

# « Why ASTRÉE does scale? »

Patrick Cousot

École normale supérieure

45 rue d'Ulm, 75230 Paris cedex 05, France

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

Dagstuhl seminar 08161, Scalable Program Analysis — Schloß  
Dagstuhl — April 15th, 2008



# 1. Abstract Interpretation



# The Theory of Abstract Interpretation

- A theory of **sound approximation of mathematical structures**, in particular those involved in the behavior of computer systems
- Systematic derivation of **sound methods and algorithms for approximating undecidable or highly complex problems** in various areas of computer science
- Main practical application is on the **safety and security of complex hardware and software** computer systems
- Abstraction: extracting information from a system description that is relevant to proving a property



## 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];
- **Hierarchies of Semantics and Proof Methods** [CC92b], [Cou02];
- **Typing & Type Inference** [Cou97];
- **(Abstract) Model Checking** [CC00];
- **Program Transformation** (including program optimization, partial evaluation, etc) [CC02];



## Applications of Abstract Interpretation (Cont'd)

- Software Watermarking [CC04];
- Bisimulations [RT04, RT06];
- Language-based security [GM04];
- Semantics-based obfuscated malware detection [PCJD07].
- Databases [AGM93, BPC01, BS97]
- Computational biology [Dan07]
- Quantum computing [JP06, Per06]

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



## 2. ASTRÉE



# Project Members

<http://www.astree.ens.fr/>



Bruno BLANCHET<sup>1</sup>



Patrick COUSOT



Radhia COUSOT



Jérôme FERET



Laurent MAUBORGNE



Antoine MINÉ



David MONNIAUX<sup>2</sup>



Xavier RIVAL

---

<sup>1</sup> Nov. 2001 — Nov. 2003.

<sup>2</sup> Nov. 2001 — Aug. 2007.



# 3. Motivation





# The Complexity of Software Design

- The **design of complex software** is difficult and economically critical
- Example ([www.designnews.com/article/CA6475332.html](http://www.designnews.com/article/CA6475332.html)):

## **Boeing Confirms 787 Delay, Fasteners, Flight Control Software Code Blamed**

John Dodge, Editor-in-Chief – Design News, September 5, 2007

Boeing officials confirmed today that a fastener shortage and **problems with flight control software** have pushed “first flight” of the Boeing 787 Dreamliner to sometime between mid-November and mid-December.

...

The software delays involve Honeywell Aerospace, which is responsible for flight control software. **The work on this part of the 787 was simply underestimated**, said Bair.



## Tool-Based Software Design Methods

- New tool-based software design methods will have to emerge to face the unprecedented growth and complexification of critical software
- E.g. FCPC (Flight Control Primary Computer)



## 4. Problematics



# Requirements of Verification Static Analysis<sup>3</sup>

- A verifying static program analyzer must be (at least)
- **useful** (with respect to a correctness proof objective)
  - **sound** (with respect to a concrete semantics)
  - **conclusive** (with respect to a specification)
  - **non-intrusive** (with respect to a system development practice)
  - **realistic** (applicable in an weird industrial environment)
  - **scalable** (to actual industrial code)
- ... and **cheap if not free!**

---

<sup>3</sup> As opposed to bug-finding static/dynamic analysis



# Making Static Analysis Easy (and Ultimately Useless)

Drop any of the requirements

- usefulness
- soundness
- conclusiveness
- non-intrusiveness
- realism
- scalability



# Abstract Static Analysis

- Sound **unprecise abstraction** is mandatory **to scale up**, but
- Sound **precise abstraction** is mandatory **to be conclusive**.

Counter-example: brute force methods (like software model checking) simply fail.



## Difficulties of Static Analysis

- Floyd/Naur proof method  $\forall P \in \mathcal{L}, \forall S \in \mathcal{S}[[P]]$ , let  $\mathcal{D}[[P]] \supseteq \mathcal{S}[[P]]$ , and  $F[[P]] \in \mathcal{D}[[P]] \mapsto \mathcal{D}[[P]]$ :

$$\text{lfp}^{\subseteq} F[[P]] \subseteq S \Leftrightarrow \exists I \in \mathcal{D}[[P]] : F[[P]](I) \subseteq I \wedge I \subseteq S$$

- Abstraction  $\langle \mathcal{D}[[P]], \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \alpha(\mathcal{D}[[P]]), \sqsubseteq \rangle$ :  
 $\Leftarrow \exists \bar{I} \in \alpha(\mathcal{D}[[P]]) : \text{lfp}^{\sqsubseteq} \alpha \circ F[[P]] \circ \gamma \sqsubseteq \bar{I} \wedge \gamma(\bar{I}) \subseteq S$

- Main difficulty: in general, there is no inductive invariant  $\bar{I}$  in the abstract:

$$\forall \bar{I} \in \alpha(\mathcal{D}[[P]]) : \alpha \circ F[[P]] \circ \gamma(\bar{I}) \not\sqsubseteq \bar{I}$$



## 5. ASTRÉE Fundamental Choices





# Language



Typical choices: **Choice of the Language**  $\forall P \in \mathcal{L}$

- **Deductive methods** and **model checking**:  $\mathcal{L} = \{P\}$ , for one (model of a) program
- **Data flow analysis**:  $\mathcal{L} = \text{C, C++}, \dots$ , one programming language
- **ASTRÉE**: the family of control/command C codes automatically generated from a synchronous specification (SAO/SCADE)<sup>4</sup>

---

<sup>4</sup> Outside this scope, ASTRÉE is likely not be useful, conclusive, non-intrusive, realistic, and/or scalable!



## 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 [Min06a]
  - pointer arithmetics & casts [Min06a]
- without
  - dynamic memory allocation
  - recursive function calls
  - unstructured/backward branching
  - conflicting side effects
  - C libraries, system calls (parallelism)

*Such limitations are quite common for embedded safety-critical software.*



# 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,  
as verified by the aiT WCET Analyzers [FHL<sup>+</sup>01].



# Concrete Semantics



## Choice of the Concrete Semantics $\mathcal{D}[[P]], F[[P]], P \in \mathcal{L}$

Set of prefix-closed traces for a transition relation defined by

- the 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>5</sup>)

---

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



## The Semantics of C is Hard (Ex. 1: Floats)

“*Put  $x$  in  $[m, M]$  modulo  $(M - m)$* ”:

$$x' = x - (\text{int}) \left( (x-m)/(M-m) \right) * (M-m);$$

- The programmer thinks  $x' \in [m, M]$
- But with  $M = 4095$ ,  $m = -M$ , IEEE double precision, and  $x$  is the greatest float strictly less than  $M$ , then  $x' = m - \epsilon$  ( $\epsilon$  very small).

Floats are not real.

ASTRÉE has an abstraction to handle this modulo problem (J. Feret, unpublished)





## The Semantics of C is Hard (Ex. 2: Runtime Errors)

What is the effect of out-of-bounds array indexing?

```
% cat unpredictable.c
#include <stdio.h>
int main () { int n, T[1];
  n = 2147483647;
  printf("n = %i, T[n] = %i\n", n, T[n]);
}
```

Yields different results on different machines:

n = 2147483647, T[n] = 2147483647	Macintosh PPC
n = 2147483647, T[n] = -1208492044	Macintosh Intel
n = 2147483647, T[n] = -135294988	PC Intel 32 bits
Bus error	PC Intel 64 bits

Execution stops after a runtime error with unpredictable results<sup>6</sup>.

---

<sup>6</sup> Equivalent semantics if no alarm.



## Different Classes of Run-time Errors

1. **Errors terminating the execution**<sup>7</sup>. 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**<sup>8</sup>. ASTRÉE warns and continues with worst-case assumptions.
3. **Errors not terminating the execution with unpredictable outcome**<sup>9</sup>. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.

⇒ ASTRÉE is sound with respect to **C standard**, unsound with respect to **C implementation**, unless **no false alarm**.

