Grand Challenges for Abstract Interpretation

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

2nd Workshop on Dependable Systems Evolution
Gresham College, London, UK
March 18, 2004

Talk Outline

A Few Words on Abstract Interpretation

Abstract Interpretation

- Abstract interpretation theory formalizes the idea of sound approximation for mathematical constructs involved in the specification of the semantics and properties of computer systems.

References

[POPL '77] P. Cousot & 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, 1977.

[Thesis] 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, 21 Mar. 1978.

[PO- PL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pages 269–282, 1979.

[JLC '92] P. Cousot & R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, 1992.

1 – Abstract Domains

- Program concrete properties are specified by the (operational) semantics of programming languages;
- Program abstract properties are elements of abstract domains (posets/lattices/...);
- Program property abstraction is performed by (effective) conservative approximation of concrete properties;
- The abstract properties (hence abstract semantics) are sound but may be incomplete with respect to the concrete properties (semantics);

2 – Correspondence between Concrete and Abstract Properties

- If any concrete property has a best approximation, approximation is formalized by Galois connections (or equivalently closure operators, Moore families, etc. [1]);
- Otherwise, weaker abstraction/ concretization correspondences are available [2];

[1] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.

[2] P. Cousot & R. Cousot. *Abstract interpretation frameworks.* JLC 2(4):511–547, 1992.

3 – Semantics Abstraction

- Program concrete semantics and specifications are defined by syntactic induction and composition of fixpoints (or using equivalent presentations [3]);
- The property abstraction is extended compositionally to all constructions of the concrete/abstract semantics, including fixpoints;
- This leads to a constructive design of the abstract semantics by approximation of the concrete semantics [4];

[3] P. Cousot & R. Cousot. *Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game theoretic form.* CAV '95, LNCS 939, pp. 293–308, 1995.

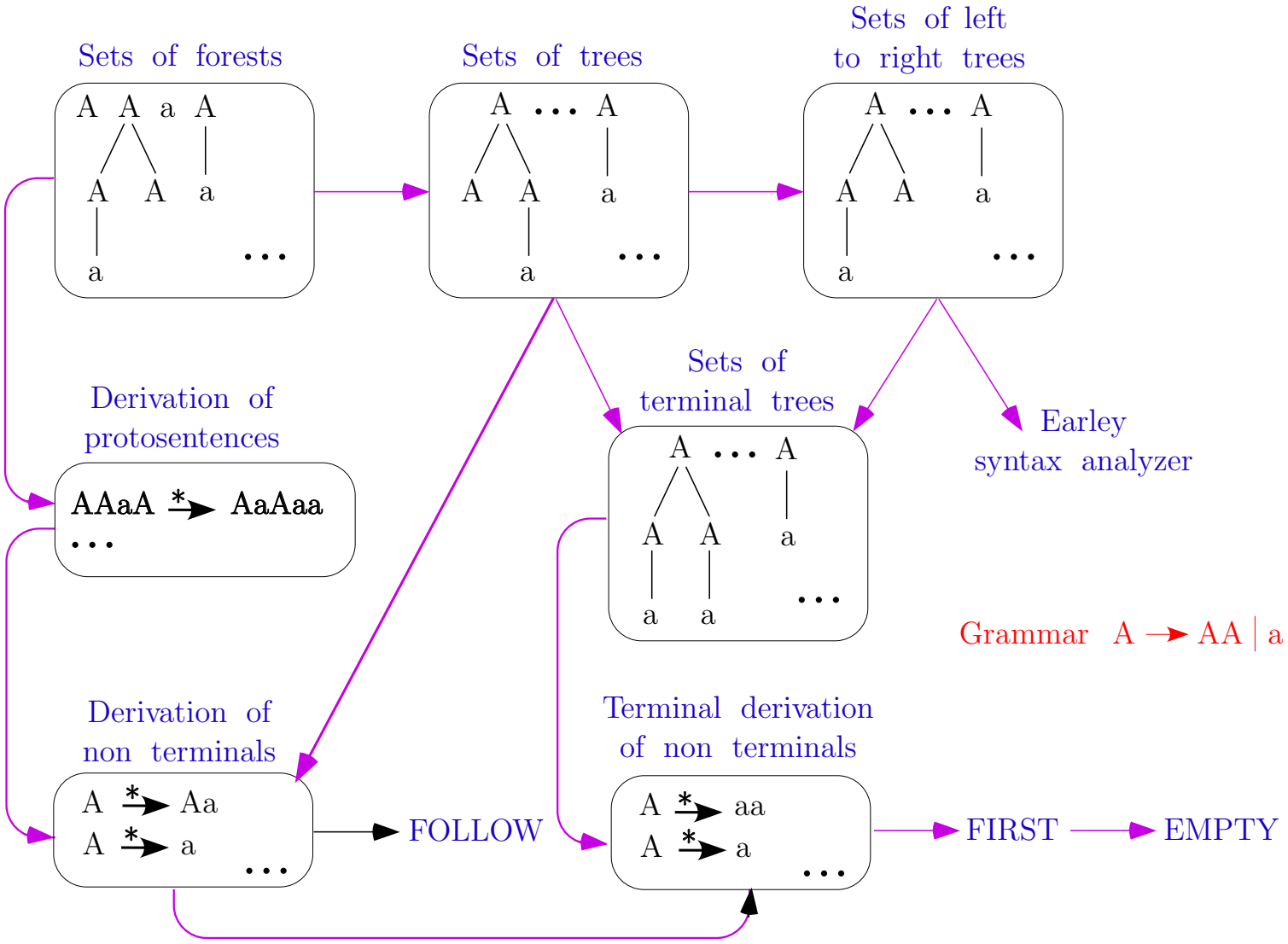
[4] P. Cousot & R. Cousot. *Inductive definitions, semantics and abstract interpretation.* POPL, 83–94, 1992.

