta_{\mathbb{R}}\left(M + M, a, b + b, a', b' + b'\right)$$

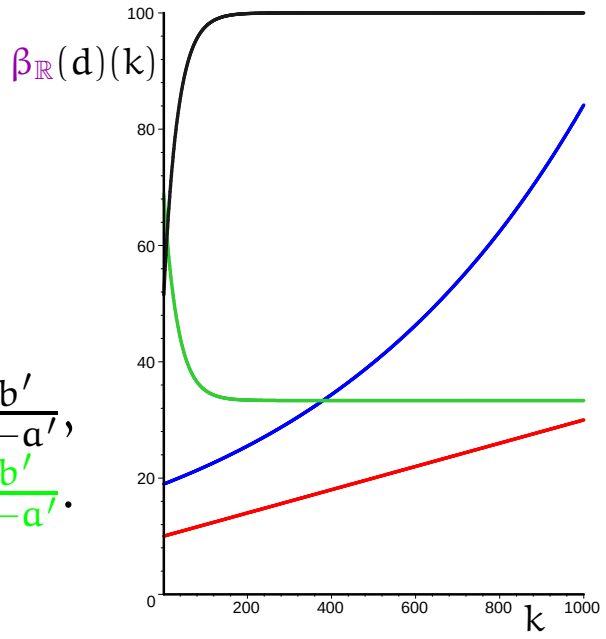
Projection I

$$\beta_{\mathbb{R}}(M,a,b,a',b')(k) = a \times (M + b' \times k) + b \qquad \text{when } a' = 1$$
$$\beta_{\mathbb{R}}(M,a,b,a',b')(k) = a \times \left((a')^k \times \left(M - \frac{b'}{1-a'}\right) + \frac{b'}{1-a'}\right) + b \quad \text{when } a' \neq 1.$$

Thus, for any $d \in (\mathbb{R}^+)^5$,
the function $\left[k \mapsto \beta_{\mathbb{R}}(d)(k)\right]$ is:

- either monotonic,
- or anti-monotonic.

$$\begin{cases} a' > 1, \\ a' = 1, \\ a' < 1 \text{ and } M < \frac{b'}{1-a'}, \\ a' < 1 \text{ and } M > \frac{b'}{1-a'}. \end{cases}$$



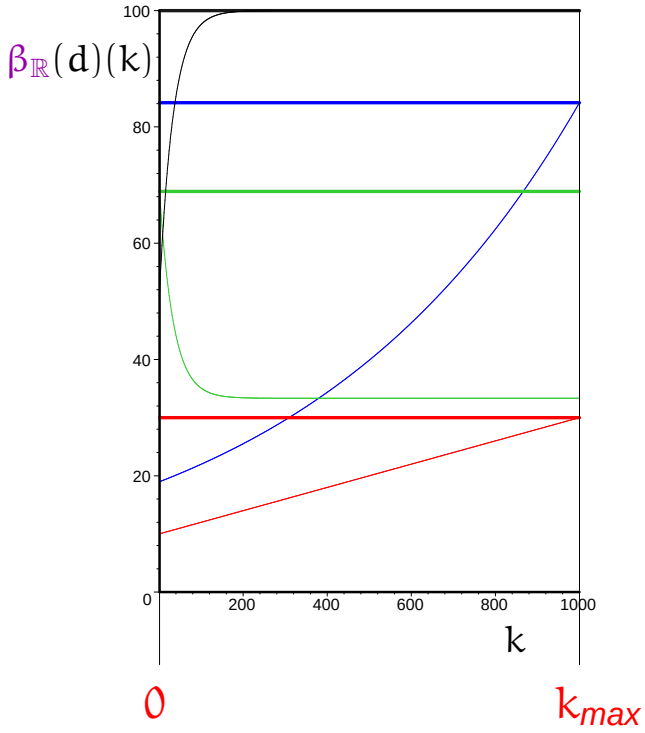
Projection II

Let $d \in (\mathbb{R}^+)^5$ and $k_{max} \in \mathbb{N}$.

$bound(d, k_{max}) \stackrel{\Delta}{=} max(\beta_{\mathbb{R}}(d)(0), \beta_{\mathbb{R}}(d)(k_{max}))$

For any $k \in \mathbb{N}$ such that $0 \leq k \leq k_{max}$:

$$\beta(d)(k) \leq bound(d, k_{max}).$$



Incrementing the loop counter

We integrate the current iteration into the first k iterations:

- the first $k+1$ iterations are chosen as the worst case among the first k iterations and the current iteration;
- the current iteration is reset.