---

<sup>7</sup> floating-point exceptions e.g. (invalid operations, overflows, NaN, etc.) when traps are activated

<sup>8</sup> e.g. overflows over signed integers resulting in some signed integer.

<sup>9</sup> e.g. memory corruptions.



## Why prefix-closed traces?

- Burstall's proof method (using traces) is equivalent to Floyd method (with set of states i.e. invariant) but much easier

```
while (x > 1) {  
    if (odd(x)) { x = x + 1; }  
    else { x = x / 2; }  
}
```

- You can always later abstract sets of (prefix-closed) traces into sets of states



## Realistic Semantics: Modulo Arithmetics

In C:

```
% cat -n modulo-c.c
 1 #include <stdio.h>
 2 int main () {
 3   int x,y;
 4   x = -2147483647 / -1;
 5   y = ((-x) -1) / -1;
 6   printf("x = %i, y = %i\n",x,y);
 7 }
 8
```

```
% gcc modulo-c.c
```

```
% ./a.out
```

```
x = 2147483647, y = -2147483648
```



# Static Analysis with ASTRÉE

```
% cat -n modulo.c
 1 int main () {
 2 int x,y;
 3 x = -2147483647 / -1;
 4 y = ((-x) -1) / -1;
 5 __ASTREE_log_vars((x,y));
 6 }
 7

% astree -exec-fn main -unroll 0 modulo.c\
|& egrep -A 1 "<integers)|(WARN)"
modulo.c:4.4-18::[call#main@1:]: WARN: signed int arithmetic range
{2147483648} not included in [-2147483648, 2147483647]
<integers (intv+cong+bitfield+set): y in [-2147483648, 2147483647] /\ Top
x in {2147483647} /\ {2147483647} >
```

ASTRÉE signals the overflow and goes on with an unknown value.



## Realistic Semantics: Floats

```
% cat -n scale.c
1 int main () {
2   float x; x = 0.70000001;
3   while (1) {
4     x = x / 3.0;
5     x = x * 3.0;
6     __ASTREE_log_vars((x));
7     __ASTREE_wait_for_clock(());
8   }
9 }
```

```
% gcc scale.c
% ./a.out
x = 0.699999988079071
```

```
% cat scale.config
__ASTREE_max_clock((1000000000));
% astree -exec-fn main -config-sem scale.config -unroll 0 scale.c\
|& grep "x in" | tail -1
direct = <float-interval: x in [0.69999986887, 0.700000047684] >
%
```



## Example of accumulation of small rounding errors

```
% cat -n rounding-c.c
1  #include <stdio.h>
2  int main () {
3      int i; double x; x = 0.0;
4      for (i=1; i<=1000000000; i++) {
5          x = x + 1.0/10.0;
6      }
7      printf("x = %f\n", x);
8  }
```

```
% gcc rounding-c.c
```

```
% ./a.out
```

```
x = 99999998.745418
```

```
%
```

since  $(0.1)_{10} = (0.0001100110011001100\dots)_2$



# Static analysis with ASTRÉE

```
% cat -n rounding.c
1  int main () {
2    double x; x = 0.0;
3    while (1) {
4      x = x + 1.0/10.0;
5      __ASTREE_log_vars((x));
6      __ASTREE_wait_for_clock(());
7    }
8  }

% cat rounding.config
__ASTREE_max_clock((1000000000));

% astree -exec-fn main -config-sem rounding.config -unroll 0 rounding.c\
  |& egrep "(x in)|(\|x\|)|(WARN)" | tail -2
direct = <float-interval: x in [0.1, 200000040.938] >
  |x| <= 1.*((0. + 0.1/(1.-1))*(1.)^clock - 0.1/(1.-1)) + 0.1
      <= 200000040.938
```





# The Patriot missile failure

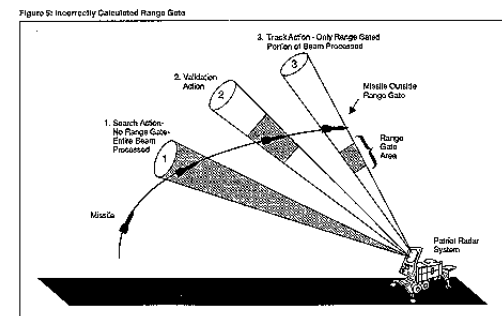
- “On February 25<sup>th</sup>, 1991, a Patriot missile ... failed to track and intercept an incoming Scud (\*).”
- The **software failure** was due to accumulated rounding error (†)



(\*) This Scud subsequently hit an Army barracks, killing 28 Americans.

(†) – “Time is kept continuously by the system’s internal clock in **tenths of seconds**”

- “The system had been in operation for over **100 consecutive hours**”
- “Because the system had been on so long, the **resulting inaccuracy** in the time calculation **caused the range gate to shift** so much that the system could not track the incoming Scud”



# Specification



## Choice of the Specification Language $\mathcal{S}[[P]] \subseteq \mathcal{D}[[P]]$

- By the choice of  $\mathcal{D}[[P]]$ ,  $\mathcal{S}[[P]]$  can be anything specifying prefix-closed sets of traces (automata, grammars, synchronous languages, temporal logic, etc.)
- but
  - Intrusive (who will write the formal specification?)
  - Costly (e.g. to check  $\gamma(\bar{I}) \subseteq S$ )
- In ASTRÉE, **implicit specification (absence of runtime error)** automatically computed from the program text



## 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: Dichotomy Search I

```
% cat dichotomy.c
int main () {
    int R[100], X; short lwb, upb, m;
    lwb = 0; upb = 99;
    while (lwb <= upb) {
        m = upb + lwb;
        m = m » 1;
        if (X == R[m]) { upb = m; lwb = m+1; }
        else if (X < R[m]) { upb = m - 1; }
        else { lwb = m + 1; }
    }
    __ASTREE_log_vars((m));
}

% astree -exec-fn main dichotomy.c |& egrep "(WARN)|(m in)"
direct = <integers (intv+cong+bitfield+set): m in [0, 99] /\ Top >
%
```



## Example: Dichotomy Search II

```
% diff dichotomy.c dichotomy-bug.c
2,3c2,3
<   int R[100], X; short lwb, upb, m;
<   lwb = 0; upb = 99;
--
>   int R[30000], X; short lwb, upb, m;
>   lwb = 0; upb = 29999;
%
% astree -exec-fn main dichotomy-bug.c |& egrep "WARN" | head -n2
dichotomy-bug.c:5.6-19::[call#main@1:loop@4=2:]: WARN: implicit signed int->signed
short conversion range [14998, 44999] not included in [-32768, 32767]
dichotomy-bug.c:7.15-19::[call#main@1:loop@4=2:]: WARN: invalid dereference:
dereferencing 4 byte(s) at offset(s) [0;4294967295] may overflow the variable R of
byte-size 120000 or mis-aligned pointer (1Z+0) may not a multiple of 4
%
```

ASTRÉE finds bugs in programs based on algorithms which have been formally proved correct.



