Automatic Software Verification by Abstract Interpretation

Patrick Cousot

The First International Conference on Foundations of Informatics, Computing and Software

Shanghai, China, June 3–6, 2008

1. Classical examples of bugs

Classical examples of bugs in integer computations

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));
}
```

← $\mathtt{fact}(n) = 2 \times 3 \times \cdots \times n$

← read n (typed on keyboard)

← write $n\,! = \mathtt{fact}(n)$

Compilation of 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
%
```

Executions of the factorial program (fact.c)

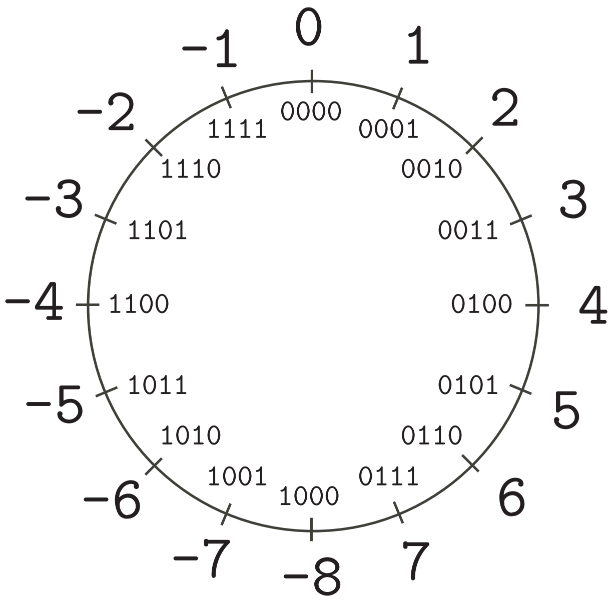
```c
#include <stdio.h>
int fact (int n ) {
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {
    r = r*i;
  }
  return r;
}
int main() { int n;
  scanf("%d",&n);
  printf("%d!=%d\n",n,fact(n));
}
```

```
% 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
```

Bug hunt

– 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*) :