Thus:

$$\mathit{next}_{\mathbb{R}}(M, a, b, a', b') \stackrel{\Delta}{=} (M, 1, 0, \mathit{max}(a, a'), \mathit{max}(b, b')).$$

For any $k \in \mathbb{N}, d \in \left(\mathbb{R}^{+}\right)^{5}$, $\beta_{\mathbb{R}}(d)(k) \leq \beta_{\mathbb{R}}(\mathit{next}_{\mathbb{R}}(d))(k+1)$.

About floating point numbers

Floating point numbers occur:

1. in the concrete semantics:

   Floating point expressions are translated into real expressions with interval coefficients [Miné—ESOP'04].

   In other abstract domains, we handle real numbers.

2. in the abstract domain implementation:

   For efficiency purpose, we implement each primitive in floating point arithmetics: each real is safely approximated by an interval with floating point number bounds.

Handling floating point computations (in the program semantics and in the analyzer)

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
```

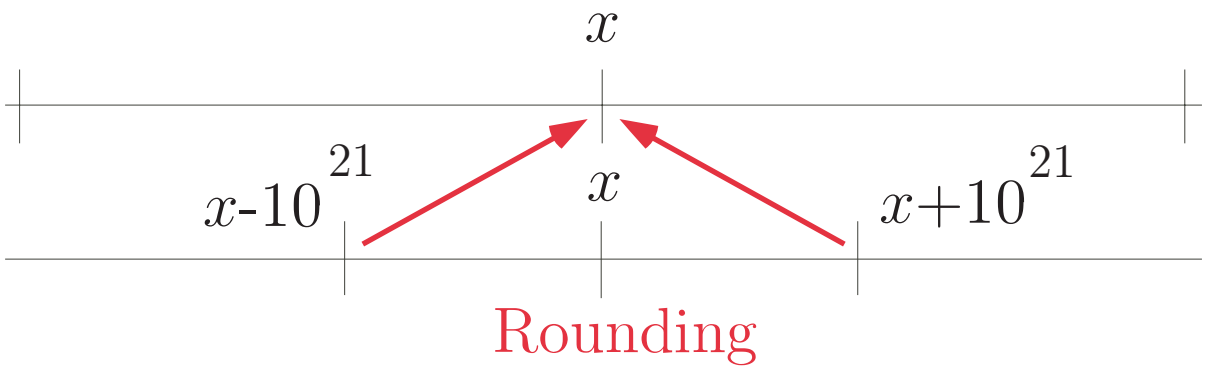
```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951487.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
0.000000
```

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

Explanation of the huge rounding error

(1) Floats

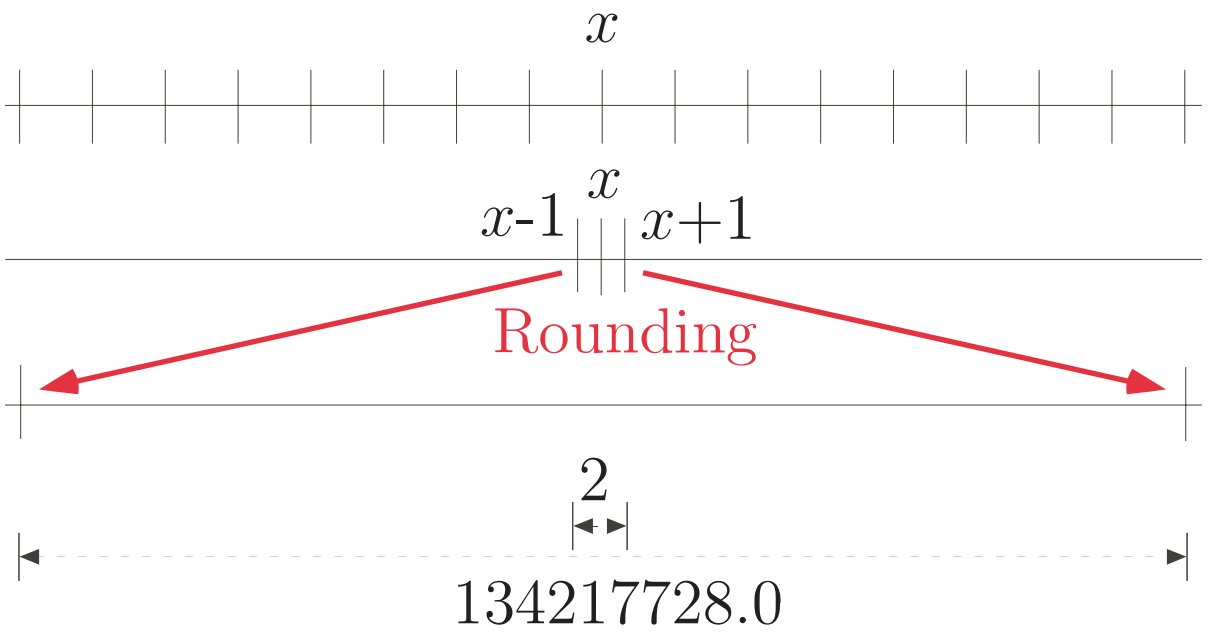
Reals



(2) Doubles

Reals

Floats



Relational Domains on Floating-Point

Problems:

- Relational numerical abstract domains rely on a **perfect mathematical concrete semantics** (in $\mathbb{R}$ or $\mathbb{Q}$).
- Perfect arithmetics in $\mathbb{R}$ or $\mathbb{Q}$ is costly.
- IEEE 754-1985 floating-point concrete semantics **incurs rounding**.

Solution:

- Build an **abstract mathematical semantics in** $\mathbb{R}$ that over-approximates the concrete floating-point semantics, including rounding.
- Implement the abstract domains on $\mathbb{R}$ **using floating-point numbers** rounded in a sound way.

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

– Allows simplification even in the interval domain
if `X` $\in$ [-1,1], we get $|\mathtt{Z}| \leq 0.750\cdots$ instead of $|\mathtt{Z}| \leq 1.25\cdots$

– Allows using a relational abstract domain (octagons)

– Example of good compromize 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
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)>
```


