Numerical Domains for Software Verification by Abstract Interpretation

Patrick Cousot

First International Workshop on Numerical Abstractions for Software Verification

Princeton, NJ , July 8th, 2008

Abstract

Abstract interpretation allows the automatic inference of numerical and symbolic invariants about program behaviors. The nature of the invariants is captured by fixpoint inference using a combination of abstract domains. We discuss and compare integer, real and floating-point numerical abstractions used in software verification by abstract interpretation. The considered abstractions extend from simple non-relational ones to more complex relational linear or non-linear abstractions. The six central issues to be considered are:

- adequacy of the concrete semantics (choice of primitives, mathematical versus computer-based definition, etc),
- nature of the abstraction (relational, non-relational, parametric, etc.) and its computer encoding,
- soundness of the abstraction and the abstract verification conditions (so that no later inference can lead to erroneous conclusions such as false positives),
- correct and effective induction (that is how inductive invariants are computed by concrete/abstract iteration with extrapolation, constraint solving, abstraction/refinement, elimination, exact resolution, etc),
- ability to scale up (for programs in the small to programs in the large, separate abstractions, etc), and
- precision of the abstraction (to avoid false alarms).

In conclusion we discuss the limitations restraining the practical use of the currently available abstractions and how they could be overcome.

Major difficulties for numerical software verification

- Machine integers are not mathematical integers $\mathbb{Z}$
- Machine floats are not mathematical reals $\mathbb{R}$

1. Numerical bugs

Integer bugs

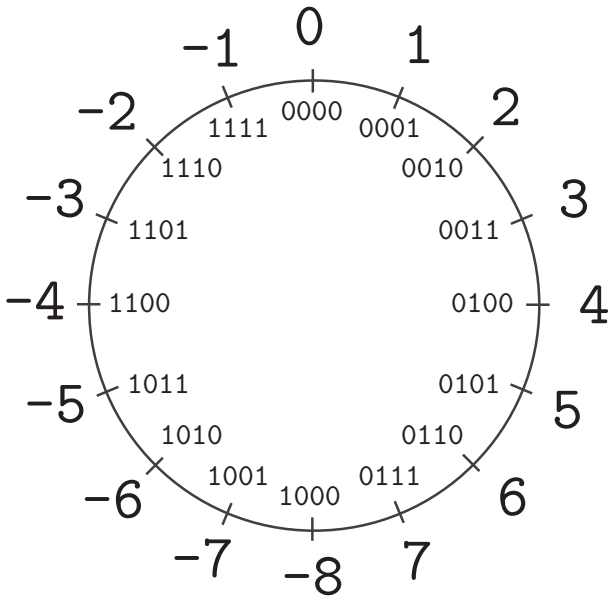
The factorial program (fact.c)

```
#include <stdio.h>
int fact (int n ) {
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {
    r = r*i;
  }
  return r;
}
int main() { int n;
  scanf("%d",&n);
  printf("%d!=%d\n",n,fact(n));
}
```

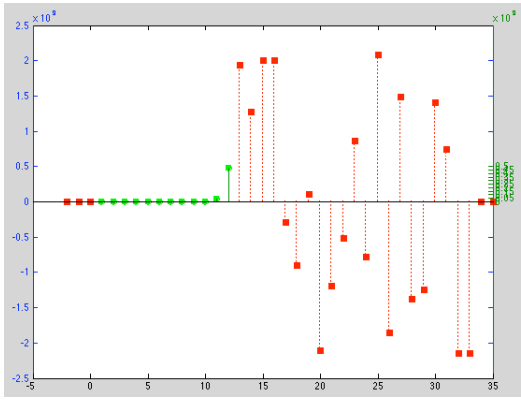
```
% gcc fact.c -o fact.exec
% ./fact.exec
3
3!  = 6
% ./fact.exec
4
4!  = 24
% ./fact.exec
100
100!  = 0
% ./fact.exec
20
20!  = -2102132736
```


Modular integer arithmetics

- Computers use integer modular arithmetics on n bits (where $n = 16$, 32, 64, etc)
- Example of an integer representation on 4 bits (in *complement to two*) :



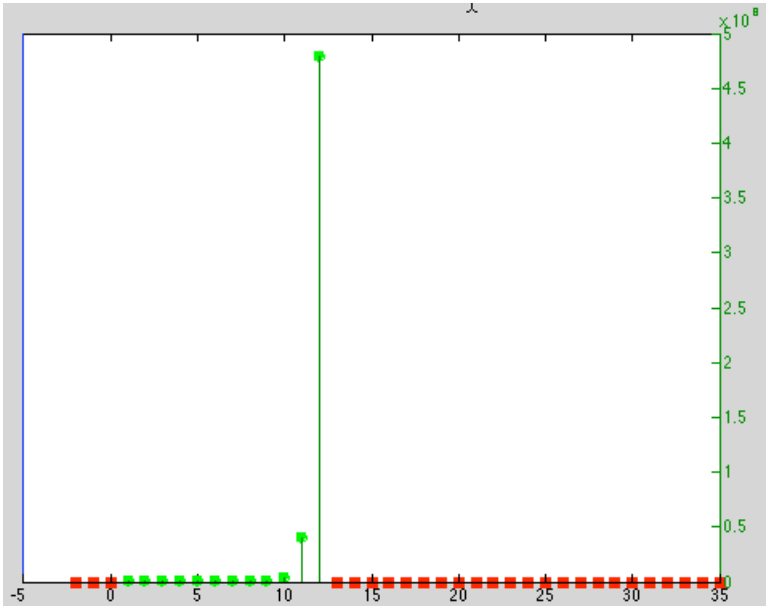
- Only integers between -8 and 7 can be represented on 4 bits
- We get $7 + 2 = -7$



Proof of absence of runtime error by static analysis

```
% cat -n fact_lim.c
 1 int MAXINT = 2147483647;
 2 int fact (int n) {
 3     int r, i;
 4     if (n < 1) || (n = MAXINT) {
 5         r = 0;
 6     } else {
 7         r = 1;
 8         for (i = 2; i<=n; i++) {
 9             if (r <= (MAXINT / i)) {
10                 r = r * i;
11             } else {
12                 r = 0;
13             }
14         }
15     }
16     return r;
17 }
18
19 int main() {
20     int n, f;
21     f = fact(n);
22 }
% astree -exec-fn main fact_lim.c |& grep WARN
%
```

→ No alarm!



Float bugs

Floats

- *Floating point numbers* are a finite subset of the *rationals*
- For example one can represent 32 floats on 6 bits, the 16 positive normalized floats spread as follows on the line:

- When real computations do not spot on a float, one must *round* the result to a close float

Example of rounding error (1)

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

```
#include <stdio.h>
int main() {
  double x, a; float y, z;
  x = 112589973951488.0;
  a = 1.0;
  y = (x+a);
  z = (x-a);
  printf("%f\n", y-z);
}
```

```
% gcc arrondi1.c -o arrondi1.exec
% ./arrondi1.exec
134217728.000000
%
```

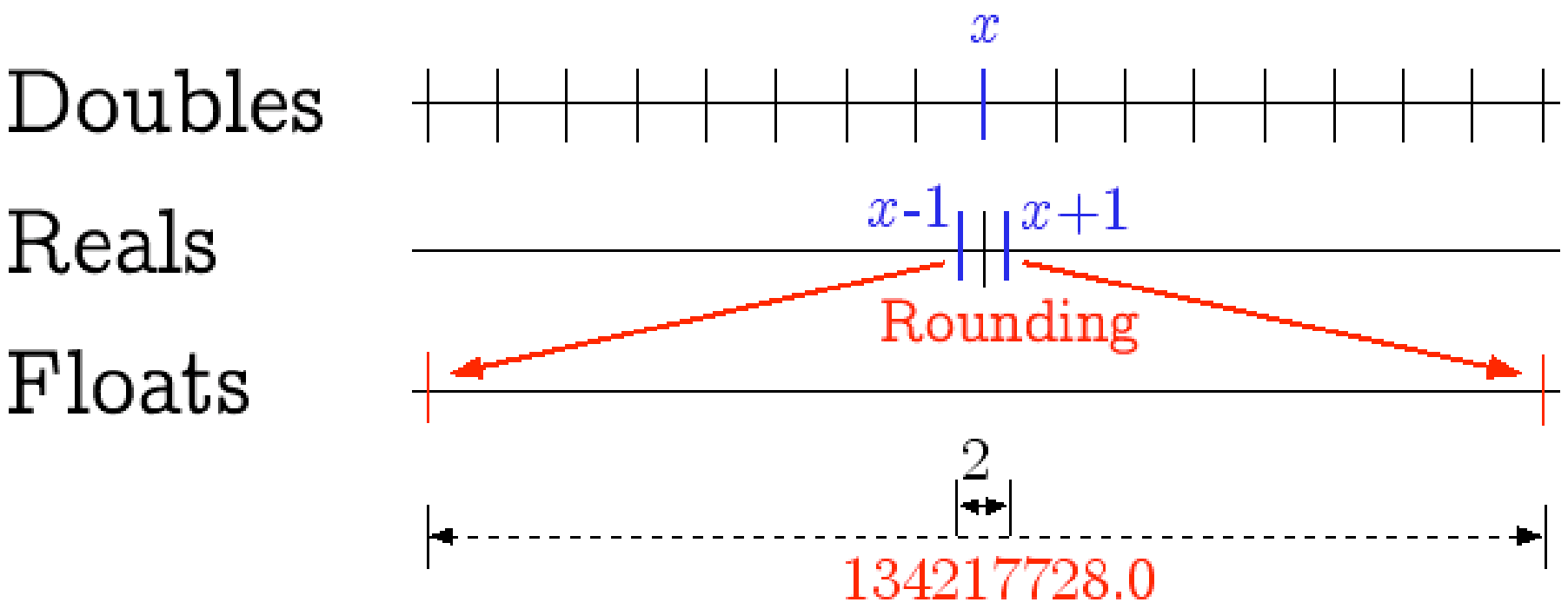

Example of rounding error (2)