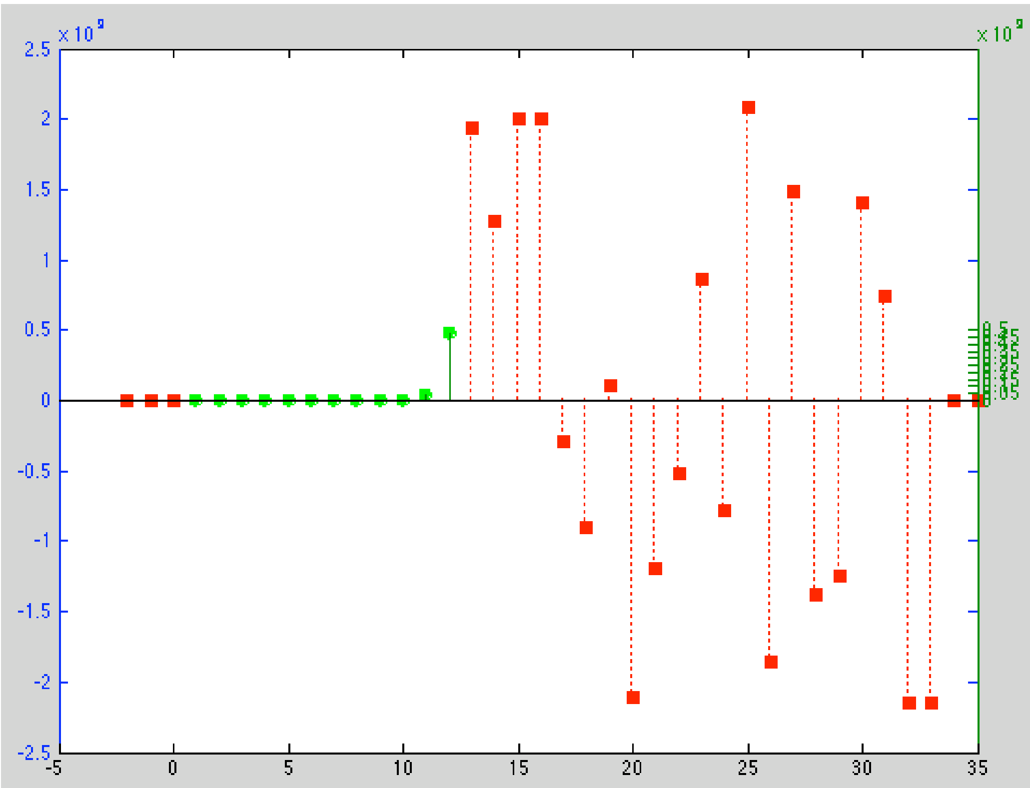


– Only integers between -8 and 7 can be represented on 4 bits

– We get $7 + 2 = -7$
$7 + 9 = 0$

The bug is a failure of the programmer

In the computer, the function `fact(n)` coincide with $n! = 2 \times 3 \times \ldots. \times n$ on the integers only for $1 \leqslant n \leqslant 12$:



And in OCaml the result is different!

```
let rec fact n = if (n = 1) then 1 else n * fact(n-1);;
```

fact(n)	C	OCaml
fact(1)	1	1
...	...	...
fact(12)	479001600	479001600
fact(13)	1932053504	−215430144
fact(14)	1278945280	−868538368
fact(15)	2004310016	−143173632
fact(16)	2004189184	−143294464
fact(17)	−288522240	−288522240
fact(18)	−898433024	−898433024
fact(19)	109641728	109641728
fact(20)	−2102132736	45350912
fact(21)	−1195114496	952369152

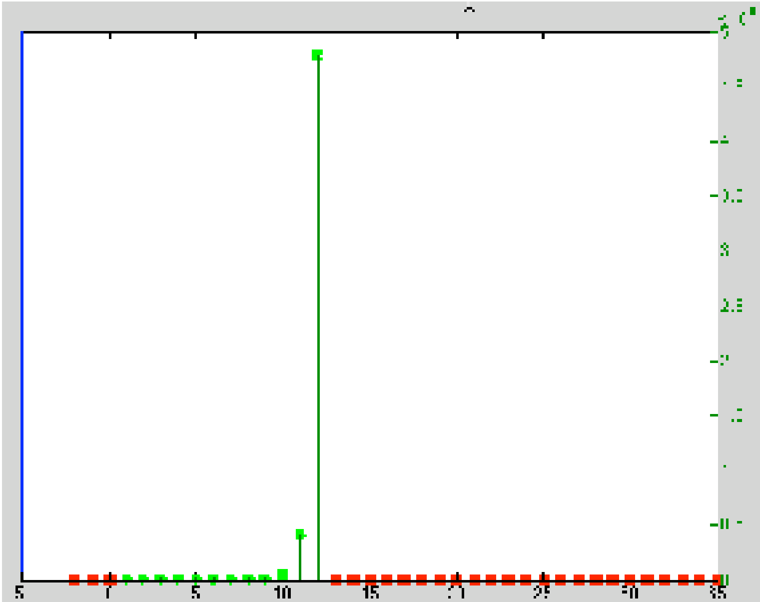
fact(22)	−522715136	−522715136
fact(23)	862453760	862453760
fact(24)	−775946240	−775946240
fact(25)	2076180480	−71303168
fact(26)	−1853882368	293601280
fact(27)	1484783616	−662700032
fact(28)	−1375731712	771751936
fact(29)	−1241513984	905969664
fact(30)	1409286144	−738197504
fact(31)	738197504	738197504
fact(32)	−2147483648	0
fact(33)	−2147483648	0
fact(34)	0	0

Why? What is the result of `fact(-1)` ?

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!