Constructors of Abstract Domains

Abstract domain constructors

– Non-relational lifting: non-relational abstract domain D on values $\mapsto\!\!\circ\!\!\to$ abstract domain $\prod_{X \in \mathsf{Var}} D$ on all variables (using balanced binary trees)

– Relational lifting: relational abstract domain $D(P)$ on packs $P \subseteq \mathsf{Var}$ of variables $\mapsto\!\!\circ\!\!\to$ abstract domain $D(\mathsf{Var})$ on all variables

– Trace partitionning: past history abstraction domain [9] $\times$ current memory state abstraction domain $\mapsto\!\!\circ\!\!\to$ prefix traces abstraction domain (using maps implemented as trees) [11]

– Boolean partitionning: prefix traces abstraction domain $\times$ boolean variables $\mapsto\!\!\circ\!\!\to$ prefix traces abstraction domain (using decision trees)

[9] (e.g. a sub-sequence of branches taken

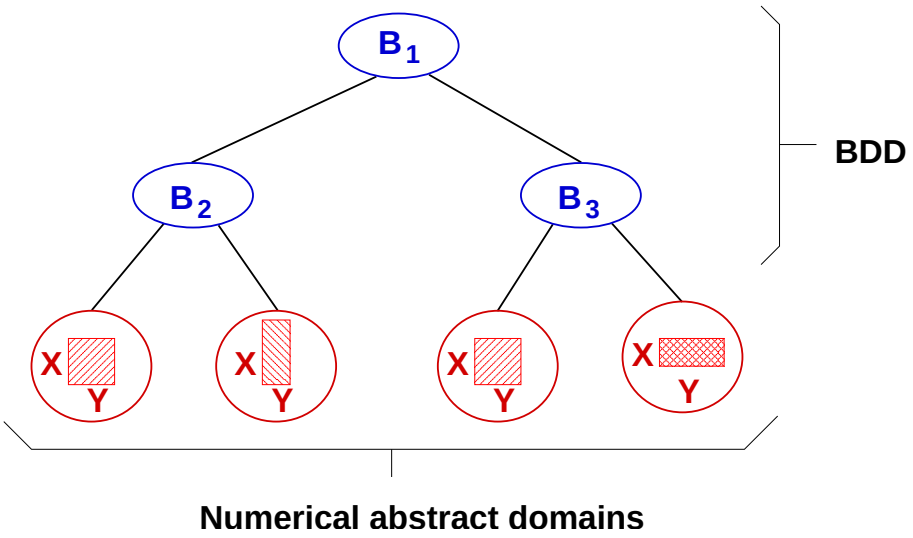
Decision Tree Abstract Domain

Synchronous reactive programs **encode control flow in boolean variables**.