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

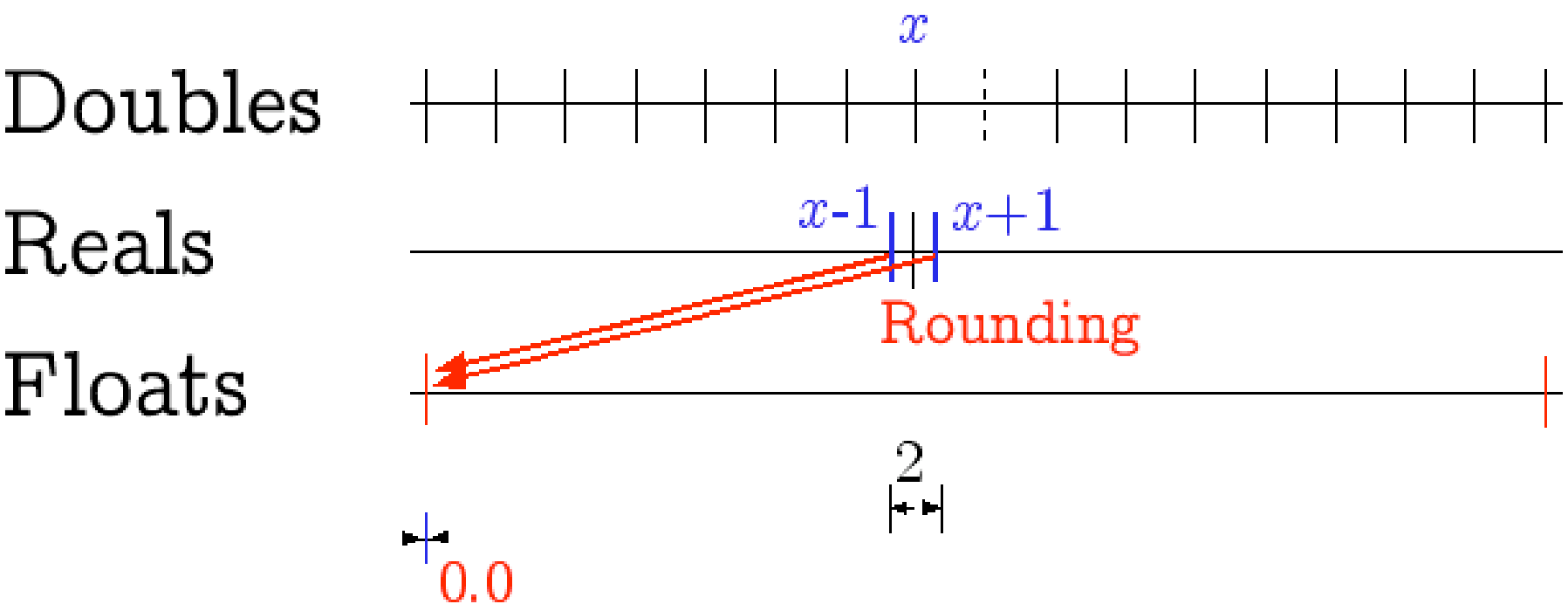
```
#include <stdio.h>
int main() {
  double x, a; float y, z;
  x = 1125899973951487.0;
  a = 1.0;
  y = (x+a);
  z = (x-a);
  printf("%f\n", y-z);
}
```

```
% gcc arrondi2.c -o arrondi2.exec
% ./arrondi2.exec
0.000000
%
```


Rounding (1)



Rounding (2)

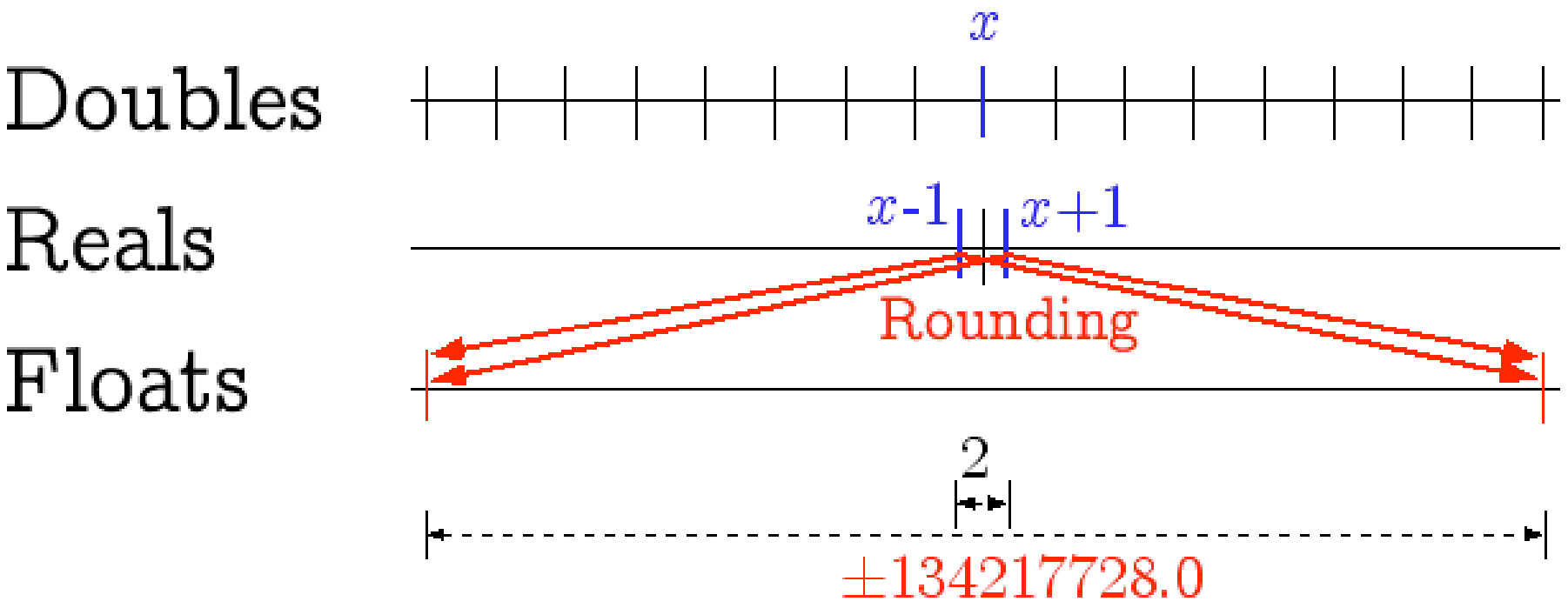


Proof of absence of runtime error by static analysis

```
% cat -n arrondi3.c
     1 int main() {
     2     double x; float y, z, r;;
     3     x = 1125899973951488.0;
     4     y = x + 1;
     5     z = x - 1;
     6     r = y - z;
     7     __ASTREE_log_vars((r));
     8 }
% astree -exec-fn main -print-float-digits 10 arrondi3.c \
  |& grep "r in "
direct = <float-interval:  r in [-134217728, 134217728] >
```
(1)

(1) Astrée considers the worst rounding case (towards $+\infty$, $-\infty$, 0 or to the nearest) whence the possibility to obtain -134217728.

The verification is done in the worst case



Examples of bugs due to rounding errors

- The patriot missile bug missing Scuds in 1991 because of a software clock incremented by $\frac{1}{10}^{\text{th}}$ of a seconde ($(0,1)_{10} = (0,0001100110011001100\ldots)_2$ in binary)
- The Exel 2007 bug : 77.1×850 gives 65,535 but displays as 100,000! (2)

2	65535-2^(-37)	100000		65536-2^(-37)	100001
3	65535-2^(-36)	100000		65536-2^(-36)	100001
4	65535-2^(-35)	100000		65536-2^(-35)	100001
5	65535-2^(-34)	65535		65536-2^(-34)	65536
6	65535-2^(-36)-2^(-37)	100000		65536-2^(-36)-2^(-37)	100001
7	65535-2^(-35)-2^(-37)	100000		65536-2^(-35)-2^(-37)	100001
8	65535-2^(-35)-2^(-36)	100000		65536-2^(-35)-2^(-36)	100001
9	65535-2^(-35)-2^(-36)-2^(-37)	65535		65536-2^(-35)-2^(-36)-2^(-37)	65536

(2) Incorrect float rounding which leads to an alignment error in the conversion table while translating 64 bits IEEE 754 floats into a Unicode character string. The bug appears exactly for six numbers between 65534.99999999995 and 65535 and six between 65535.99999999995 and 65536.

2. Software verification

Major difficulties for software verification: undecidability and complexity

- The mathematical proof problem is undecidable [3]
- Even assuming finite states, the complexity is much too high for combinatorial exploration to succeed
- Example: 1.000.000 lines $\times$ 50.000 variables $\times$ 64 bits $\simeq 10^{27}$ states
- Exploring 10^{15} states per seconde, one would need 10^{12} s $>$ 300 centuries (and a lot of memory)!

[3] there are infinitely many programs for which a computer cannot solve them in finite time even with an infinite memory.

How to fight undecidability and complexity?

- Unsoundness : test, bug finding analysis, bounded model checking, ...
- Incompleteness : static analysis by abstract interpretation

The difficulty of scaling up

- The abstraction must be coarse enough to be effectively computable with reasonable resources
- The abstraction must be precise enough to avoid false alarms
- Abstractions to *infinite domains with widenings* are more expressive than abstractions to *finite domains* [4] (when considering the analysis of a programming language) [CC92]
- Abstractions are ultimately incomplete (even intrinsically for some semantics and specifications [CC00])

[4] e.g. predicate abstraction which always abstract to a finite domain.

3. Abstract-interpretation-based static analysis [1]

Reference

[1] 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 Joseph Fourier, Grenoble. 1978.