Examples of classical bugs in floating point computations

Mathematical models and their implementation on computers

- Mathematical models of physical systems use real numbers

- Computer modeling languages (like SCADE) use real numbers

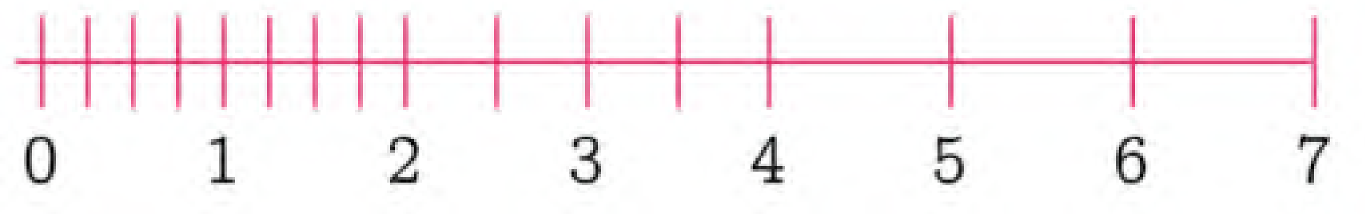
- Real numbers are hard to represent in a computer (π has an infinite number of decimals)

- Computer programming languages (like C or OCAML) use floating point numbers

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)

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

```c
#include <stdio.h>
int main() {
  double x, a; float y, z;
  x = 1125899973951488.0;
  a = 1.0;
  y = (x+a);
  z = (x-a);
  printf("%f\n", y-z);
}
```

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

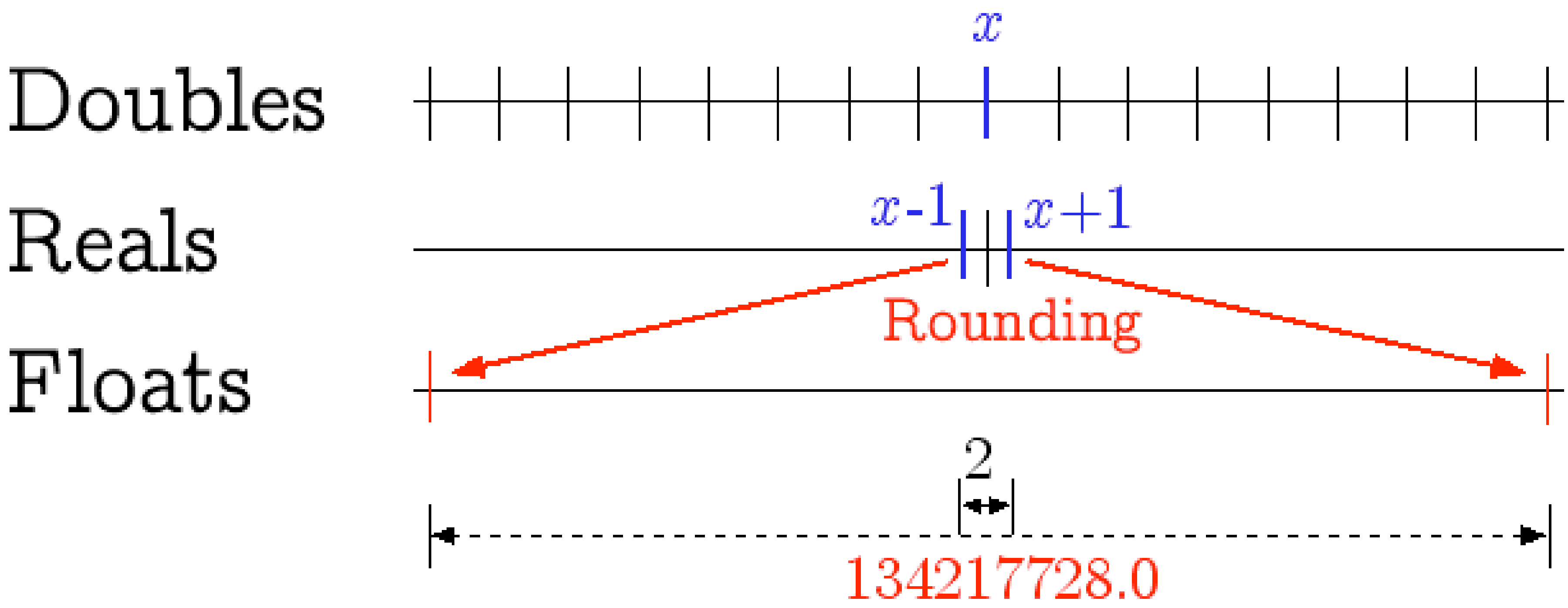
Example of rounding error (2)