# Iterator



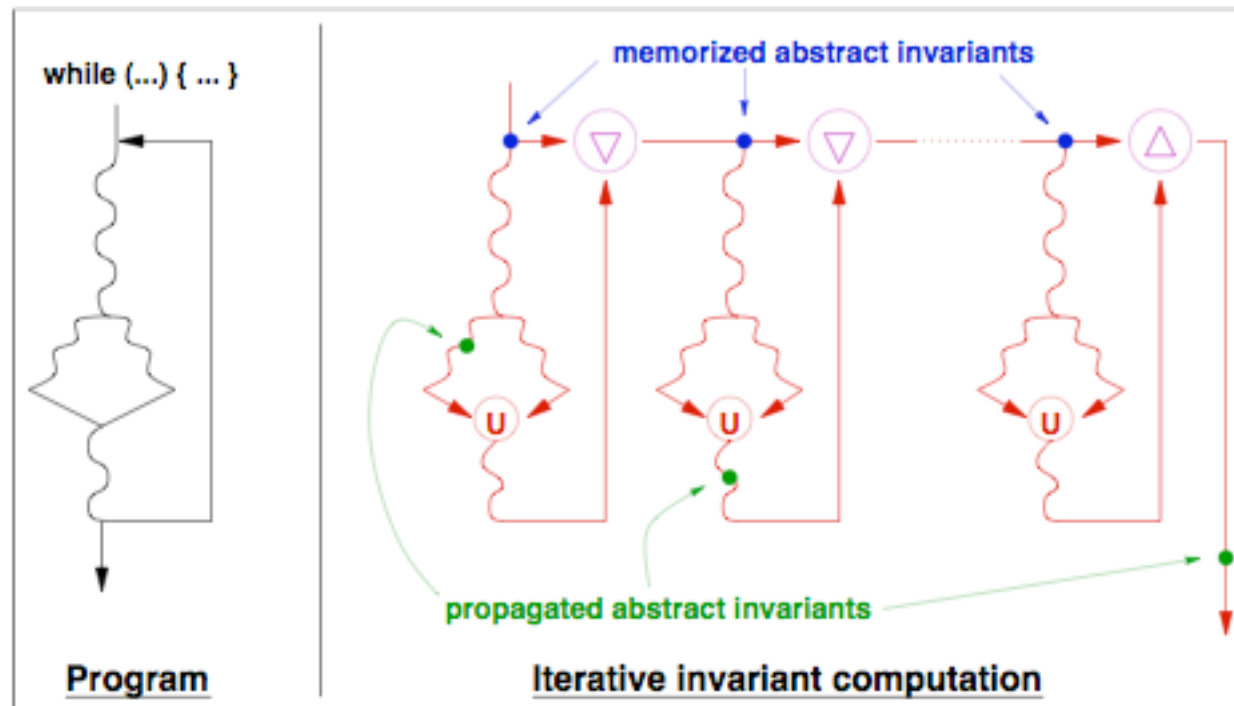
## Choice of the Abstract Iterator $\text{lfp}^{\sqsubseteq} \alpha \circ F[[P]] \circ \gamma$

- **Control graph** (would lose useful information), or
- **ASTRÉE**:
  - isomorphic projection of the set of prefix-closed traces to contexts = call stack + program point
  - by structural induction on the abstract syntax tree
    - initialize (empty traces at program entry point)
    - given a prefix-closed set of traces up to the predecessor contexts, extend each trace by one computation step/transition to the next contexts
    - repeat with widening/narrowing until stabilization





Abstract Iterator  $\text{lfp}^{\sqsubseteq} \alpha \circ F[[P]] \circ \gamma$



# Abstraction



## Bad ideas on Abstraction

- Abstract exclusively to **finite domains** (provably worse than infinite domain plus widening [CC92a])
- **Uniform** abstractions (same abstraction everywhere, everytime, like in dataflow analysis)
- Keep as much **disjunctions** as possible (e.g. predicate abstraction, abstraction is all about “how to get rid of disjunctions”!)
- **Cascaded abstractions**, one after the other (provably worse than reduced product)
- . . .