1) Design of a concrete model

- Operational semantics: programs $\rightarrow$ infinite state transition system
- Collecting semantics: programs $\rightarrow$ strongest properties (e.g. finite/infinite execution traces, reachable states, ...)
- Fixpoint characterization of the collecting semantics:

$$C[\![P]\!] \triangleq \mathsf{lfp}^{\subseteq} F[\![P]\!]$$

- Semantic verification by fixpoint checking:

$$C[\![P]\!] \subseteq S \iff \mathsf{lfp}^{\subseteq} F[\![P]\!] \subseteq S$$

2) Design of an abstraction

– Abstraction [5]:
$\langle$concrete properties, $\subseteq\rangle \underset{\alpha[\![P]\!]}{\overset{\gamma[\![P]\!]}{\leftrightarrows}} \langle$abstract properties, $\sqsubseteq\rangle$

– Abstract semantics: abstraction of the concrete semantics

$$C[\![P]\!] \subseteq \gamma[\![P]\!](C^\sharp[\![P]\!])$$

– Fixpoint abstract semantics (by abstraction of the fixpoint characterization of the collecting semantics)

$$C^\sharp[\![P]\!] = \mathsf{lfp}^{\sqsubseteq} F^\sharp[\![P]\!]$$

(where $F[\![P]\!] \circ \gamma[\![P]\!] \subseteq \gamma[\![P]\!] \circ F^\sharp[\![P]\!]$).

[5] Note that the abstraction/concretization is always specific to a program, an often misundertsood point, a.o. [DKW08, Sec. C, p. 1170]

3) Static analysis

- Solve the abstract fixpoint constraint $F^\sharp[\![P]\!](X) \sqsubseteq X$ (so $\mathsf{lfp}^{\sqsubseteq} F^\sharp[\![P]\!] \sqsubseteq X$)
- In some cases, the solution is directly computable (e.g. by Gaussian elimination)
- Chaotic iteration (finite, ascending chain condition for $\sqsubseteq$, ...):

  $$X = \bot \text{ and then } X := F^\sharp[\![P]\!](X) \text{ until } X = F^\sharp[\![P]\!](X)$$

- Convergence acceleration for infinite abstractions:
  - widening: $X := X \nabla[\![P]\!] F^\sharp(X)$ until $F^\sharp[\![P]\!](X) \sqsubseteq X$
  - narrowing: $X := X \Delta[\![P]\!] F^\sharp(X)$ until $X = F^\sharp[\![P]\!](X)$
  - overapproximating limit X: $C^\sharp[\![P]\!] \sqsubseteq X$

Example: invariant computation by fixpoint approximation [CC77]

$\{y \geqslant 0\}$ $\leftarrow$ hypothesis

```
x = y
```

$\{I(x,y)\}$ $\leftarrow$ loop invariant

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

Floyd-Naur-Hoare verification conditions:

$$(y \geqslant 0 \wedge x = y) \Longrightarrow I(x,y) \qquad \textit{initialisation}$$

$$(I(x,y) \wedge x > 0 \wedge x' = x - 1) \Longrightarrow I(x',y) \qquad \textit{iteration}$$

Equivalent fixpoint equation:

$$I(x,y) \;=\; x \geqslant 0 \wedge (x = y \vee I(x+1,y)) \qquad (\textit{i.e. } I = F(I)\,^{(6)})$$

(6) We look for the most precise invariant I, implying all others, that is $\mathsf{lfp}^{\Rightarrow} F$.

Accelerated Iterates $I = \lim_{n \to \infty} F^n(\text{false})$

$$I^0(x,y) = \text{false}$$

$$I^1(x,y) = x \geqslant 0 \wedge (x = y \vee I^0(x+1,y)) = 0 \leqslant x = y$$

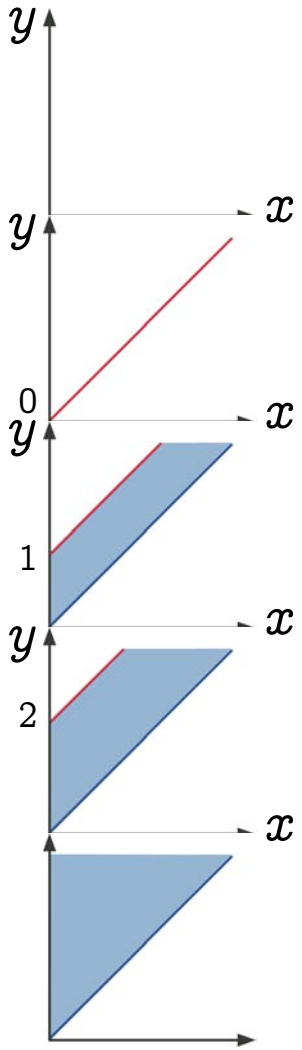
$$I^2(x,y) = x \geqslant 0 \wedge (x = y \vee I^1(x+1,y)) = 0 \leqslant x \leqslant y \leqslant x+1$$

$$I^3(x,y) = x \geqslant 0 \wedge (x = y \vee I^2(x+1,y)) = 0 \leqslant x \leqslant y \leqslant x+2$$

$$I^4(x,y) = I^2(x,y) \triangledown I^3(x,y) \leftarrow \text{widening} = 0 \leqslant x \leqslant y$$

$$I^5(x,y) = x \geqslant 0 \wedge (x = y \vee I^4(x+1,y)) = I^4(x,y) \quad \text{fixed point!}$$

The invariants are computer representable with octagons!



4) Sound abstract verification

– Assuming the specification S to be expressible in the abstract, i.e. $\alpha[\![P]\!] \circ \gamma[\![P]\!](S) = S$, the specification S is checked in the abstract (yes, no, I don't know):

$$X \sqsubseteq \alpha[\![P]\!](S) \Longrightarrow \gamma[\![P]\!](X) \subseteq S$$

– By soundness, holds in the concrete (no false negative):

$$C[\![P]\!] = \mathsf{lfp}^{\subseteq} F[\![P]\!] \subseteq \gamma(\mathsf{lfp}^{\sqsubseteq} F^\sharp[\![P]\!](X)) \subseteq \gamma[\![P]\!](X) \subseteq S$$

– By incompleteness, there may be false alarms (false positive : I don't know) i.e.

$$X \not\sqsubseteq \alpha[\![P]\!](S)$$

5) Refinement

- In case of false alarm, the abstraction α must be refined into a more precise α'
- The static analyzer must be designed to allow for easy refinements (e.g. manual refinement, sometimes human intelligence may help)

6) Modular abstraction

The abstract semantics is decomposed into:

– A structural fixpoint iterator (by composition on the program syntax)

– A collection of parametric abstract domains with:

  - parameters to adjust the expressivity of the abstraction
  - parametric convergence acceleration (parameters to adjust the frequence and precision of widenings/narrowings)
  - analysis directives (to locally adjust the choice of abstractions)

– A reduction performing the conjunction of the abstractions

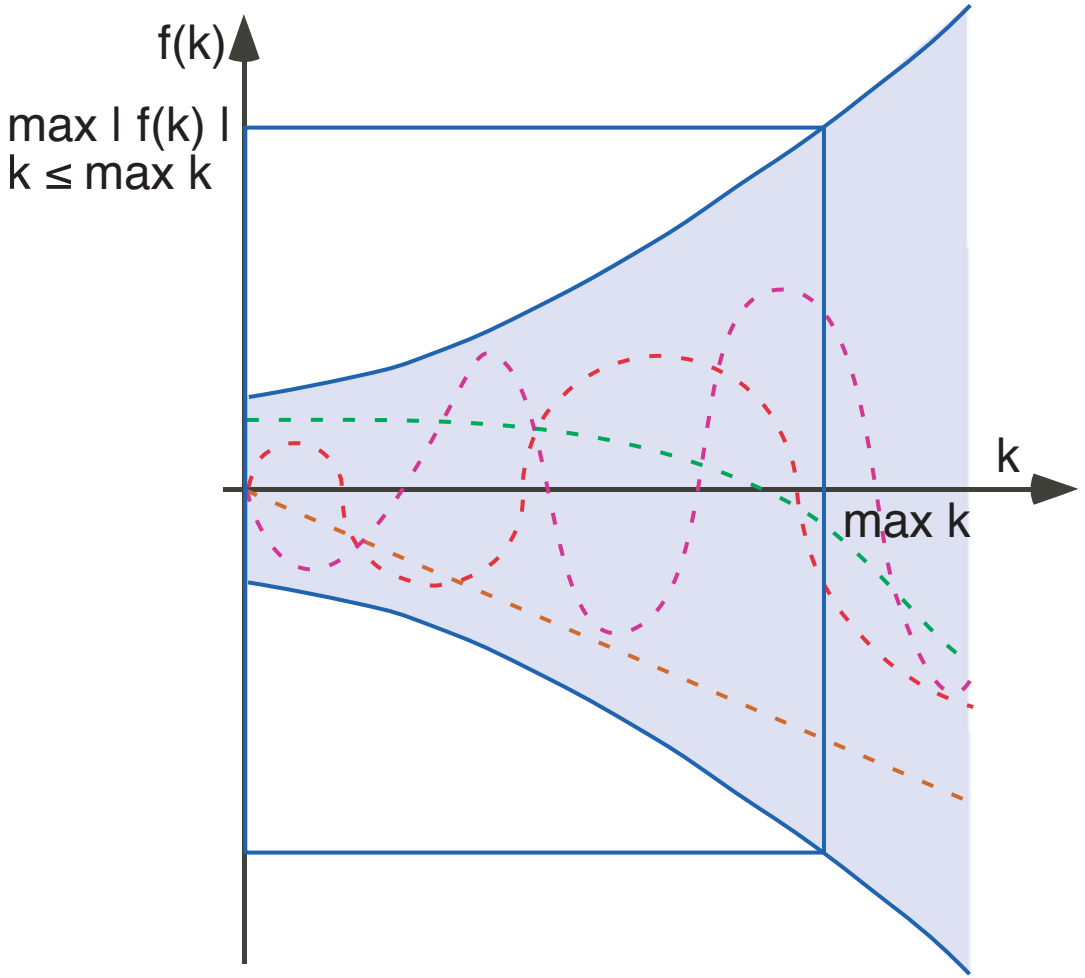
$\Rightarrow$ Easily extensible for refinement by addition of abstract domains!