Code Sample:

```
bool B1,B2,B3;
float N,X,Y;
N = f(B1);
if (B1)
  { X = g(N); }
else
  { Y = h(N); }
```

Decision Tree:



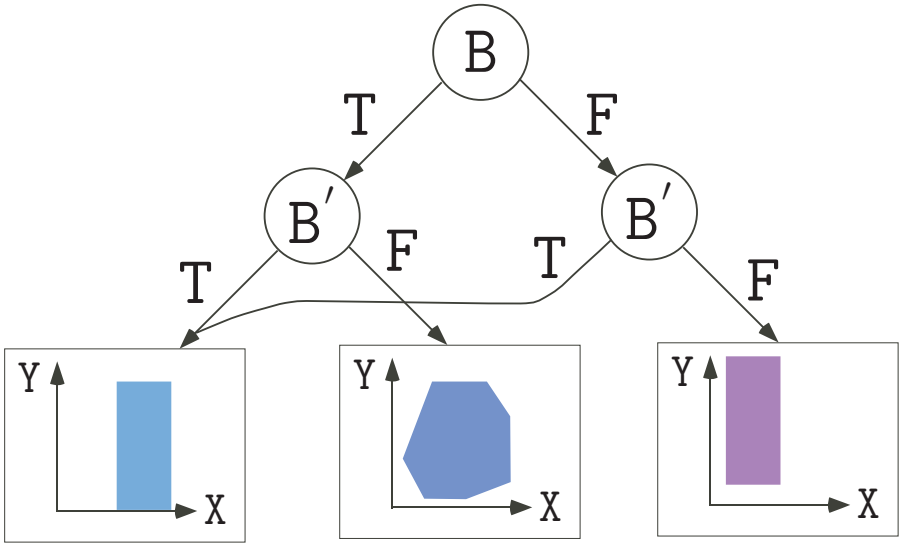
There are too many booleans (**4 000**) to build one big tree so we:

- limit the **BDD height** to 3 (analysis parameter);
- use a **syntactic criterion** to select variables in the BDD and the numerical parts.

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

Iteration Strategies for Fixpoint Approximation

Iteration Refinement: Loop Unrolling

Principle:

- Semantically equivalent to:

```
while (B) { C }   ⟹   if (B) { C }; while (B) { C }
```

- More precise in the abstract:
  - **less** concrete execution paths are **merged** in the abstract.

Application:

- Isolate the **initialization phase** in a loop (e.g. first iteration).

Iteration Refinement: Trace Partitioning

Principle:

- Semantically equivalent to:

```
if (B) { C1 } else { C2 }; C3
```

$$\Downarrow$$

```
if (B) { C1; C3 } else { C2; C3 };
```

- More precise in the abstract:
  - concrete execution paths are **merged later**.