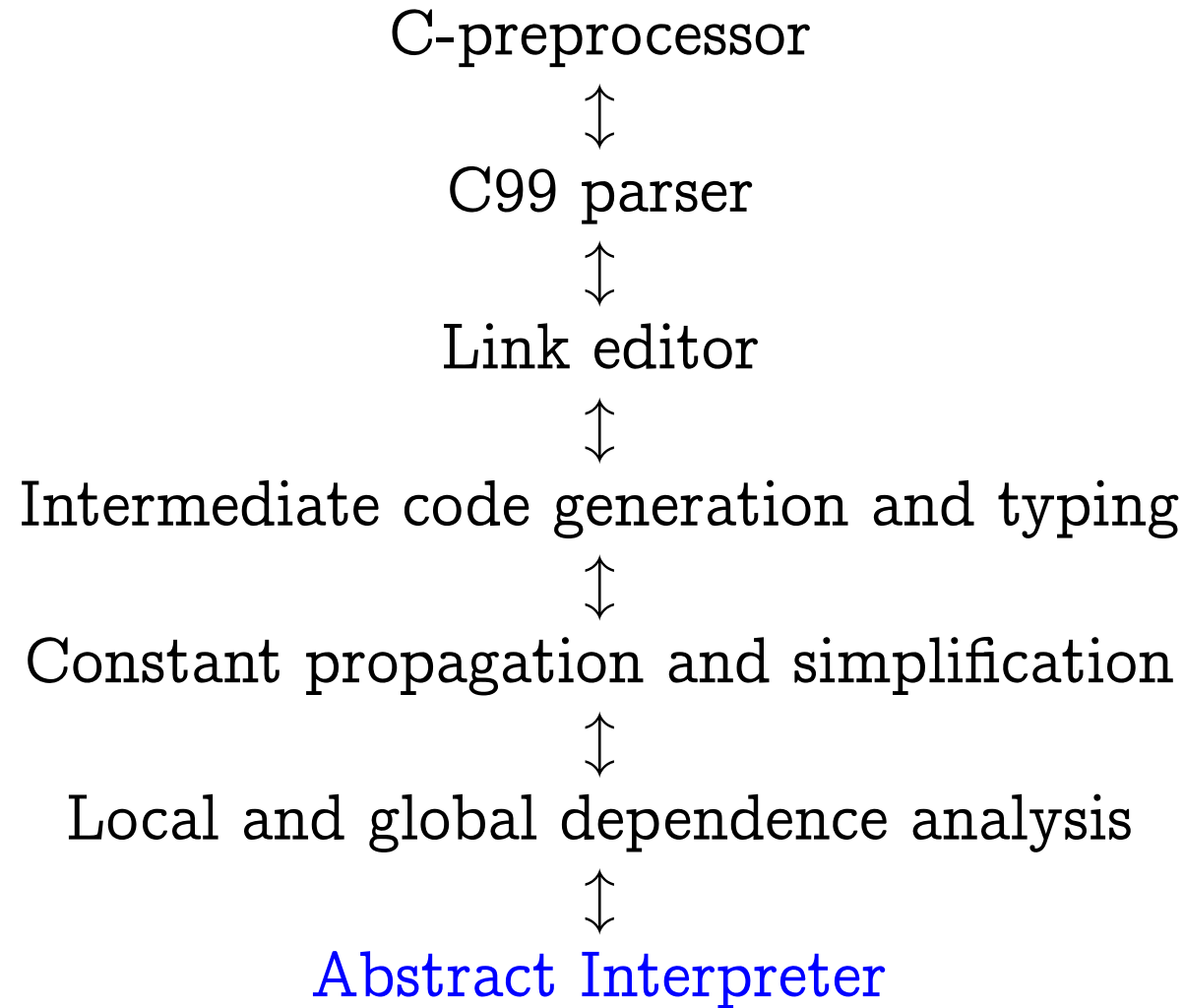


## Choice of the Abstraction $\langle \alpha, \gamma \rangle$

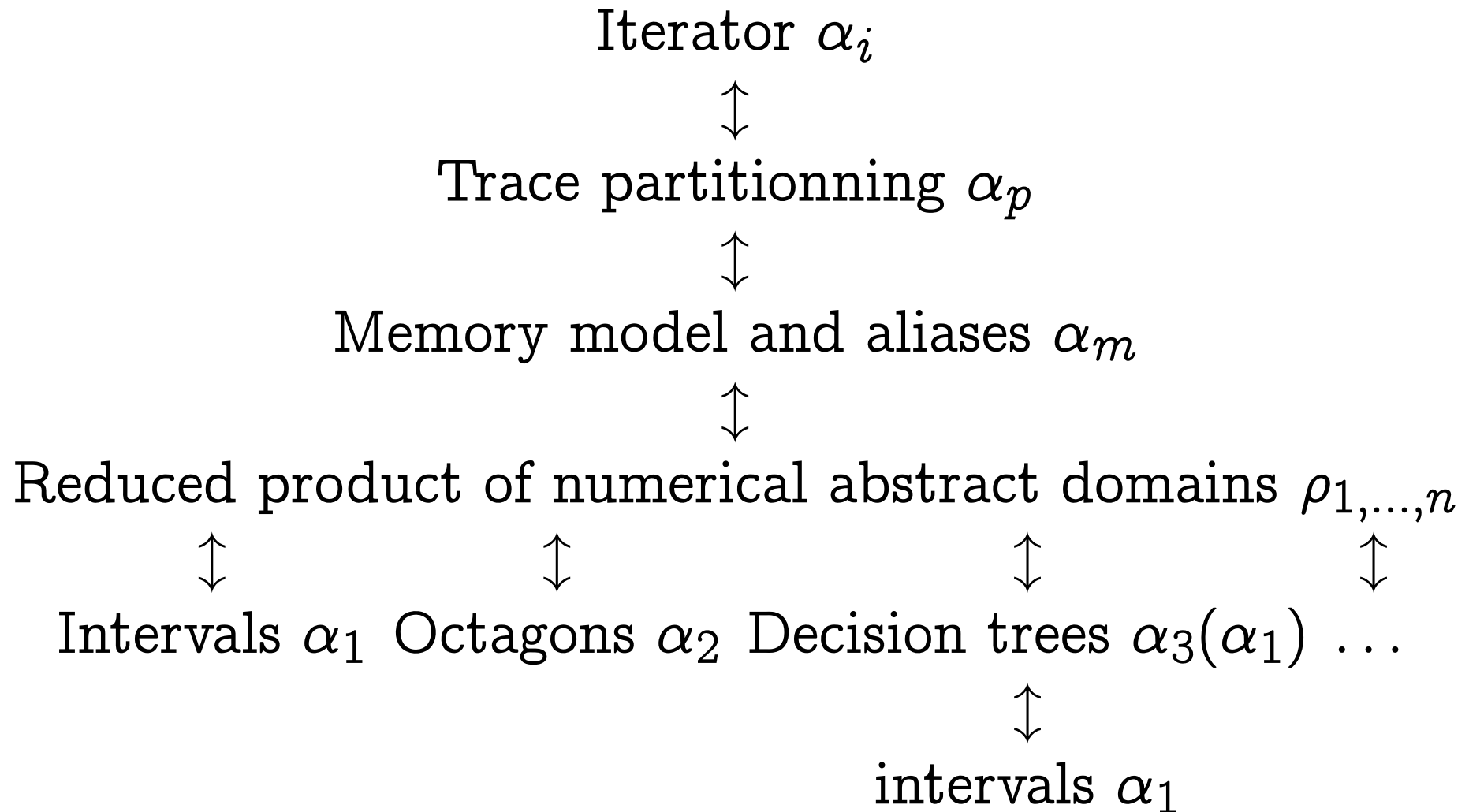
- Extremely complex  $\rightarrow$  divide and conquer using a reduced product [CC79]
- $\alpha = \rho_{1,\dots,n}(\alpha_i, \alpha_p, \alpha_m, \alpha_1, \dots, \alpha_i(\alpha_j) \dots, \alpha_n)$ , where
  - $\rho_{1,\dots,n}$  is the reduction,
  - $\alpha_i$  is the trace projection (to each context = call stack + program point),
  - $\alpha_p$  is the trace abstraction (trace partitionning [MR05]),
  - $\alpha_m$  is the state abstraction (memory model [Min06a]),
  - $\alpha_1, \dots, \alpha_n$  are the basic abstractions or abstraction functors on abstract variables  $\mathcal{X}$  (mutable, remanent)



# ASTRÉE's Architecture

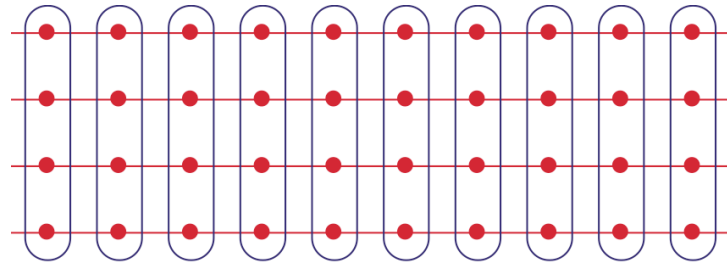


# The Abstract Interpreter

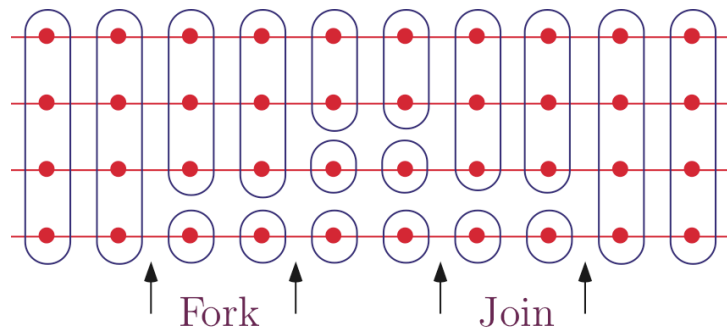


# Trace Partitionning Abstraction $\alpha_p$ [MR05]

State-based partitionning at control points:



Trace-based partitionning at control points:



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



# Trace Partitioning

## Principle:

- Semantic equivalence:

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



```
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

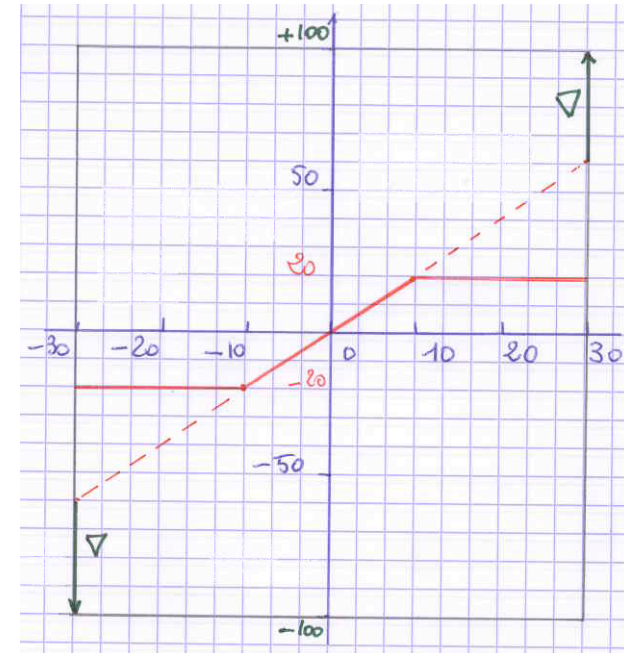




# Case analysis with loop unrolling

## – 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;
    __ASTREE_known_fact((( -30.0 <= x) && (x <= 30.0)));
    while ((i < 3) && (x >= t[i+1])) {
        i = i + 1;
    }
    r = (x - t[i]) * c[i] + d[i];
    __ASTREE_log_vars((r));
}
```



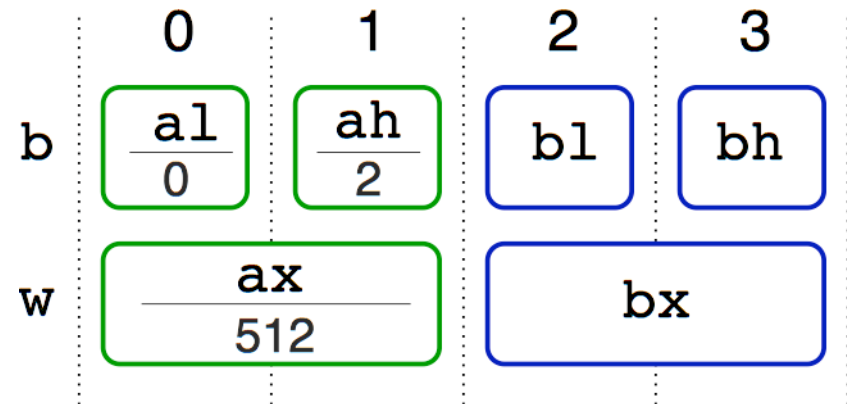
```
% astree -exec-fn main -no-trace -no-relational trace-partitioning.c |& egrep "(WARN)|(r in)"
direct = <float-interval: r in [-20, 20] >
%
% astree -exec-fn main -no-partition -no-trace -no-relational trace-partitioning.c \
|& egrep "(WARN)|(r in)"
direct = <float-interval: r in [-100, 100] >
%
```