Example: typical combination of abstractions in ASTRÉE

```
/* 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.  */
```


Example of abstract domain in ASTRÉE

Overapproximation with an arithmetico-geometric series:



Arithmetico-geometric series [7] [Fer05a]

– Abstract domain: $(\mathbb{R}^+)^5$

– Concretization:

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

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

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

Reference

[2] J. Feret. The arithmetic-geometric progression abstract domain. In *VMCAI'05*, Paris, LNCS 3385, pp. 42–58, Springer, 2005.

[7] here in $\mathbb{R}$ but must be implemented in the floats by appropriate roundings!

Arithmetic-geometric progressions (Example 1)

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
```

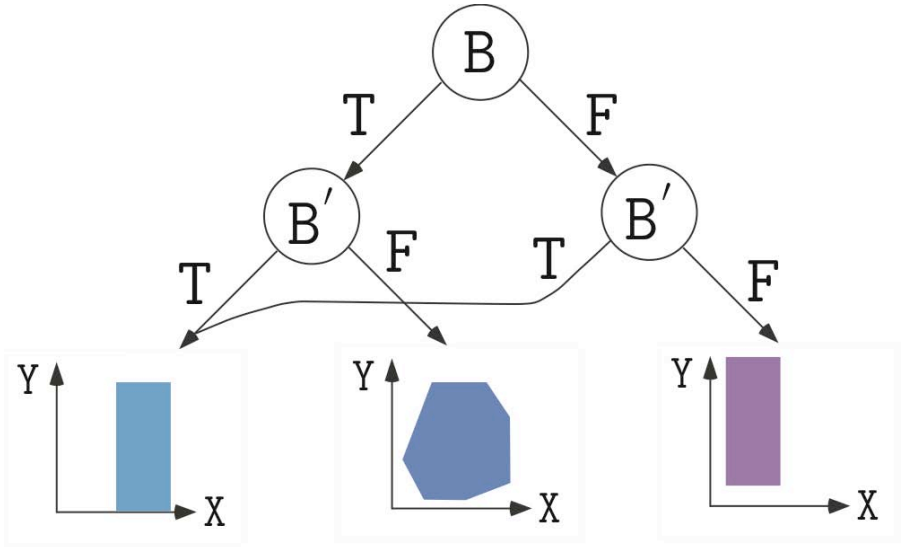
← potential overflow!

```
% 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.
```

Example of abstract domain functor in ASTRÉE: decision trees

– 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

4. Programs

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.

5. Specification

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).

6. Concrete semantics

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

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

```
y = x - (int) ((x-m)/(M-m))*(M-m);
```

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

```
% cat -n modulo.c
#include <stdio.h>
#include <math.h>
int main () {
   float m, M, x, y; M = 4095.0; m = -M;
   x = 4094.9997558593750; /* largest float strictly less than M */
   y = x - (int) ((x-m)/(M-m))*(M-m);
printf("%.20f\n",y);
}
% gcc modulo.c; ./a.out
-4095.00024414062500000000
%
```


Analysis by Astrée

```
% cat modulo-a.c
int main () {
   float m, M, x, y;
M = 4095.0; m = -M;
x = 4094.9997558593750; /* largest float strictly less than M */
   y = x - (int) ((x-m)/(M-m))*(M-m);
__ASTREE_log_vars((y));
}
% astree -exec-fn main -print-float-digits 25 modulo-a.c |& grep "y in"
direct = <float-interval: y in [-4095.000244140625, 4094.999755859375] >
%
```


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

Analysis by ASTRÉE