4 — Effective Analysis/Checking/ Verification Algorithms

- Computable abstract semantics lead to effective program analysis/checking/verification algorithms;
- Furthermore fixpoints can be approximated iteratively by convergence acceleration through widening/narrowing that is non-standard induction [5].

[5] P. Cousot & R. Cousot. *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.* ACM POPL, pp. 238–252, 1977.

Example: Grammar Abstraction



Applications of Abstract Interpretation

- **Static Program Analysis** [POPL '77,78,79] inluding **Dataflow Analysis** [POPL '79,00], **Set-based Analysis** [FPCA '95], **Predicate Abstraction** [Manna's festschrift '04]
- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92, TCS 277(1–2) 2002]
- **Typing** [POPL '97]
- **Model Checking** [POPL '00]

Applications of Abstract Interpretation (Cont'd)

- **Program Transformation** [POPL '02]
- **Software watermarking** [POPL '02]

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

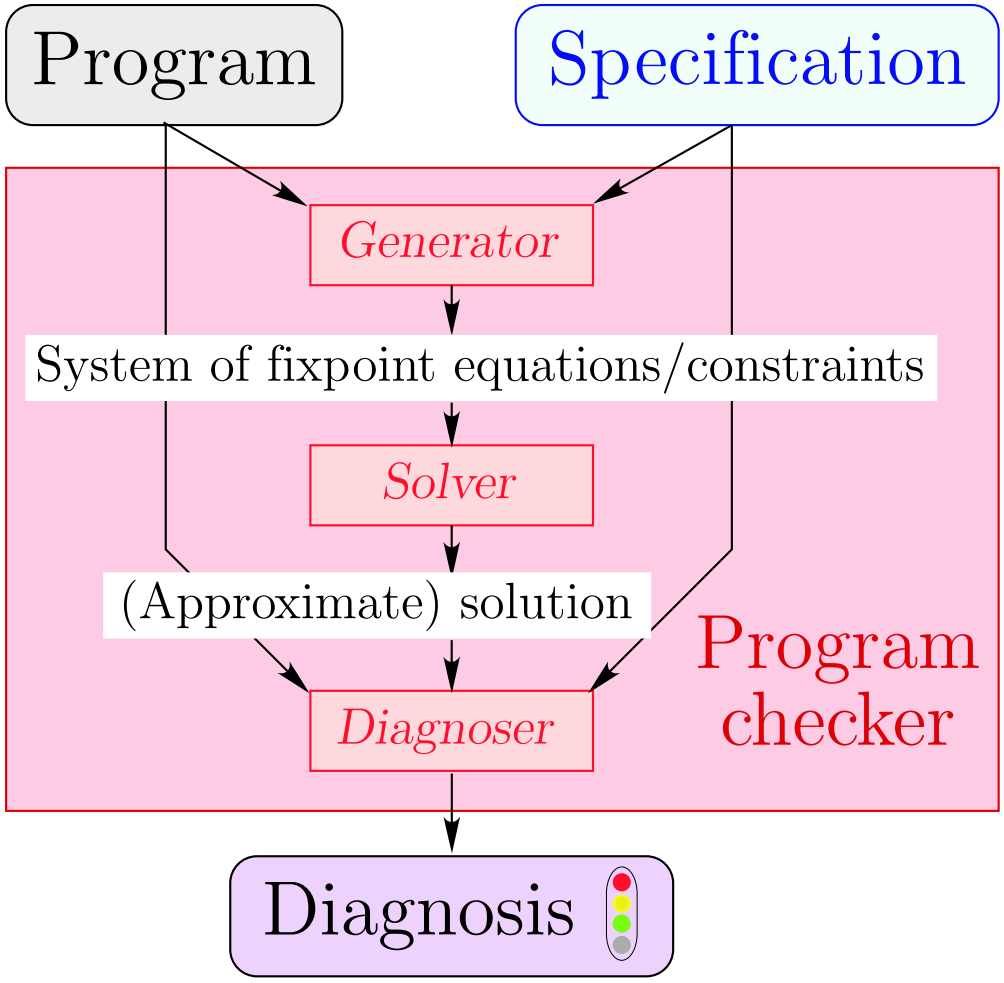
A Practical Application of Abstract Interpretation to the Verification of Safety Critical Embedded Software