Application:

```
if (B)
  { X=0; Y=1; }
else
  { X=1; Y=0; }
R = 1 / (X-Y);
```

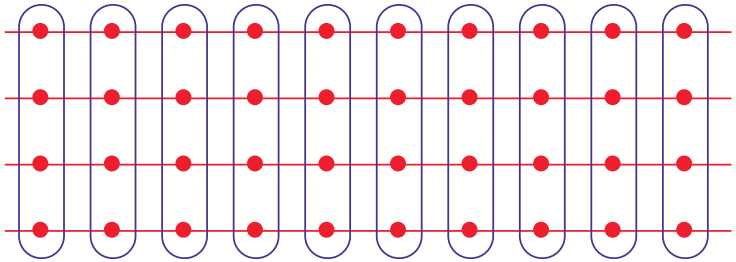
/ cannot result in a division by zero

Control Partitionning for Case Analysis

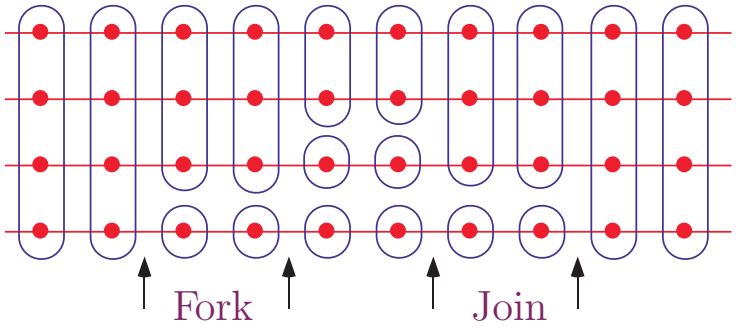
– Code Sample:

```
/* 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 ≤ x ≤ 100 ...

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

Control point partitionning:



Trace partitionning:

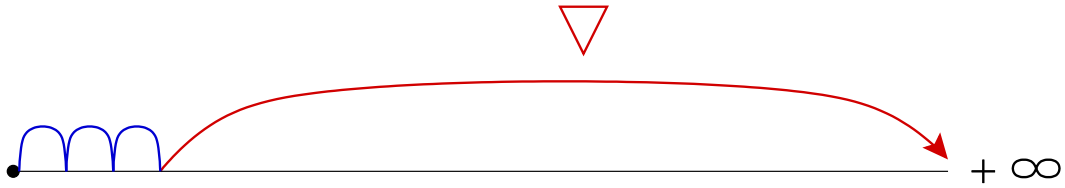


Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

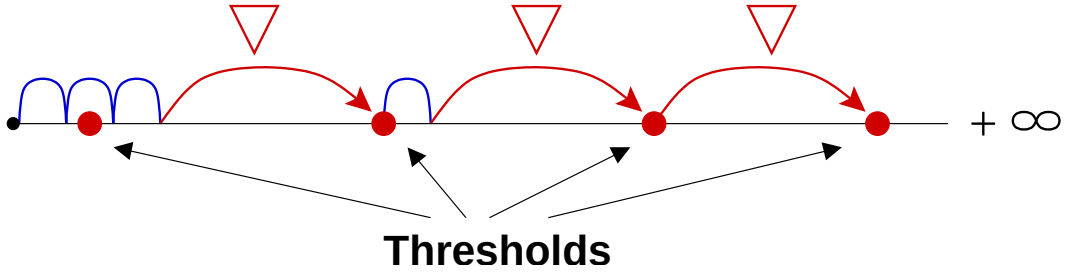
Convergence Accelerator: Widening

Principle:

- Brute-force widening:



- Widening **with thresholds**:



Examples:

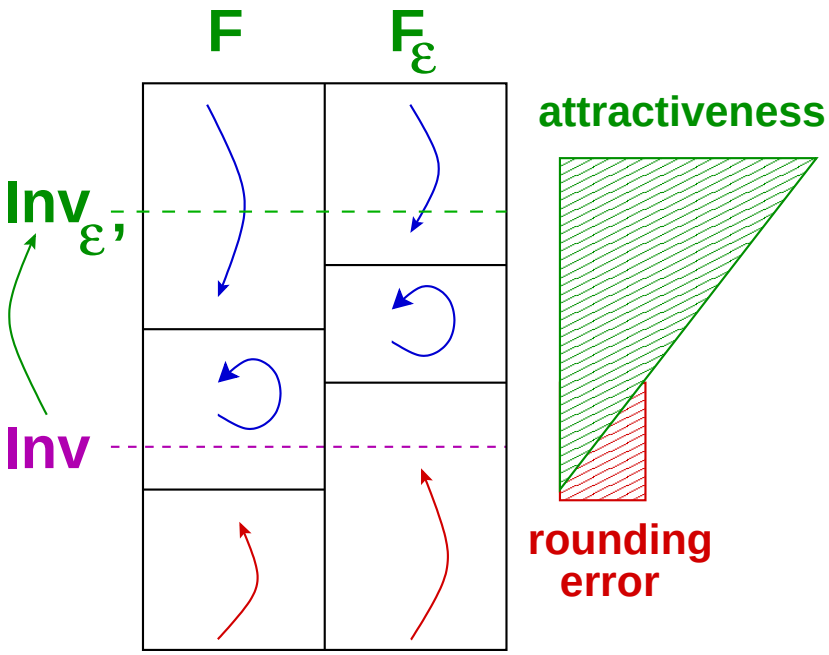
- 1., 10., 100., 1000., etc. for floating-point variables;
- maximal values of data types;
- syntactic program constants, etc.

Fixpoint Stabilization for Floating-point

Problem:

- Mathematically, we look for an abstract invariant **inv** such that $F(\mathbf{inv}) \subseteq \mathbf{inv}$.
- Unfortunately, abstract computation uses floating-point and incurs rounding: maybe $F_\varepsilon(\mathbf{inv}) \not\subseteq \mathbf{inv}$!

Solution:



- Widen **inv** to $\mathbf{inv}_{\varepsilon'}$ with the hope to jump into a **stable zone of** F_ε.
- Works if F has some **attractiveness** property that fights against rounding errors (otherwise iteration goes on).
- ε' is an analysis parameter.

Experience with the use of ASTRÉE

Use of ASTRÉE

- The analyzer:
  - ♦ Full automatic mode:
    Should do well for the families of programs ASTRÉE is designed for
  - ♦ $\simeq 150$ options, so as to set
    - ◊ The input (one or many files...)
    - ◊ The iteration strategy, the packing strategy
    - ◊ The domains to enable or disable, domain parameters
    - ◊ The export of invariants to disk
  - ♦ Standard output: alarms, invariants
  - ♦ Can be run in parallel mode
- A graphical interface:
  Navigation through invariants (saved invariants)

Main Practical Results

- Used on 2 families of synchronous embedded programs
- Results: 3 development versions in the second family
- 2.2 GHz bi-opteron, 1 processor used, 64-bit architecture

Nb of lines	70 000	226 000	400 000
Number of iterations	32	51	88
Memory (Gb)	0.6	1.3	2.2
Time	46mn	3h57mn	11h48mn
False alarms	0	0	0

Conclusion

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^{th} *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.

Recent progress

– More gereral memory model (`union`, pointer arithmetics) [LETCS '03]

Reference

[LETCS '03] A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. ACM SIGPLAN/SIGBED Conf. on LCTES, Ottawa, Ontario, Canada, June 14-16, 2006. ACM Press, pp. 54-63.

Conclusion

- **Success story**:
  - we succeed where **a commercial abstract interpretation-based static analysis tool failed**
    (because of prohibitive time and memory consumption and very large number of false alarms);
- **Usable** in practice for verification:
  - **directly applicable** to other similar programs
    by changing some analyzer parameters,
  - approach **generalizable** to other program families