```
% cat -n unpreditable-a.c
     1  const int false = 0;
     2  int main () { int n, T[1], x;
     3  n = 1;
     4  x = T[n];
     5  __ASTREE_assert((false));
     6  }
% astree -exec-fn main unpreditable-a.c |& grep "WARN"
unpreditable-a.c:4.4-8::[call#main@2:]: WARN: invalid dereference: dereferencing
4 byte(s) at offset(s) [4;4] may overflow the variable T of byte-size 4
%
```

No alarm on `assert(false)` because execution is assumed to stop after a definite runtime error with unpredictable results [8].

[8] Equivalent semantics if no alarm.

Different Classes of Run-time Errors

1. Errors terminating the execution [9]. 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 [10]. ASTRÉE warns and continues with worst-case assumptions.
3. Errors not terminating the execution with unpredictable outcome [11]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.

$\Rightarrow$ ASTRÉE is sound with respect to C standard, unsound with respect to C implementation, unless no false alarm of type 3.

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

(10) e.g. overflows over signed integers resulting in some signed integer.

(11) e.g. memory corruptionss.

7. Control and memory abstraction in the iterator

Characterization of the iterator

- structural (by induction on the program syntax)
- flow sensitive (the execution order of statements is taken into account)
- path sensitive (distinguishes between feasible paths through a program)
- context sensitive (function calls are analyzed differently for each call site)
- interprocedural (function bodies are analyzed in the context of each respective call site)

Paths versus reachable states analysis

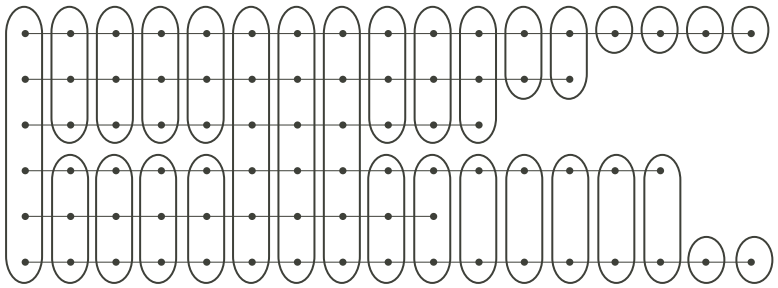
- The merge over all paths analysis is more precise than fixpoint reachable states analysis for non-distributive abstract domains (but more costly)
- The merge over all paths can be obtained in fixpoint form by disjunctive completion of the abstract domain [CC79]
- The disjunctive completion is costly (a terminating analysis such as constant propagation can become non terminating)

Reference

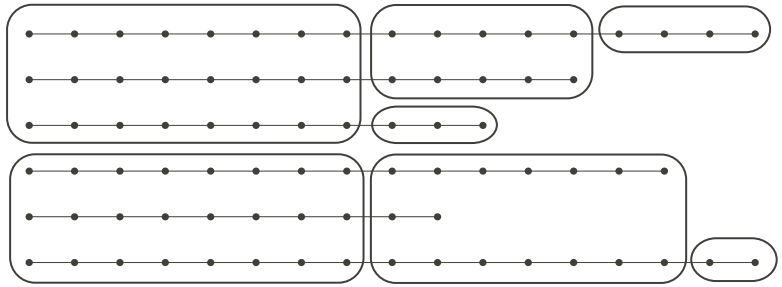
[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Trace partitioning

– State partitionning by program points [12]



– Trace partitionning [13]