Reference

[1] 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, pages 85–108. Springer, 2002.

[2] 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.

Static Program Analysis



ASTRÉE: A Sound, Automatic, Specializable, Domain-Aware, Parametric, Modular, Efficient and Precise Static Program Analyzer

www.astree.ens.fr

- C programs:
  - structured C programs;
  - no dynamic memory allocation;
  - no recursion.
- Application Domain: safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.

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`)

Abstract Semantics

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

Implicit Specification: Absence of Runtime Errors

- No violation of the norm of C (e.g. array index out of bounds)
- No implementation-specific undefined behaviors (e.g. maximum short integer is 32767)
- 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/A380 fly-by-wire system

- C program, automatically generated from a high-level specification
- A340: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays.

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 volatile output variables;

wait_for_clock ();
end loop

- Requirements: the only interrupts are clock ticks;
- Execution time of loop body less than a clock tick [3].

Reference

[3] 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. *ESOP (2001)*, LNCS 2211, 469–485.

Characteristics of the ASTRÉE Analyzer

Static: compile time analysis (≠ run time analysis Rational Purify, Parasoft Insure++)

Program Analyzer: analyzes programs not micromodels of programs (≠ PROMELA in SPIN or Alloy in the Alloy Analyzer)

Automatic: no end-user intervention needed (≠ ESC Java, ESC Java 2)

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

Characteristics of the ASTRÉE Analyzer (Cont'd)

Multiabstraction: uses many numerical/symbolic abstract domains ($\neq$ symbolic constraints in Bane)

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)

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

Characteristics of the ASTRÉE Analyzer (Cont'd)

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

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

Characteristics of the ASTRÉE Analyzer (Cont'd)

Modular: an analyzer instance is built by selection of OCAML modules from a collection each implementing an abstract domain

Precise: few or no false alarm when adapted to an application domain $\longrightarrow$ VERIFIER!

Benchmarks for the Primary Flight Control Software of the Airbus A340

- Comparative results (commercial software):

  4,200 (false?) alarms,

  3.5 days;

- Our results:

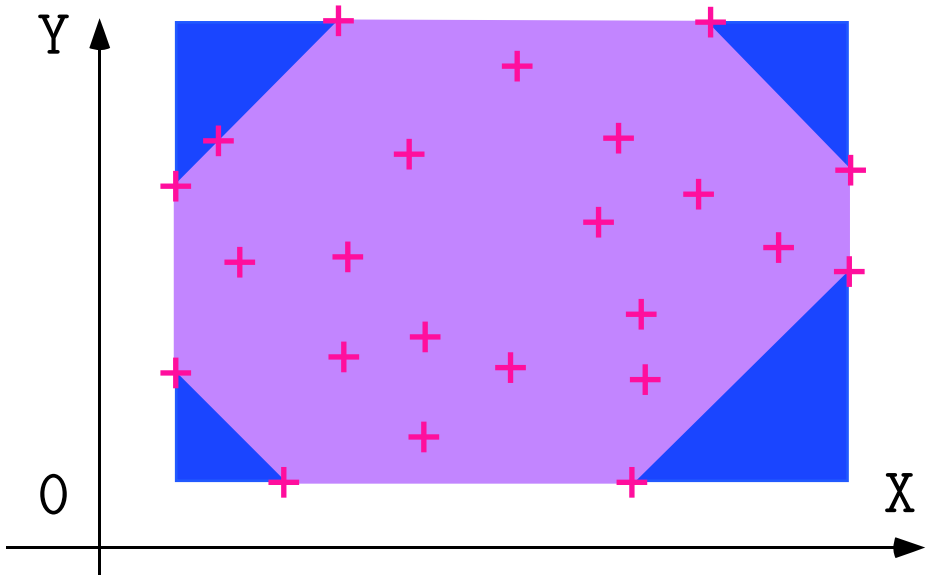
  0 alarm,

  $1^{\mathrm{h}}20$ on 2.8 GHz PC,

  300 Megabytes

  $\longrightarrow$ A world première!

General-Purpose Abstract Domains: Intervals and Octagons



Intervals:

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

Octagons [4]:

$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 78 \\ 1 \leq y \leq 20 \\ x - y \leq 03 \end{cases}$$

Difficulties: many global variables, IEEE 754 floating-point arithmetic (in program and analyzer)

Reference

[4] A. Miné. A New Numerical Abstract Domain Based on Difference-Bound Matrices. In *PADO'2001*, LNCS 2053, Springer, 2001, pp. 155–172.

[5] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP'04*, Barcelona, LNCS , Springer, 2004 (to appear).

Floating-Point Computations

- Code Sample:

```
/* 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
```

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

```
/* 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
```

Clock Abstract Domain for Counters

- Code Sample:

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

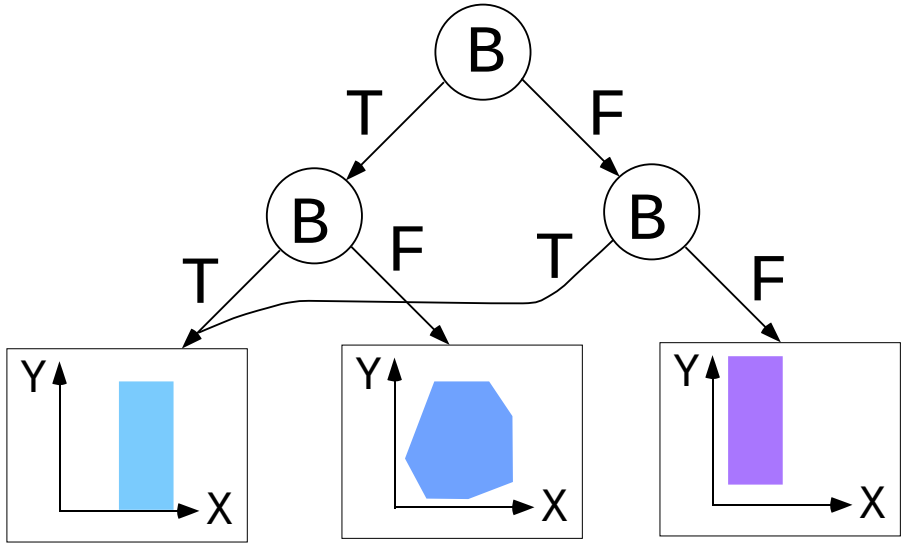
  - Output `T` is true iff the volatile input `I` has been true for the last `n` clock ticks.
  - The clock ticks every `s` seconds for at most `h` hours, thus `R` is bounded.
  - To prove that `R` cannot overflow, we must prove that `R` cannot exceed the elapsed clock ticks *(impossible using only intervals)*.

- 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`.

Boolean Relations for Boolean Control

- Code Sample:

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



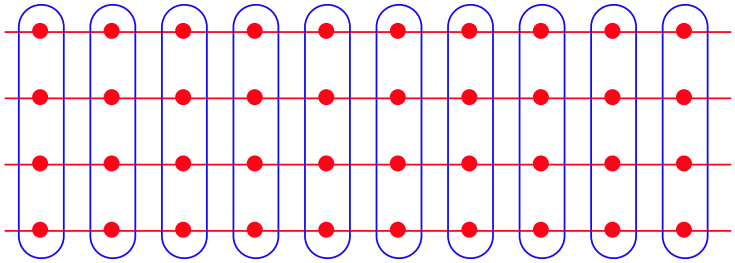
The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