## State Abstraction (Memory Model) $\alpha_m$ [Min06a]

The union type, pointer arithmetics and pointer transtyping is handled by allowing **aliasing at the byte level** [1]:

```
union {  
  struct { uint8  al,ah,b1,bh; } b;  
  struct { uint16 ax,bx; } w;  
} r;  
r.w.ax = 0; r.b.ah = 2;
```



- A box (auxiliary variable) in  $\mathcal{X}$  for each offset and each scalar type
- intersection semantics for overlapping boxes

### Reference

- [1] A. Miné. Field-Sensitive Value Analysis of Embedded C Programs with Union Types and Pointer Arithmetics. In *LCTES '2006*, pp. 54–63, June 2006, ACM Press.



## Maximal Abstraction $\alpha_1$

- The verification condition (ultimate phase of ASTRÉE) includes the test

$$\exists \bar{I} \in \alpha(\mathcal{D}[[P]]) : \dots \wedge \gamma(\bar{I}) \subseteq S$$

(in the abstract) and so the abstract domain  $\alpha(\mathcal{D}[[P]])$  should contain all possible  $S \in \mathcal{S}[[P]]$

- In ASTRÉE  $\mathcal{S}[[P]]$  is the abstract domain of intervals [CC76] (plus  $\neq 0$ )



## Choice of abstractions $\alpha_2, \dots, \alpha_n$ in ASTRÉE

The other abstract domains  $\alpha_2, \dots, \alpha_n$  can be chosen thanks to parameters when launching ASTRÉE, for example:

```
/* Launching the forward abstract interpreter */  
/* Domains: Guard domain, and Boolean packs (based on Absolute  
value equality relations, and Symbolic constant propagation  
(max_depth=20), and Linearization, and Integer intervals, and  
congruences, and bitfields, and finite integer sets, and Float  
intervals), and Octagons, and High_passband_domain(10), and  
Second_order_filter_domain (with real roots)(10), and  
Second_order_filter_domain (with complex roots)(10), and  
Arithmetico-geometric series, and new clock, and Dependencies  
(static), and Equality relations, and Modulo relations, and  
Symbolic constant propagation (max_depth=20), and Linearization,  
and Integer intervals, and congruences, and bitfields, and  
finite integer sets, and Float intervals. */
```



## Reduction [CC79, CCF<sup>+</sup>08]

Example: reduction of intervals [CC76] by simple congruences [Gra89]

```
% cat -n congruence.c
```

```
1 /* congruence.c */
2 int main()
3 { int X;
4   X = 0;
5   while (X <= 128)
6     { X = X + 4; };
7   __ASTREE_log_vars((X));
8 }
```

```
% astree congruence.c -no-relational -exec-fn main |& egrep "(launched)|(WA
direct = <integers (intv+cong+bitfield+set): X in {132} >
```

Intervals :  $X \in [129, 132]$  + congruences :  $X = 0 \bmod 4 \implies X \in \{132\}$ .



# Refinement Strategies



## Cost/Precision Ratio Adjustment

- We prefer **coarse abstractions** (for scalability, this excludes e.g. polyhedra)
- We anticipate the need for **necessary refinements** (for precision)



# Abstraction/Refinement

- **Parameterized refinement**: choose abstractions which precision can be refined/coarsened thanks to
  - manual parametrization
  - manual directives
  - automated directives
- **Unexpected refinement**: add a new abstract domain (and reduction)





# Parameterized Refinement



## Termination

SLAM uses CEGAR and does not terminate<sup>10</sup> on

```
% cat slam.c
int main() { int x, y;
  x = 0; y = 0;
  while (x < 2147483647)
    { x = x + 1; y = y + 1; }
  __ASTREE_assert((x == y));
}
```

whereas ASTRÉE uses widening/narrowing-based extrapolation techniques to prove the assertion

```
% astree -exec-fn main slam.c |& egrep "WARN"
%
```

---

<sup>10</sup> CEGAR cannot generate the invariant  $y = x - 1$  so produces all counter examples  $x = i + 1 \wedge y = i$ ,  $i = 0, 1, 2, 3, \dots$



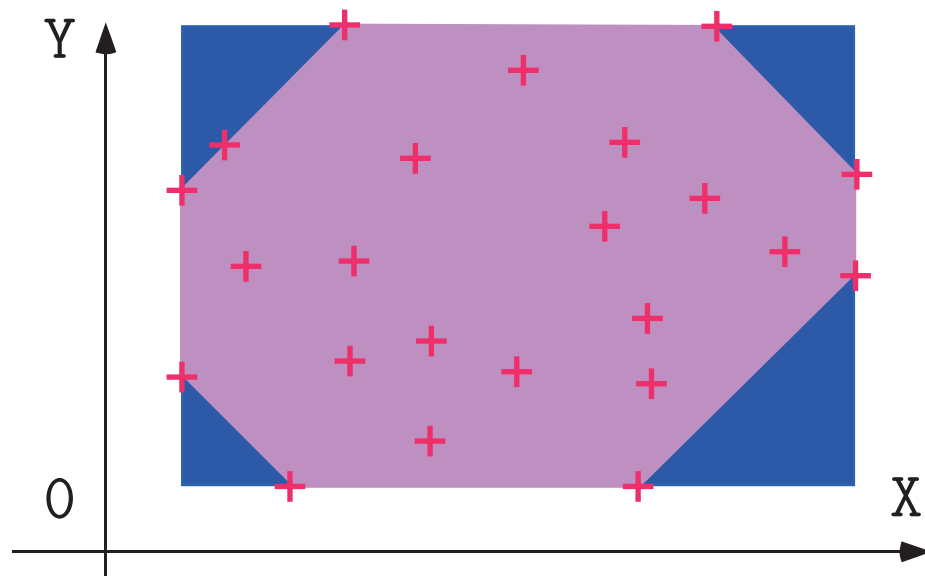
## Parameterized Abstraction e.g. Array Smashing

`--smash-threshold  $n$`  (400 by default)

smash elements of arrays of size  $> n$ , otherwise individualize array elements (each handled as a simple variable).



# Parameterized Abstract Domains: Intervals and Octagons



Intervals [CC76]:

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

Octagons [Min01]:

$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 20 \\ x - y \leq 07 \end{cases}$$

**Difficulties:** many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [CC77, Min01, Min04a]



# Parameterized Widening e.g. Intervals

## Thresholds for integer widening:

```
let widening_sequence =  
  [ of_int 0; of_int 1; of_int 2; of_int 3; of_int 4; of_int 5;  
    of_int 32767; of_int 32768; of_int 65535; of_int 65536;  
    of_string "2147483647"; of_string "2147483648"; of_string "4294967295" ]
```

## Thresholds for float widening:

```
let widening_sequence =  
  [ neg 1.; neg 0.15; neg 0.1; neg 0.01; neg 0.001; neg 0.000001; 0.; 0.000001; 0.001; 0.01; 1.;  
    1e1; 1e2; 1e3; 1e4; 70000.25; 1e5; 1e6; 1.5e6; 2e6; 2.5e6; 3e6; 3.5e6; 4e6; 4.5e6; 5e6; 5.5e6;  
    6e6; 6.5e6; 7e6; 7.5e6; 8e6; 8.5e6; 9e6; 9.5e6; 1e7; 10000020.; 1.5e7; 2e7; 2.5e7; 3e7; 3.5e7;  
    4e7; 4.5e7; 5e7; 5.5e7; 6e7; 6.5e7; 7e7; 7.5e7; 8e7; 8.5e7; 9e7; 9.5e7; 1e8; 1e9; 1e10; 1e11; 1e12;  
    1e15; 1e18; 1e20; 1e22; 1e25; 1e28; 1e30; 1e32]
```