– Trace partitionning abstract interpretation combines the effects of case analysis and symbolic execution [MR05, RM07]

(12) all reachable states corresponding to a given program point are over-approximated by a local invariant on memory states reachable at that program points

(13) portions of traces starting at a given program point for given memory values and finishing at a given program point are analyzed by an overapproximating abstract execution

Example of 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:

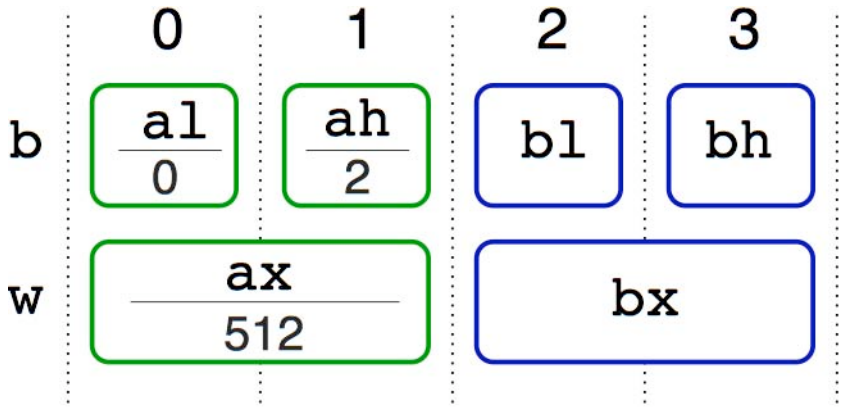
```
if (B)
  { X=0; Y=1; }
else
  { X=1; Y=0; }
R = 1 / (X-Y);
```

cannot result in a division by zero

Memory abstraction [Min06a]

The union type, pointer arithmetics and pointer transtyping is handled by allowing aliasing at the byte level [3]:

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



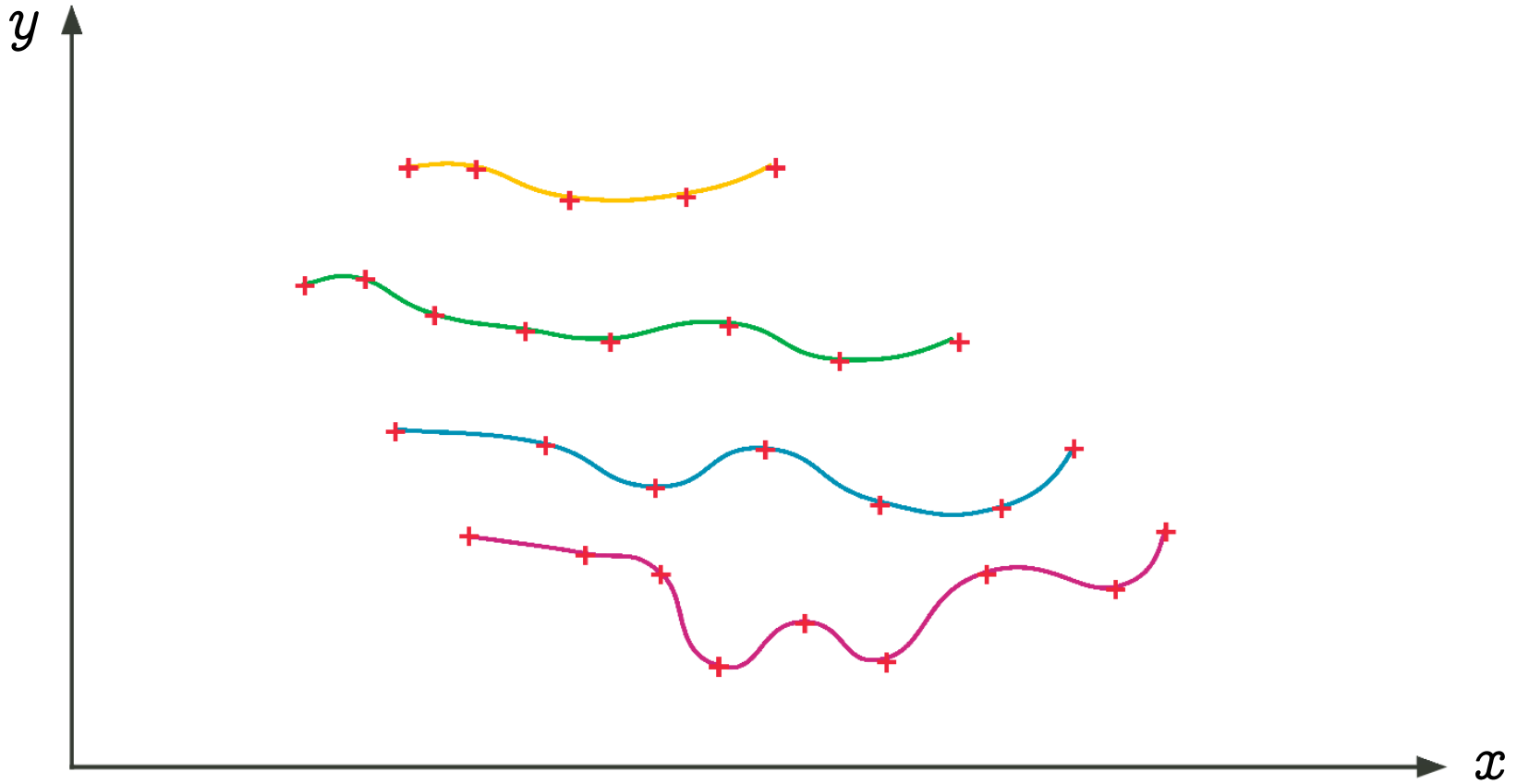
- An abstract variable (box) for each offset and each scalar type
- Intersection semantics for overlapping boxes
- Abstract domains handle separate mutable or cumulative abstract variables only!

Reference

[3] 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.

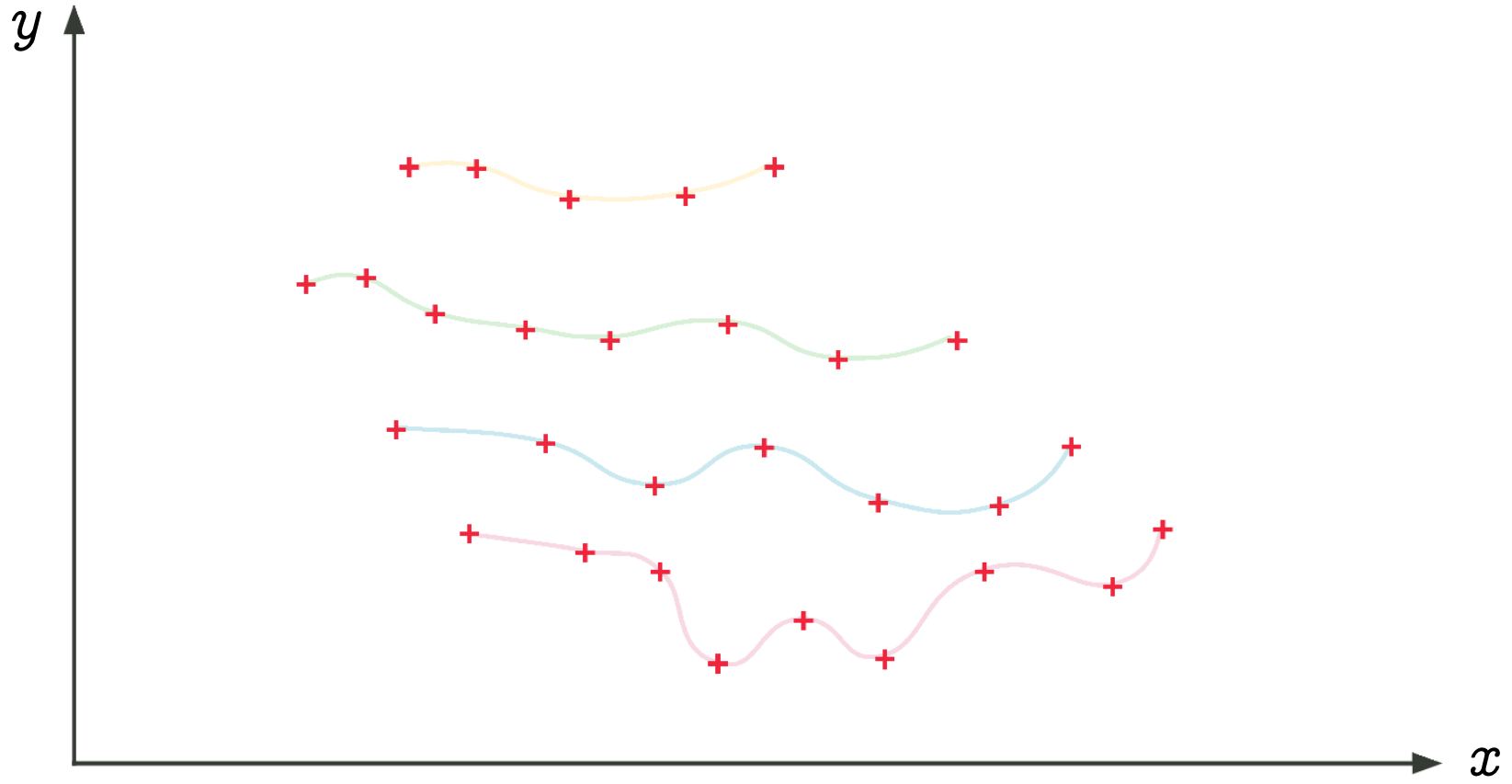
8. General purpose numerical abstractions

Semantics



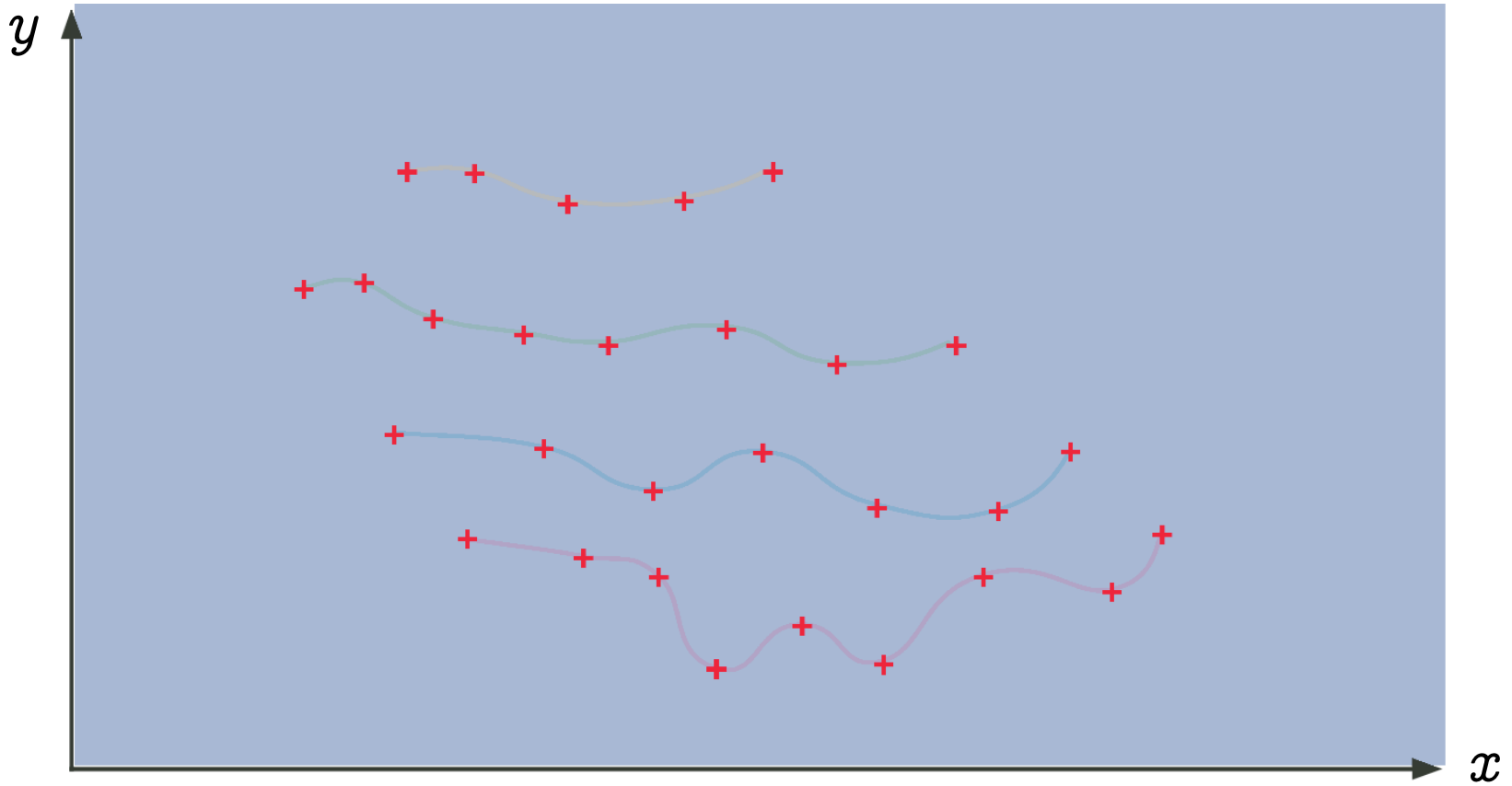
(Infinite) set of traces (finite ou infinite)

Abstraction to a set of states (invariant)



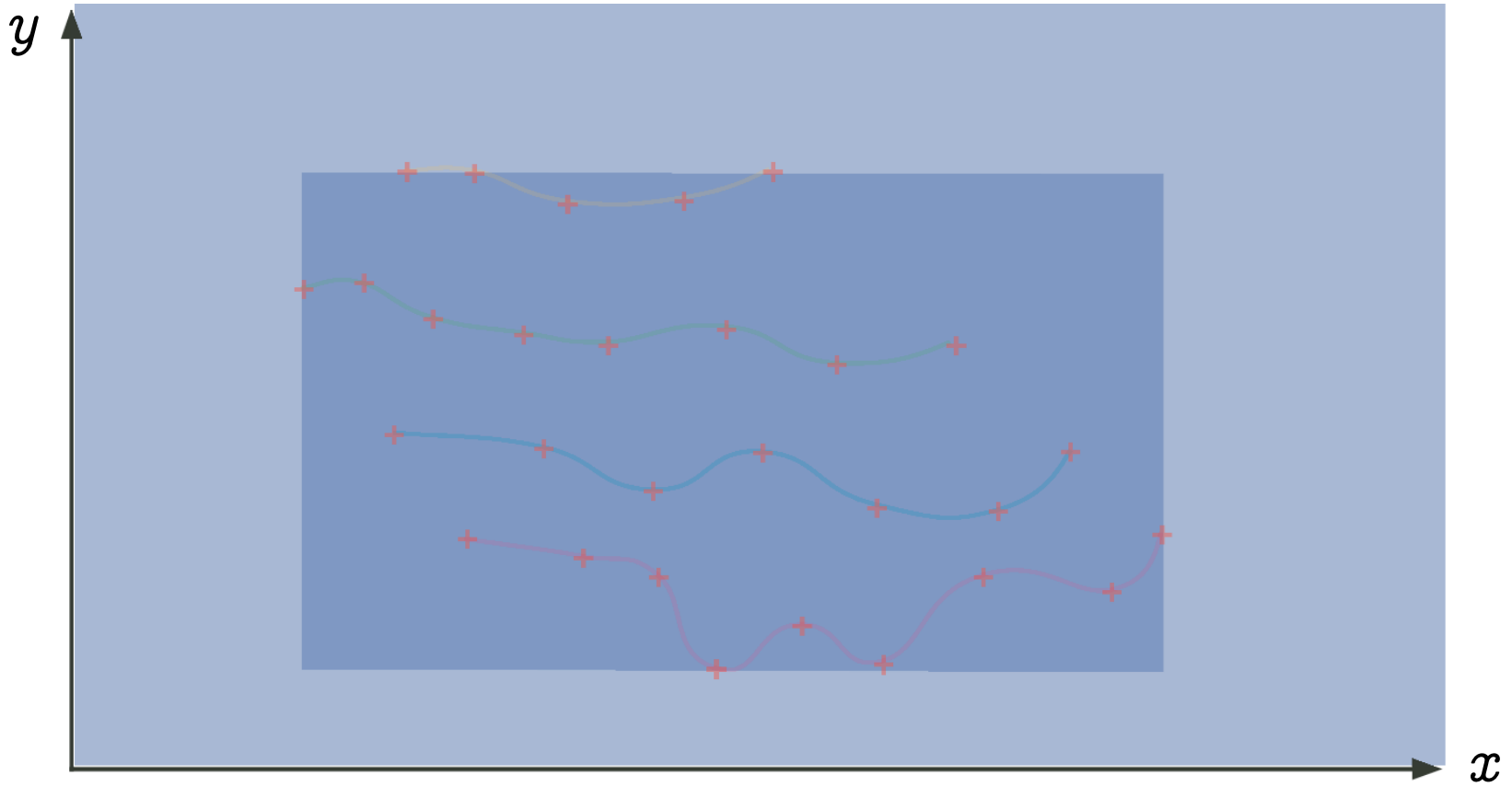
Set of points $\{(x_i, y_i) : i \in \Delta\}$, Floyd/Hoare/Naur invariance proof method [Cou02]

Abstraction by signs



Signs $x \geq 0$, $y \geq 0$ [CC79]

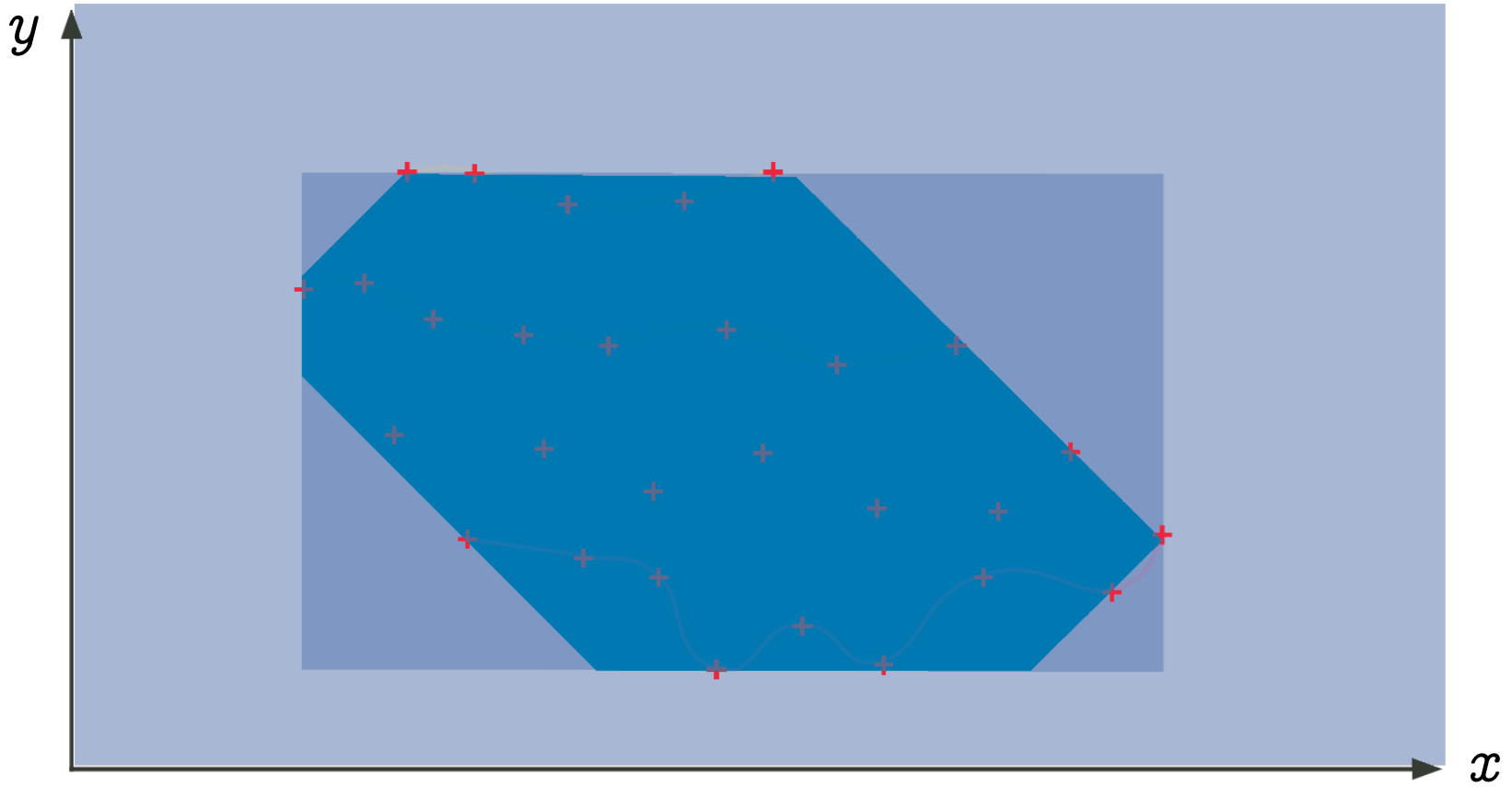
Abstraction by intervals



Intervals $a \leq x \leq b$, $c \leq y \leq d$ [CC77]

Sound implementation with floats!

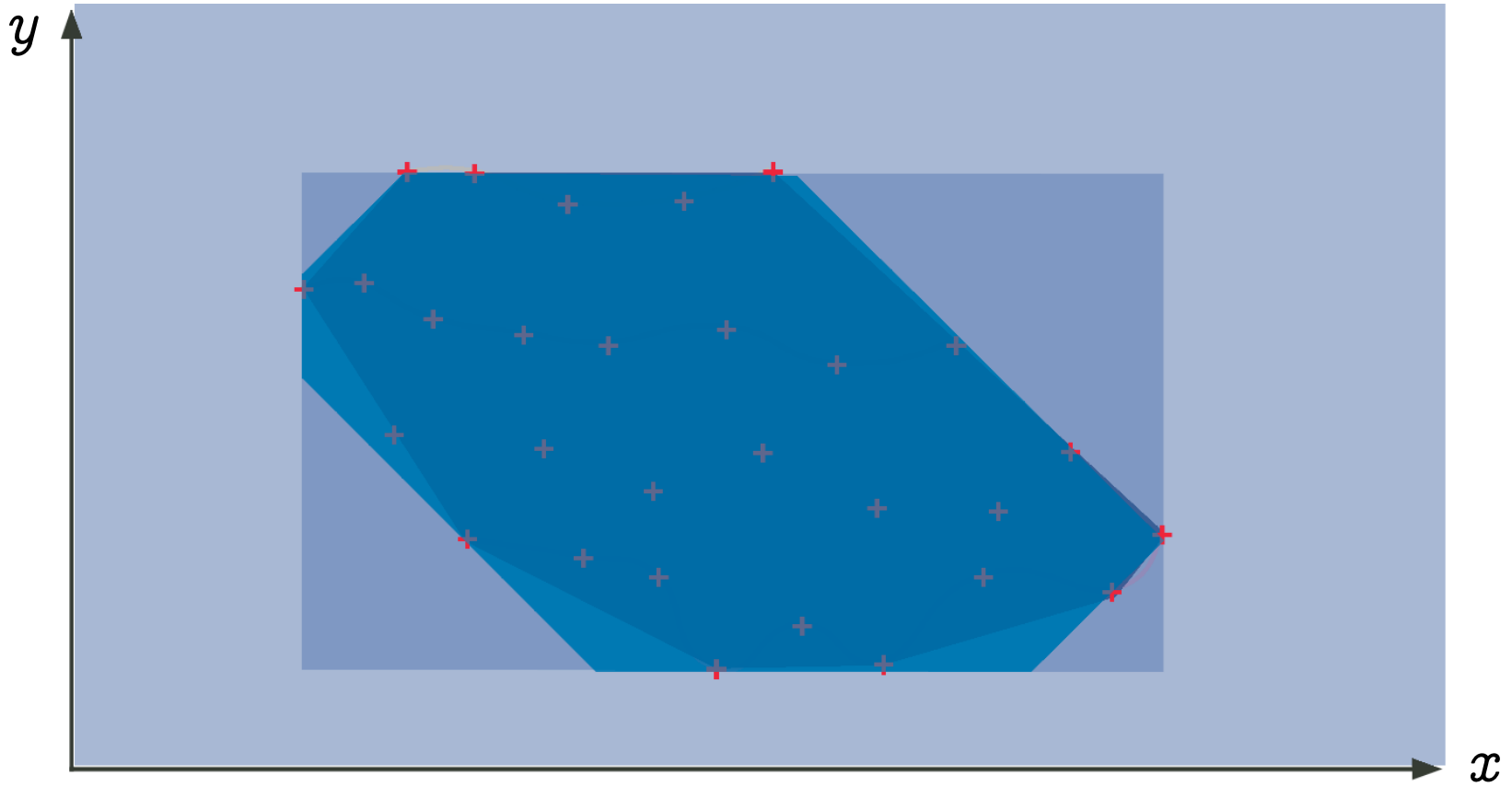
Abstraction by octagons



Octagons $x - y \leq a$, $x + y \leq b$ [Min06b]

Sound implementation with floats!

Abstraction by polyedra



Polyedra $a.x + b.y \leq c$ [CH78]

NEW Sound implementation with floats!

Floating-point linearization [Min04a, Min04b]

- 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

9. Reduction

Reduction [CC79, CCF+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 "(WARN)|(X in)"
direct = <integers (intv+cong+bitfield+set): X in {132} >
```