Control Partitionning for Case Analysis

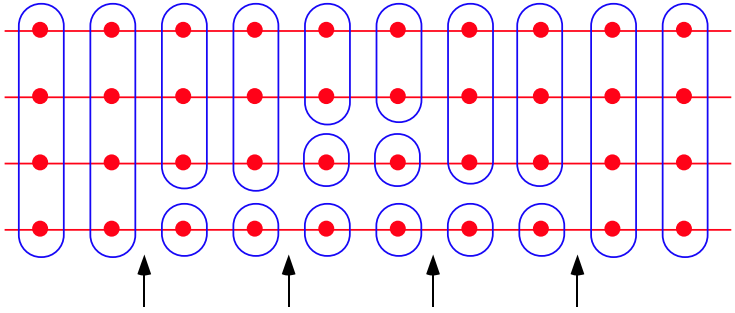
- Code Sample:

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

  ... found invariant −100 ≤ x ≤ 100 ...

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

Control point partitionning:

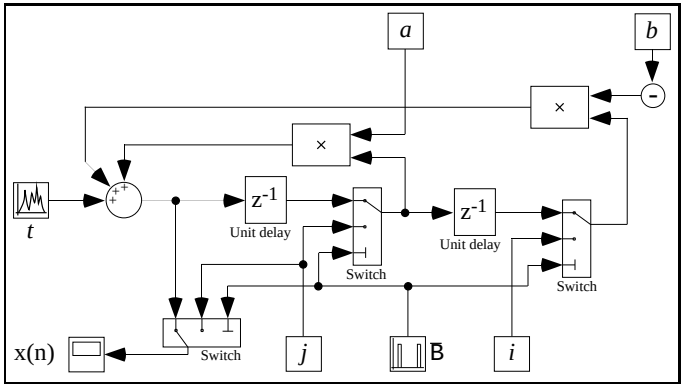


Trace partitionning:

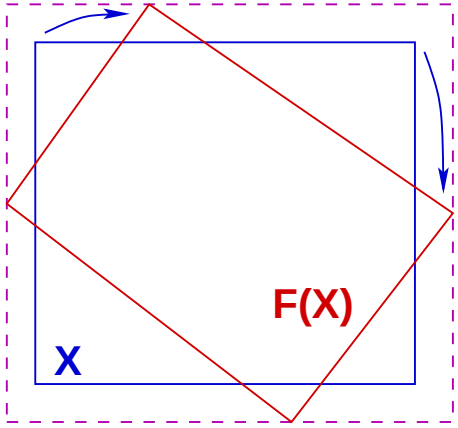


Ellipsoid Abstract Domain for Filters

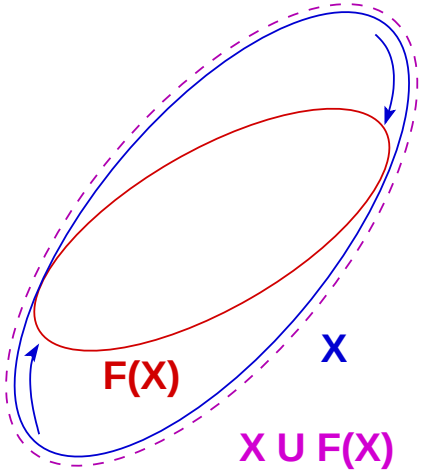
2d Order Digital Filter:



- 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.