    by including new abstract domains and specializing the iteration strategy.
    (e.g.: power-on self-test for a family of embedded systems.)

Main Project Results

- The proof of strong safety properties is amenable to static analysis methods:
  - Very few or no false alarms
  - Reasonable resource usage
  - Thanks to a specialized abstract interpreter
- Many practical and theoretical advances:
  - Relational numerical domains and floating point
  - Packing, linearization and relational domains
  - Development of new, specialized domains
  - Implementation of symbolic domains, e.g. partitioning, symbolic...

Crystal ball

Future & Grand Challenges

Future (2/5 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)

Moving to product-based q.a.

Currently: DO178B is all about process

Static analysis is not about process but about the product (program).

Process does not prove anything on the result, sound static analysis does.

Increased use of sound static analysis for safety-critical systems.

Outside critical systems

Sound static analysis tools have limitations

- false alarms
- cost of analysis (CPU, memory, time)
- limitations on accepted programs (ugly pointer arithmetics etc.)

Unsound bug finders may be more suitable for now for less critical applications.

Integrating verification into the process

Our experience: hard to verify = complex invariants = design not understood too well

If analysis hard, maybe fix the design / program and not the analyzer.

Needs analysis to be early in the process not as last-step q.a.

Need for feedback to designers to make analysis easier.

Designers should avoid unsafe or complicated constructs

Future integrated verification

Verification useful if integrated in development cycle.

To begin at Simulink / Scade / other high-level spec (work in progress)

Not as a final step (too late to fix "bizarre" things, often no "smoking gun" justifying redesign)

Needs efficient verification (run during the night)

Perspectives

- Allow for the parallelization of the analysis:
  It works, allows cutting down the analysis time
- Extension of the memory model (work in progress):
  Unions, pointer arithmetic
- Analyze asynchronous programs
- Certify the assembly code
  Validation of the translation (successful prototype)
- Prove formally some ASTRÉE components
- Tracking semi-automatically the source of alarms:
  - Either prove an alarm false
  - Or restrict the alarm context
    (help to find a scenario or the imprecision)

  Encouraging early results: alarms successfully diagnosed

THE END, THANK YOU

More references at URL www.di.ens.fr/~cousot
www.astree.ens.fr.

Bibliography

References

[2] 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.

Bibliography

[Bla05] B. Blanchet. Security protocols: From linear to classical logic by abstract interpretation. *Inf. Process. Lett.*, 95(5):473–479, Sep. 2005.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

[CC92] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In 19^{th} *POPL*, pages 83–94, Albuquerque, NM, US, 1992. ACM Press.