**Delayed widenings:** `--forced-union-iterations-at-beginning`  
 $n$  (2 by default)

**Enforced widenings:** `--forced-widening-iterations-after`  $n$  (250  
by default), ...), etc.



# Parameterized Octagons

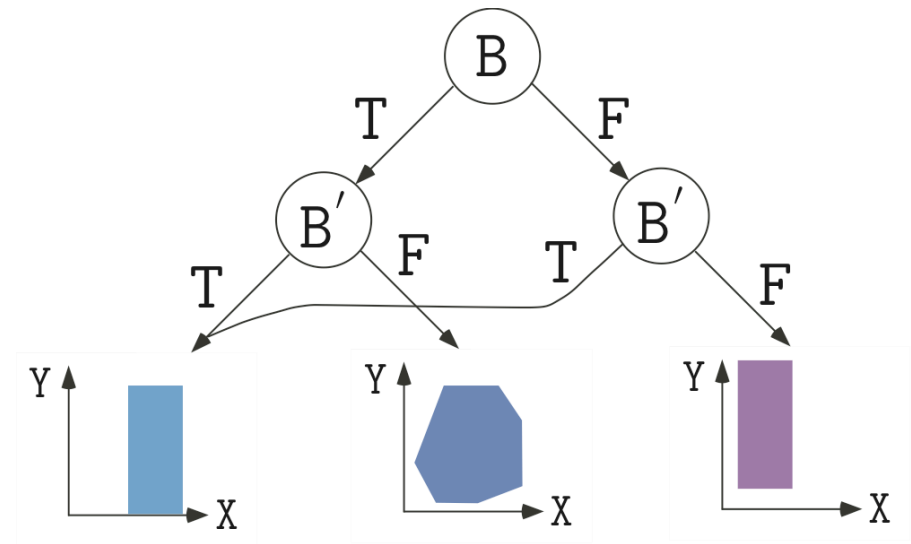
- Using octagons on all numerical variables would **not scale up**
- The analysis is parameterized by “**packs of variables**” stating which independent subsets of the variables should be related (everywhere, at which program points, in which context, ...)
- **Automatic packing** by another analysis (e.g. pre-analysis, on the fly, etc). In ASTRÉE pre-analysis at the block level.
- **Parameters** can modify the choice of packs **globally** (e.g. `--fewer-oct: no packs at the function level`, `--max-array-size-in-octagons  $n$` : unsmashed array elements of size  $> n$  don't go to octagons packs)
- **Directives** can modify the choice of packs **locally**:  
`(__ASTREE_octagon_pack(( $V_1, \dots, V_n$ )));`



# Decision Trees 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



## Parameterized Decision Trees

- Using decision trees on all variables would **not scale up**
- The analysis is parameterized by “**packs of variables**” stating which booleans go in nodes and numerical variables in leaves
- **Automatic packing** by a simple dependence analysis: Candidates for packing in a decision tree are the boolean variables to which a boolean expression is assigned or which are involved in a test as well as the non-volatile and non-constant variables which depend directly or indirectly on such a boolean
- **Parameters** can modify the choice of packs **globally** (e.g. `--max-bool-var  $n$ ,  $n = 3$  by default`)
- **Directives** can modify the choice of packs **locally** to state which boolean variables to put in internal nodes and numerical variables to put in abstract domains at the leaves `__ASTREE_boolean_pack`
- TODO: partition on small values (other than booleans)





## Example of directive

```
% cat repeat1.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
    int x = 100; BOOL b = TRUE;

    while (b) {
        x = x - 1;
        b = (x > 0);
    }
}

% astree -exec-fn main repeat1.c |& egrep "WARN"
repeat1.c:5.8-13::[call#main@2:loop@4>=4:]: WARN: signed int arithmetic
range [-2147483649, 2147483646] not included in [-2147483648, 2147483647]
%
```



## Example of directive (Cont'd)

```
% cat repeat2.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
    int x = 100; BOOL b = TRUE;
    __ASTREE_boolean_pack((b,x));
    while (b) {
        x = x - 1;
        b = (x > 0);
    }
}

% astree -exec-fn main repeat2.c |& egrep "WARN"
%
```

The insertion of this directive could be automated in ASTRÉE (if the considered family of programs has “repeat” loops).



# Parameterized Loop Partitionning

- No loop unrolling a priori
- Unrolling is controlled by parameters `--unroll u` and directives

`__ASTREE_partition_control((p)) while (B) {C}; C'; __ASTREE_partition_merge(());`

so that the analysis is semantically equivalent to:

$$\bigcup_{i=0}^p \left( (B; C)^i; \neg B; C' \right)$$

partitionning of the first  $p$  iterations

$$\cup \left( \bigcup_{i=0}^p (B; C)^i \right); \left( \bigcup_{i=p+1}^u \left( (B; C)^i; \neg B \right) \right); C'$$

semantic unrolling of the next  $u - p$  iterations

$$\cup \left( \bigcup_{i=0}^u (B; C)^i \right); \left( \bigtriangleup_{i=u+1}^{+\infty} (B; C)^i \right); \neg B; C'$$

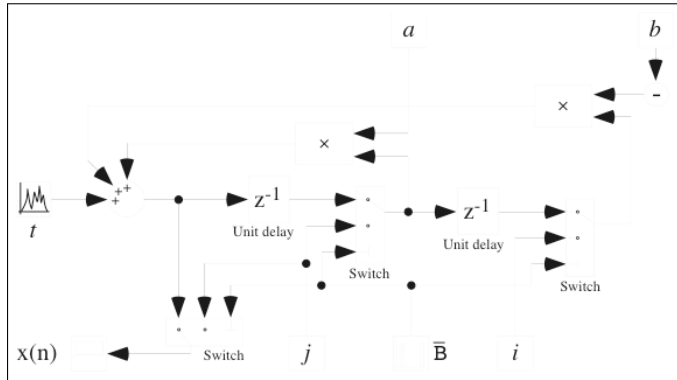
next iterations with widening



# Unexpected Refinement

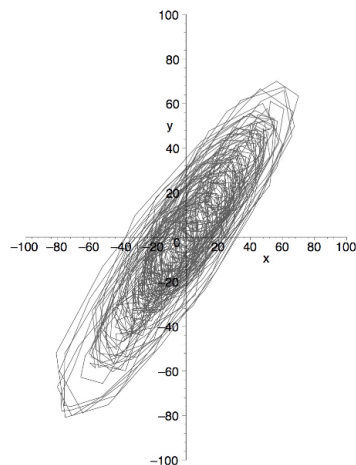


## 2<sup>d</sup> Order Digital Filter:

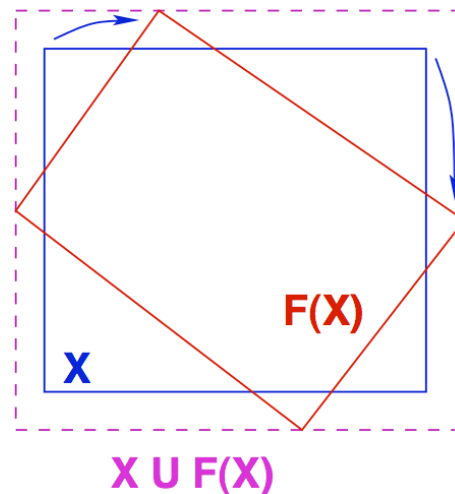


## Ellipsoid Abstract Domain for Filters

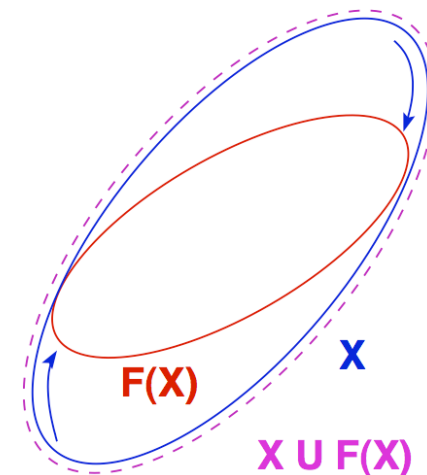
- Computes  $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is **bounded**, which must be proved in the abstract.
- There is **no stable interval or octagon**.
- The simplest stable surface is an **ellipsoid**.