unstable interval

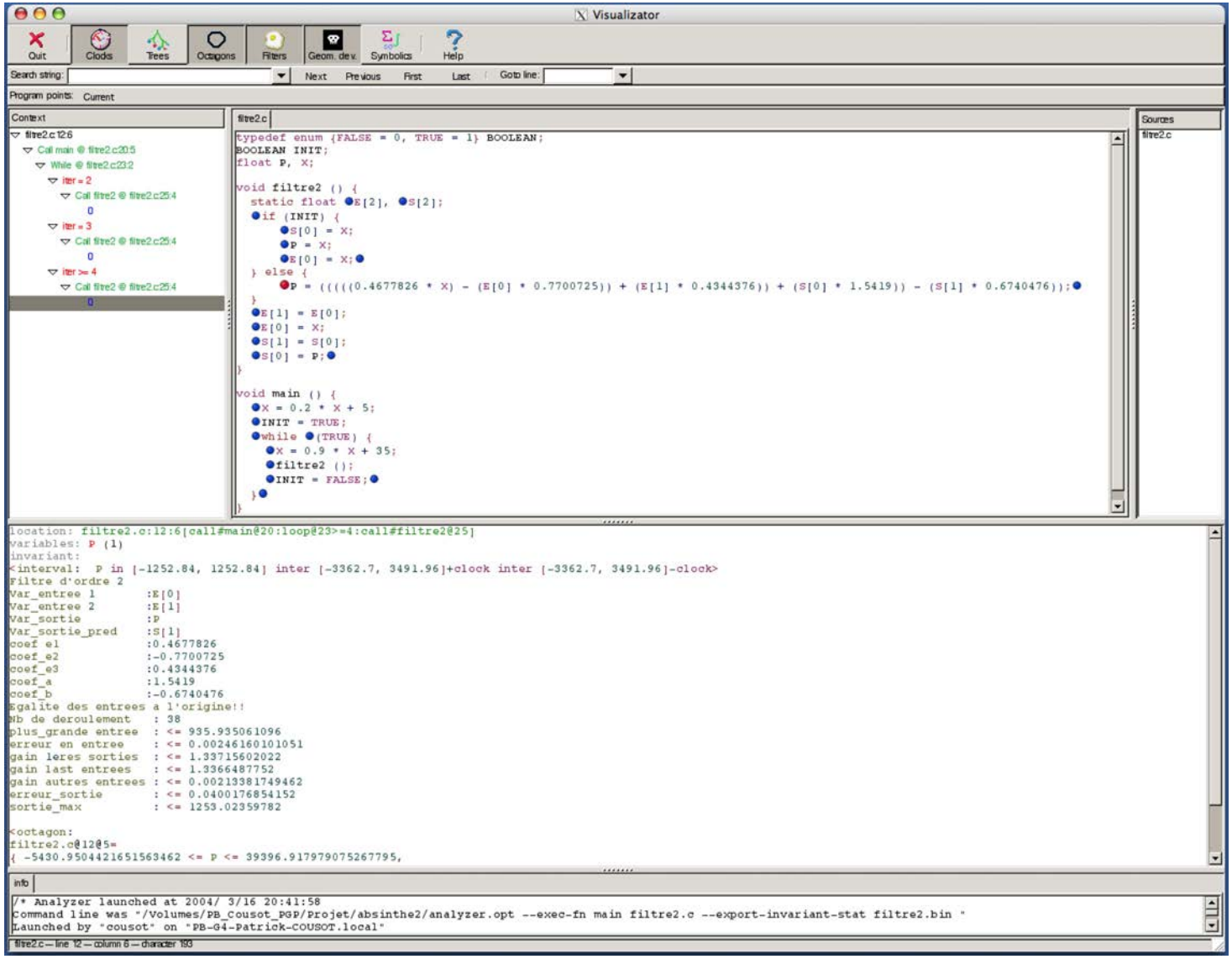


stable ellipsoid

Reference

[6] J. Feret. Static analysis of digital filters. In *ESOP'04*, Barcelona, LNCS , Springer, 2004 (to appear).

Example of Analysis Session



The main loop invariant

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.

Conclusion on Verification by Abstraction

- Most applications of abstract interpretation tolerate a small rate (typically 5 to 15%) of false alarms:
  - Program transformation $\rightarrow$ do not optimize,
  - Typing $\rightarrow$ reject some correct programs, etc,
  - WCET analysis $\rightarrow$ 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.

Grand Challenges for Abstract Interpretation

Scaling-up Static Program Analysis

- Success on 100 000 LOCS (primary flight control software of the Airbus A340)
- Working on 500 000 LOCS (primary flight control software of the Airbus A380)

$\longrightarrow$ We should be able to statically analyze huge software (3 000 000 LOCS (all software of the A380) to 30 000 000 LOCS (Windows OS)).

Formal Verification of Static Analyzers

- Abstract interpretation can formalize static program analysis [POPL '77,78,79]
- The design and programming of ASTRÉE is tightly guided by abstract interpretation

$\longrightarrow$ We should be able to completely verify the ASTRÉE static analyzer.

Formalization of Compilation

Abstract interpretation can formalize:

- Parsing [TCS 290(1) 2002]
- Typing [POPL '97]
- Static analysis [POPL '77,78,79]
- Program transformation (partial evaluation, code generation) [POPL '02]

$\longrightarrow$ We should be able to completely formalize compilation.

THE END, THANK YOU

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