[CC95] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc. 7th FPCA*, pages 170–181, La Jolla, CA, US, 25–28 June 1995. ACM Press.

[CC97] P. Cousot and R. Cousot. Grammar analysis by abstract interpretation. Res. rep., LIENS, École Normale Supérieure, Paris, FR, June 1997.

[CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *27th POPL*, pages 12–25, Boston, MA, US, Jan. 2000. ACM Press.

[CC02] P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interrpetation. In *29th POPL*, pages 178–190, Portland, OR, US, Jan. 2002. ACM Press.

[CC03] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoret. Comput. Sci.*, 290(1):531–544, Jan. 2003.

[CC04] P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In *31st POPL*, pages 173–185, Venice, IT, 14–16 Jan. 2004. ACM Press.

[CC06] P. Cousot and R. Cousot. Grammar analysis and parsing by abstract interpretation, invited chapter. In T. Reps, M. Sagiv, and J. Bauer, editors, *Program Analysis and Compilation, Theory and Practice: Essays dedicated to Reinhard Wilhelm*, LNCS 4444, pages 178–203. Springer, 2006.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th POPL*, pages 84–97, Tucson, AZ, 1978. ACM Press.

[Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, 277(1—2):47–103, 2002.

[Cou03] P. Cousot. Verification by abstract interpretation, invited chapter. In N. Dershowitz, editor, *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, pages 243–268. LNCS 2772, Springer, Taormina, IT, 29 June – 4 Jul. 2003.

[Dan07] V. Danos. Abstract views on biological signalling. In *Mathematical Foundations of Programming Semantics, 23rd Annual Conf. (MFPS XXIII)*, 2007.

[GM04] R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *31st POPL*, pages 186–197, Venice, IT, 2004. ACM Press.

[JP06] Ph. Jorrand and S. Perdrix. Towards a quantum calculus. In *Proc. 4th Int. Work. on Quantum Programming Languages, ENTCS*, 2006.

[PCJD07] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. Semantics-based approach to malware detection. In *34th POPL*, pages 238–252, Nice, France, 17–19 Jan. 2007. ACM Press.

[Per06] S. Perdrix. *Modèles formels du calcul quantique : ressources, machines abstraites et calcul par mesure*. PhD thesis, Institut National Polytechnique de Grenoble, Laboratoire Leibniz, 2006.

[PG05a] M. Dalla Preda and R. Giacobazzi. Control code obfuscation by abstract interpretation. In *Proc. 3rd IEEEInt. Conf. SEFM '05*, Koblenz, DE, 2005. IEEE Comp. Soc. Press.

[PG05b] M. Dalla Preda and R. Giacobazzi. Semantic-based code obfuscation by abstract interpretation. In L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Proc. 32nd Int. Coll. ICALP '05*, volume 3580 of *Lisbon, PT, LNCS*, pages 1325–1336. Springer, 11–15 Jul. 2005.

[Riv05] X. Rival. Understanding the origin of alarms in ASTRÉE. In C. Hankin and I. Siveroni, editors, *Proc. 12th Int. Symp. SAS '05*, pages 303–319, London, UK, LNCS 3672, 7–9 Sep. 2005.

[RT04] F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In D. Schmidt, editor, *Proc. 30th ESOP '04*, volume 2986 of *LNCS*, pages 18–32, Barcelona, ES, Mar. 29 – Apr. 2 2004. Springer.

Papers available on http://www.di.ens.fr/~cousot/COUSOTpapers.shtml.