execution trace



unstable interval



stable ellipsoid



## Filter Example [Fer04]

```
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 [Fer05] Example 1

```
% 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
```



# Arithmetic-Geometric Progressions [Fer05] (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; }
        else { R = 0; }
        T = (R >= 100);
        __ASTREE_wait_for_clock(());
    }
}
```

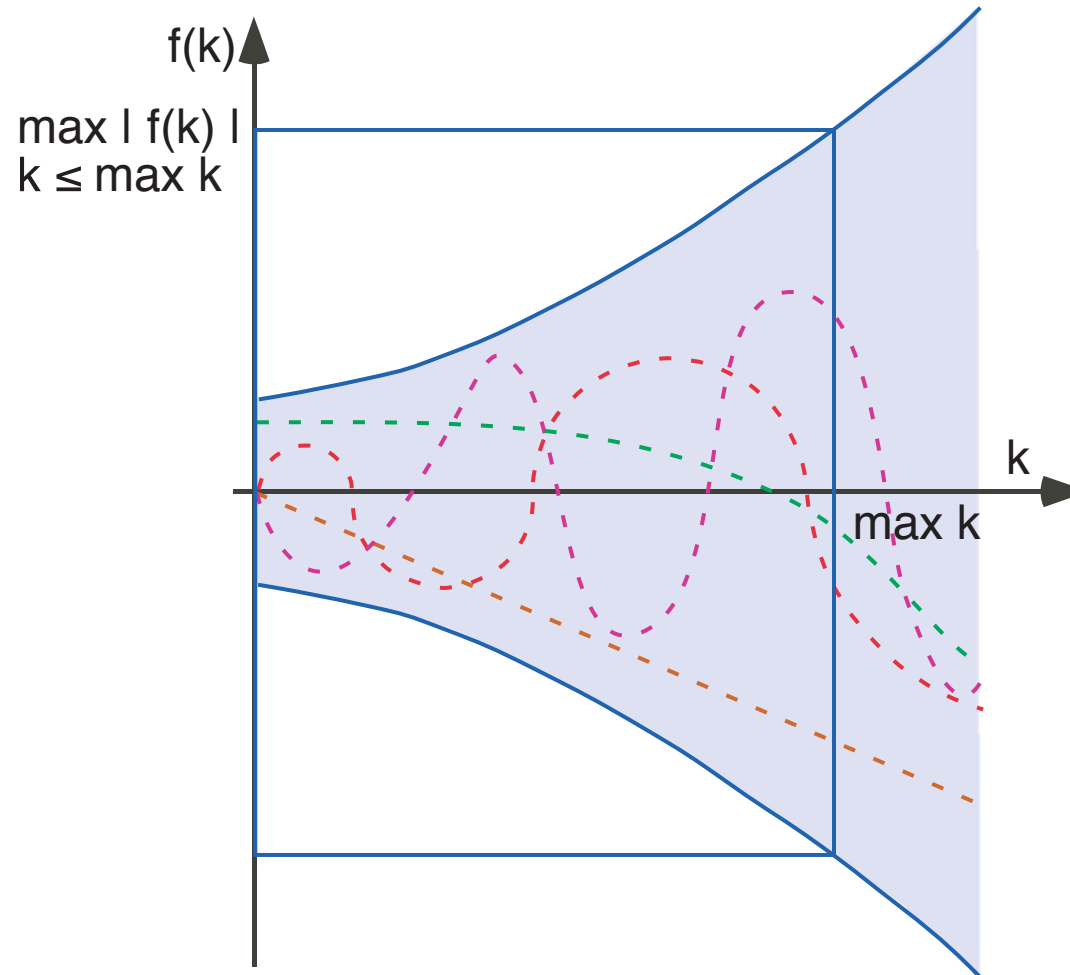
← potential overflow!

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





# Overapproximation with an Arithmetic-Geometric Progression



## Arithmetic-geometric progressions<sup>11</sup> [Fer05]

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

– Concretization:

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

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

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

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

---

### References

- <sup>11</sup> [2] J. Ferret. The arithmetic-geometric progression abstract domain. In *VMCAI'05*, Paris, LNCS 3385, pp. 42–58, Springer, 2005.



# Obsolete Abstraction

## Clock Abstract Domain

### 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.



## Incompleteness

ASTRÉE does not know that

$$\forall x, y \in \mathbb{Z} : 7y^2 - 1 \neq x^2$$

so on the following program

```
void main() { int x, y;  
    if ((-4681 < y) && (y < 4681) && (x < 32767) && (-32767 < x) && ((7*y*y - 1) == x*x))  
        { y = 1 / x; };  
}
```

it produces a **false alarm** (surely forever in this unrealistic program!)

```
% astree -exec-fn main false-alarm.c |& egrep "WARN"  
false-alarm.c:5.9-14::[call#main@1:]: WARN: integer division by zero ([-32766, 32766]  
and {1} / Z)  
%
```



## 6. Conclusion



## Characteristics of the ASTRÉE Analyzer

- Sound: – ASTRÉE is a **bug eradicator**: finds all bugs in a well-defined class (runtime errors)
- ASTRÉE is not a **bug hunter**: finding some bugs in a well-defined class (e.g. by *bug pattern detection* like FindBugs™, PREfast or PMD)
  - ASTRÉE is **exhaustive**: covers the whole state space ( $\neq$  MAGIC, CBMC)
  - ASTRÉE is **comprehensive**: never omits potential errors ( $\neq$  UNO, CMC from coverity.com) or sort most probable ones to avoid overwhelming messages ( $\neq$  Splint)



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

**Static:** compile time analysis ( $\neq$  run time analysis Rational Purify, Parasoftware 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), or PREfast (annotate functions with intended use)



## 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 Bandera, Bogor, Java PathFinder, Spin, VeriSoft)

**Efficient:** always terminate ( $\neq$  counterexample-driven automatic abstraction refinement BLAST, SLAM)





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

**Extensible/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 → it is a VERIFIER!



## The Future of the ASTRÉE Analyzer

- ASTRÉE has shown **usable and useful** in one industrial context (*electric flight control*):
  - as a R & D tool for A340 V2 and A380,
  - as a production tool for the A350 (?);
- **More applications** are forthcoming (ES\_PASS project);
- **Industrialization** is simultaneously under consideration;
- **Parallel programs** will even be more fun!



# THE END, THANK YOU



## 7. Bibliography



- [AGM93] G. Amato, F. Giannotti, and G. Mainetto. Data sharing analysis for a database programming language via abstract interpretation. In R. Agrawal, S. Baker, and D.A.Bell, editors, *Proc. 19<sup>th</sup> Int. Conf. on Very Large Data Bases*, pages 405–415, Dublin, IE, 24–27 Aug. 1993. MORGANKAUFMANN.
- [BCC<sup>+</sup>02] 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, invited chapter. In T. Mogensen, D.A. Schmidt, and I.H. Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pages 85–108. Springer, 2002.
- [BCC<sup>+</sup>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. In *Proc. ACM SIGPLAN '2003 Conf. PLDI*, pages 196–207, San Diego, CA, US, 7–14 June 2003. ACM Press.
- [BPC01] J. Bailey, A. Poulovassilis, and C. Courtenage. Optimising active database rules by partial evaluation and abstract interpretation. In *Proc. 8<sup>th</sup> Int. Work. on Database Programming Languages*, LNCS 2397, pages 300–317, Frascati, IT, 8–10 Sep. 2001. Springer.