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

10. Refinement

Inexpressivity

- The weakest invariant to prove the specification may be inexpressible with the current reduced abstractions

  $\Rightarrow$ false alarms are unavoidable and cannot be solved
- No solution, but abstraction refinement!

Abstraction/refinement by tuning the cost/precision ratio in Astrée

– Approximate reduced product of a choice of coarsenable/refinable abstractions

– Tune their precision/cost ratio by

  - Globally by parametrization

  - Locally by (automatic) analysis directives

  so that the overall abstraction is not uniform.

11. Parametrization

Parameterized abstractions

– Parameterize the cost / precision ratio of abstractions in the static analyzer

– Examples:

- array smashing: `--smash-threshold` n (400 by default)
  $\rightarrow$ smash elements of arrays of size $> n$, otherwise individualize array elements (each handled as a simple variable).

- packing in octogons: (to determine which groups of variables are related by octagons and where)
  · `--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

Parameterized widenings

- Parameterize the rate and level of precision of widenings in the static analyzer
- Examples:
  - delayed widenings: `--forced-union-iterations-at-beginning` n (2 by default)
  - enforced widenings: `--forced-widening-iterations-after` n (250 by default)
  - thresholds for widening (e.g. for integers):

```
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" ]
```

12. Analysis directives

Analysis directives

– Require a local refinement of an abstract domain

– Example:

```
% 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 have been automated in ASTRÉE (if the considered family of programs had had "repeat" loops).

Automatic analysis directives

- The directives can be inserted automatically by static analysis
- Example:

```
% cat p.c
int clip(int x, int max, int min) {
 if (max >= min) {
  if (x <= max) {
   max = x;
  }
  if (x < min) {
   max = min;
  }
 }
 return max;
}
void main() {
 int m = 0; int M = 512; int x, y;
 y = clip(x, M, m);
  __ASTREE_assert(((m<=y) && (y<=M)));
}
% astree -exec-fn main p.c |& grep WARN
%
```

```
% astree -exec-fn main p.c -dump-partition
...
int (clip)(int x, int max, int min)
{
  if ((max >= min))
  {  __ASTREE_partition_control((0))
    if ((x <= max))
    {
      max = x;
    }
    if ((x < min))
    {
      max = min;
    }
    __ASTREE_partition_merge_last(());
  }
  return max;
}
...
%
```

13. Domain-specific abstractions

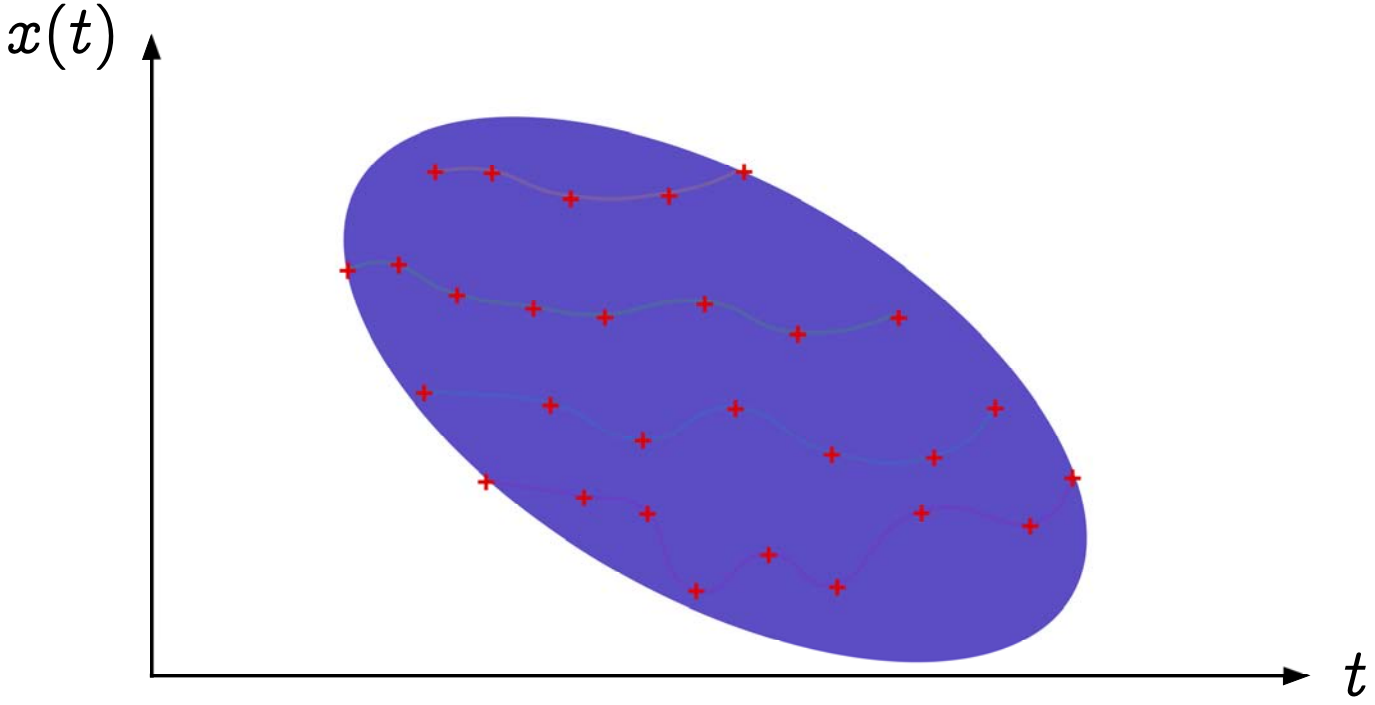
Adding new abstract domains

– In case of inexpressivity: add a new abstract domain:

  - representation of the (parameterized) abstract properties
  - abstract property transformers for language primitives
  - (parameterized) widening
  - reduction with other abstractions

– Examples : ellipsoids for filters [Fer05b], exponentials for accumulation of small rounding errors [Fer05a], quaternions, ...

Abstraction by ellipsoid for filters

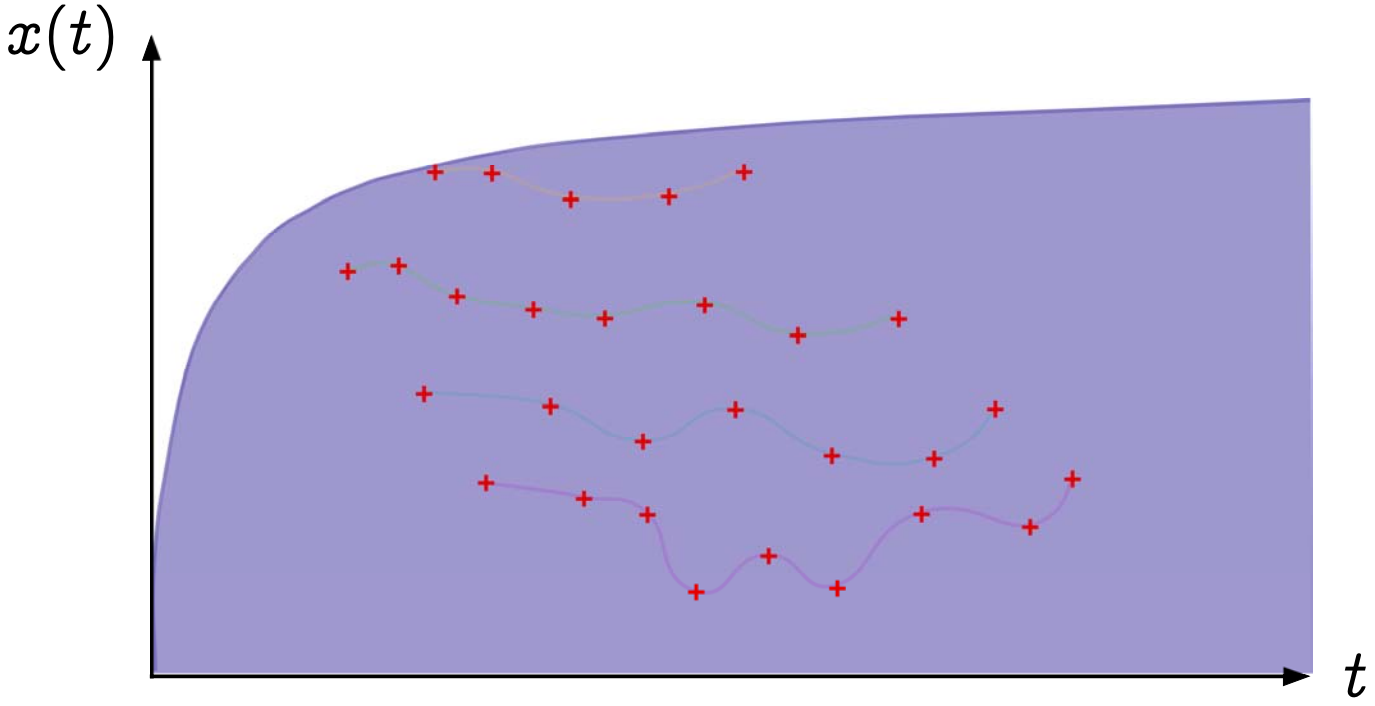


Ellipsoids $(x-a)^2+(y-b)^2 \leq c$ [Fer05b]

Example of analysis by ASTRÉE