- [BS97] V. Benzaken and X. Schaefer. Static integrity constraint management in object-oriented database programming languages via predicate transformers. In M. Aksit and S. Matsuoka, editors, *Proc. 11<sup>th</sup> European Conf. on Object-Oriented Programming, ECOOP '97*, LNCS 1241. Springer, Jyväskylä, FI, 9–13 June 1997.
- [CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. 2<sup>nd</sup> Int. Symp. on Programming*, pages 106–130, Paris, FR, 1976. Dunod.
- [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<sup>th</sup> POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.
- [CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6<sup>th</sup> POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.
- [CC92a] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proc. 4<sup>th</sup> Int. Symp. on PLILP '92*, Leuven, BE, 26–28 Aug. 1992, LNCS 631, pages 269–295. Springer, 1992.
- [CC92b] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *19<sup>th</sup> 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. 7<sup>th</sup> FPCA*, pages 170–181, La Jolla, CA, US, 25–28 June 1995. ACM Press.
- [CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *27<sup>th</sup> 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 interpretation. In *29<sup>th</sup> 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 *31<sup>st</sup> POPL*, pages 173–185, Venice, IT, 14–16 Jan. 2004. ACM Press.
- [CCF<sup>+</sup>05] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyser. In M. Sagiv, editor, *Proc. 14<sup>th</sup> ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 21–30. Springer, 2–10 Apr. 2005.





- [CCF<sup>+</sup>07] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Varieties of static analyzers: A comparison with ASTRÉE, invited paper. In M. Hinchey, He Jifeng, and J. Sanders, editors, *Proc. 1<sup>st</sup> TASE '07*, pages 3–17, Shanghai, CN, 6–8 June 2007. IEEE Comp. Soc. Press.
- [CCF<sup>+</sup>08] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the ASTRÉE static analyzer. In M. Okada and I. Satoh, editors, *11<sup>th</sup> ASIAN 06*, pages 272–300, Tokyo, JP, 6–8 Dec. 2006, 2008. LNCS 4435, Springer.
- [CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5<sup>th</sup> POPL*, pages 84–97, Tucson, AZ, 1978. ACM Press.
- [Cou97] P. Cousot. Types as abstract interpretations, invited paper. In *24<sup>th</sup> POPL*, pages 316–331, Paris, FR, Jan. 1997. 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.



- [Cou07] P. Cousot. Proving the absence of run-time errors in safety-critical avionics code, invited tutorial. In *Proc. 7<sup>th</sup> ACM & IEEE Int. Conf. EMSOFT '2007*, pages 7–9. ACM Press, 2007.
- [Dan07] V. Danos. Abstract views on biological signaling. In *Mathematical Foundations of Programming Semantics, 23<sup>rd</sup> Annual Conf. (MFPS XXIII)*, 2007.
- [DS07] D. Delmas and J. Souyris. ASTRÉE: from research to industry. In G. Filé and H. Riis-Nielsen, editors, *Proc. 14<sup>th</sup> Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 437–451. Springer, 22–24 Aug. 2007.
- [Fer04] J. Feret. Static analysis of digital filters. In D. Schmidt, editor, *Proc. 30<sup>th</sup> ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 33–48. Springer, Mar. 27 – Apr. 4, 2004.
- [Fer05] J. Feret. The arithmetic-geometric progression abstract domain. In R. Cousot, editor, *Proc. 6<sup>th</sup> Int. Conf. VMCAI 2005*, pages 42–58, Paris, FR, 17–19 Jan. 2005. LNCS 3385, Springer.
- [FHL<sup>+</sup>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. In T.A. Henzinger and C.M. Kirsch, editors, *Proc. 1<sup>st</sup> Int. Work. EMSOFT '2001*, volume 2211 of *LNCS*, pages 469–485. Springer, 2001.



- [GM04] R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *31<sup>st</sup> POPL*, pages 186–197, Venice, IT, 2004. ACM Press.
- [Gra89] P. Granger. Static analysis of arithmetical congruences. *Int. J. Comput. Math.*, 30:165–190, 1989.
- [JP06] Ph. Jorrand and S. Perdrix. Towards a quantum calculus. In *Proc. 4<sup>th</sup> Int. Work. on Quantum Programming Languages, ENTCS*, 2006.
- [Mau04] L. Mauborgne. ASTRÉE: Verification of absence of run-time error. In P. Jacquart, editor, *Building the Information Society*, chapter 4, pages 385–392. Kluwer Acad. Pub., 2004.
- [Min] A. Miné. The Octagon abstract domain library.  
<http://www.di.ens.fr/~mine/oct/>.
- [Min01] A. Miné. A new numerical abstract domain based on difference-bound matrices. In O. Danvy and A. Filinski, editors, *Proc. 2<sup>nd</sup> Symp. PADO '2001*, Århus, DK, 21–23 May 2001, LNCS 2053, pages 155–172. Springer, 2001.
- [Min04a] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In D. Schmidt, editor, *Proc. 30<sup>th</sup> ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 3–17. Springer, Mar. 27 – Apr. 4, 2004.



- [Min04b] A. Miné. *Weakly Relational Numerical Abstract Domains*. Thèse de doctorat en informatique, École polytechnique, Palaiseau, FR, 6 Dec. 2004.
- [Min05] A. Miné. Weakly relational numerical abstract domains: Theory and application, invited paper. In *1<sup>st</sup> Int. Work. on Numerical & Symbolic Abstract Domains, NSAD '05*, Maison Des Polytechniciens, Paris, FR, 21 Jan. 2005.
- [Min06a] A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *Proc. LCTES '2006*, pages 54–63. ACM Press, June 2006.
- [Min06b] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.
- [Min06c] A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In E.A. Emerson and K.S. Namjoshi, editors, *Proc. 7<sup>th</sup> Int. Conf. VMCAI 2006*, pages 348–363, Charleston, SC, US, 8–10, Jan. 2006. LNCS 3855, Springer.
- [Mon05] D. Monniaux. The parallel implementation of the ASTRÉE static analyzer. In *Proc. 3<sup>rd</sup> APLAS '2005*, pages 86–96, Tsukuba, JP, 3–5 Nov. 2005. LNCS 3780, Springer.



- [MR05] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. In M. Sagiv, editor, *Proc. 14<sup>th</sup> ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 5–20. Springer, Apr. 2005.
- [PCJD07] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. Semantics-based approach to malware detection. In *34<sup>th</sup> 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.
- [Riv05a] X. Rival. Abstract dependences for alarm diagnosis. In *Proc. 3<sup>rd</sup> APLAS '2005*, pages 347–363, Tsukuba, JP, 3–5 Nov. 2005. LNCS 3780, Springer.
- [Riv05b] X. Rival. Understanding the origin of alarms in ASTRÉE. In C. Hankin and I. Siveroni, editors, *Proc. 12<sup>th</sup> 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. 30<sup>th</sup> ESOP '04*, volume 2986 of *LNCS*, pages 18–32, Barcelona, ES, Mar. 29 – Apr. 2 2004. Springer.



- [RT06] F. Ranzato and F. Tapparo. Strong preservation of temporal fixpoint-based operators by abstract interpretation. In A.E. Emerson and K.S. Namjoshi, editors, *Proc. 7<sup>th</sup> Int. Conf. VMCAI 2006*, pages 332–347, Charleston, SC, US, 8–10 Jan. 2006. LNCS 3855 , Springer.
- [SD07] J. Souyris and D. Delmas. Experimental assessment of ASTRÉE on safety-critical avionics software. In F. Saglietti and N. Oster, editors, *Proc. Int. Conf. on Computer Safety, Reliability, and Security (textscSafecom 2007)*, volume Nuremberg, DE, LNCS 4680, pages 479–490. Springer, 18–21 Sep. 2007.
- [Sou04] J. Souyris. Industrial experience of abstract interpretation-based static analyzers. In P. Jacquart, editor, *Building the Information Society*, chapter 4, pages 393–400. Kluwer Acad. Pub., 2004.