```
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; }
}
```


Abstraction by exponentials for accumulation of small rounding errors



Exponentials $a^x \leq y$ [Fer05a]

Arithmetic-Geometric Progressions: Example 2

```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{ X=E;
  if (FIRST) { P = X; }
  else
    { P =  (P - ((((2.0 * P) - A) - B)
            * 4.491048e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
```

```
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1 +
1.19209290217e-07)^clock - 5.87747175411e-39
/ 1.19209290217e-07 <= 23.0393526881
```

14. Why not automatic abstraction completion?

Abstraction completion

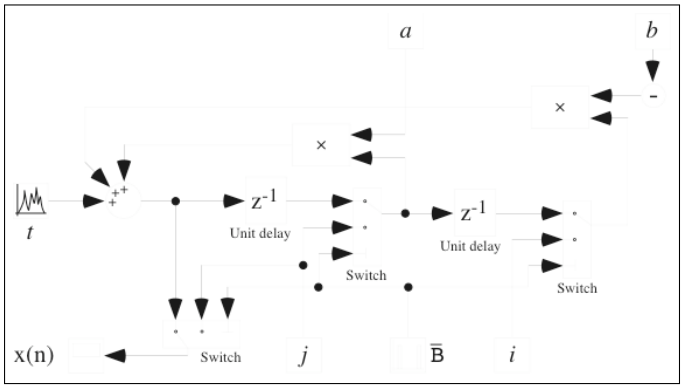
- Completion is the process of refining an abstraction of a semantics until a specification can be proved [CC79, GRS00]
- In theory, always possible by an infinite fixpoint computation *in the concrete!* [Cou00, GRS00]
- In complicated cases, the most abstract complete refined abstraction is *identity* (in which case the refinement ultimately amounts to computing the collecting semantics)
- Examples of *refinement semi-algorithms*:
  - counter-example-guided abstraction refinement [CGJ$^+$00]
  - fixpoint abstraction refinement [CGR07]

The limits of fixpoint abstraction completion

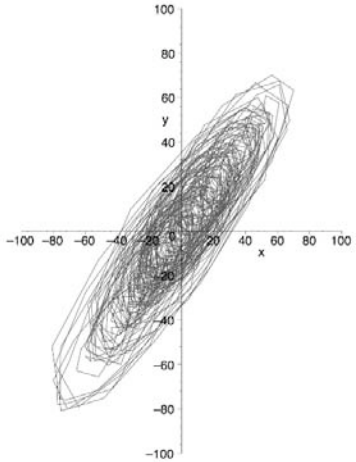
– Abstraction completion algorithms have misunderstood *severe limits*:
  - the refinement may be useless (corrected in [CGR07])
  - may not terminate (by ultimately computing in the infinite collecting semantics)
  - cannot to pass to the limit
  - cannot invent
    · efficient data representations of refined abstract properties (rely on state enumeration)
    · effective abstract transformer algorithms (rely on set of states transformers)

Ellipsoid Abstract Domain for Filters

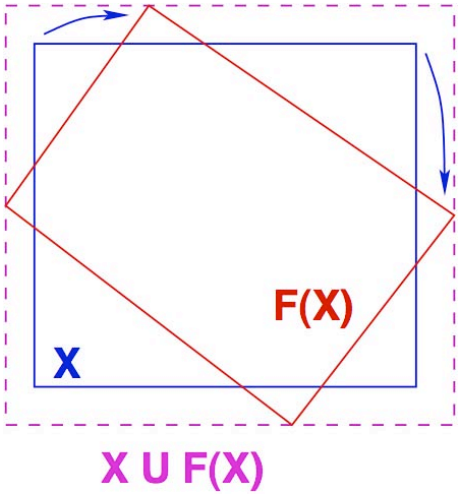
2^d Order Digital Filter:



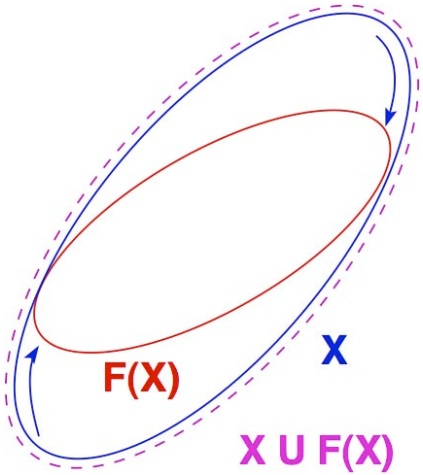
- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

15. Industrial applications of abstract interpretation

Industrial results obtained with ASTRÉE

Automatic proofs of absence of runtime errors in Electric Flight Control Software:

- Software 1 : 132.000 lignes de C, 40mn sur un PC 2.8 GHz, 300 mégaoctets (nov. 2003)
- Software 2 : 1.000.000 de lignes de C, 34h, 8 gigaoctets (nov. 2005)

No false alarm

World premières !

THE END

Thank you for your attention

16. Bibliography

Short bibliography

[BCC+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.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. 2nd 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 *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

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

[CC92] 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. 4th Int. Symp. on PLILP '92*, Leuven, BE, 26–28 Aug. 1992, LNCS 631, pages 269–295. Springer, 1992.

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

[CCF+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. 1st TASE '07*, pages 3–17, Shanghai, CN, 6–8 June 2007. IEEE Comp. Soc. Press.

[CCF+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, *11th ASIAN 06*, pages 272–300, Tokyo, JP, 6–8 Dec. 2006, 2008. LNCS 4435, Springer.

[CGJ+00] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E.A. Emerson and A.P. Sistla, editors, *Proc. 12th Int. Conf. CAV '00*, Chicago, IL, US, LNCS 1855, pages 154–169. Springer, 15–19 Jul. 2000.

[CGR07] P. Cousot, P. Ganty, and J.-F. Raskin. Fixpoint-guided abstraction refinements. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 333–348. Springer, 22–24 Aug. 2007.

[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.

[Cou00] P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 1–25. Springer, 26–29 Jul. 2000.

[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.

[DKW08] V. D'Silva, D. Kroening, and G. Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, 27(7):1165–1178, 2008.

[DS07] D. Delmas and J. Souyris. ASTRÉE: from research to industry. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 437–451. Springer, 22–24 Aug. 2007.

[Fer05a] J. Feret. The arithmetic-geometric progression abstract domain. In R. Cousot, editor, *Proc. 6th Int. Conf. VMCAI 2005*, pages 42–58, Paris, FR, 17–19 Jan. 2005. LNCS 3385, Springer.

[Fer05b] J. Feret. Numerical abstract domains for digital filters. In *1st Int. Work. on Numerical & Symbolic Abstract Domains, NSAD'05*, Maison Des Polytechniciens, Paris, FR, 21 Jan. 2005.

[Gra89] P. Granger. Static analysis of arithmetical congruences. *Int. J. Comput. Math.*, 30:165–190, 1989.

[GRS00] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[Min04a] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In D. Schmidt, editor, *Proc. 30th 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.

[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.

[Mon08] D. Monniaux. The pitfalls of verifying floating-point computations. *TOPLAS*, 30(3):Article No. 12, may 2008.

[MR05] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. In M. Sagiv, editor, *Proc. 14th ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 5–20. Springer, Apr. 2Ñ-10, 2005.

[RM07] X. Rival and L. Mauborgne. The trace partitioning abstract domain. *TOPLAS*, 29(5), Aug. 2007.