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

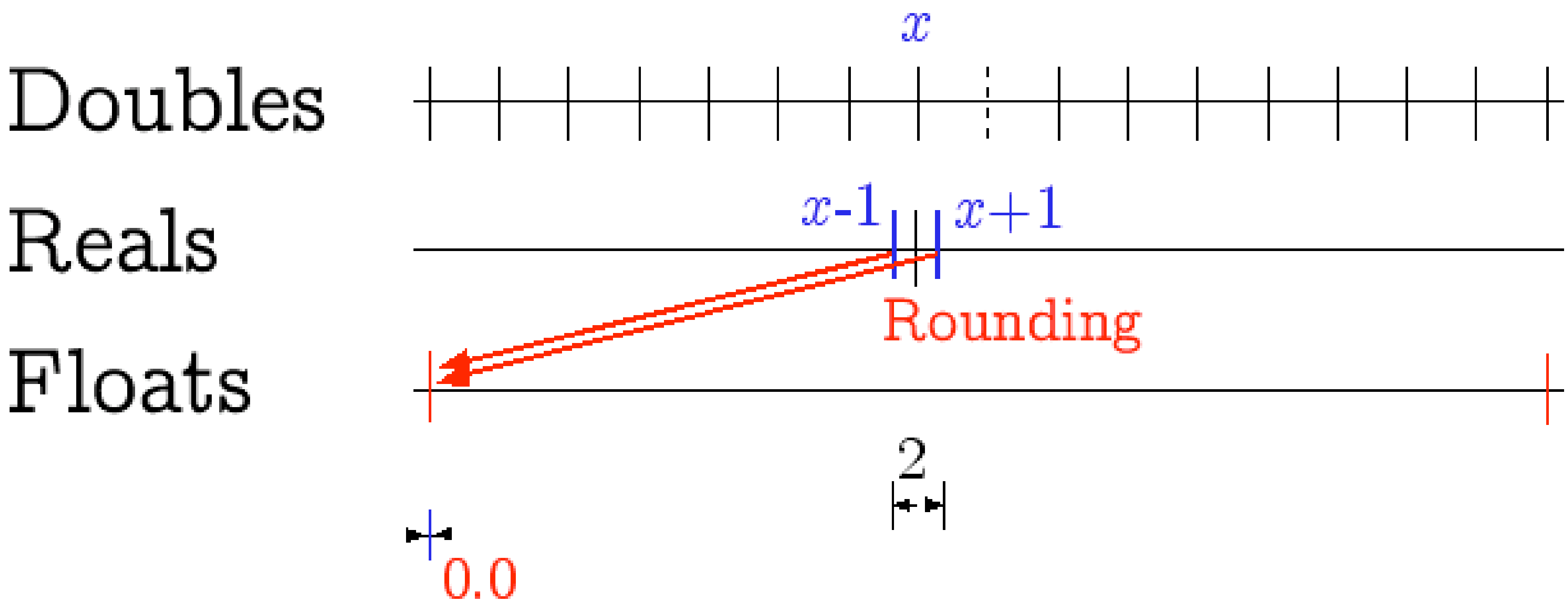
```c
#include <stdio.h>
int main() {
  double x, a; float y, z;
  x = 1125899973951487.0;
  a = 1.0;
  y = (x+a);
  z = (x-a);
  printf("%f\n", y-z);
}
```

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

Bug hunt (1)



Bug hunt (2)

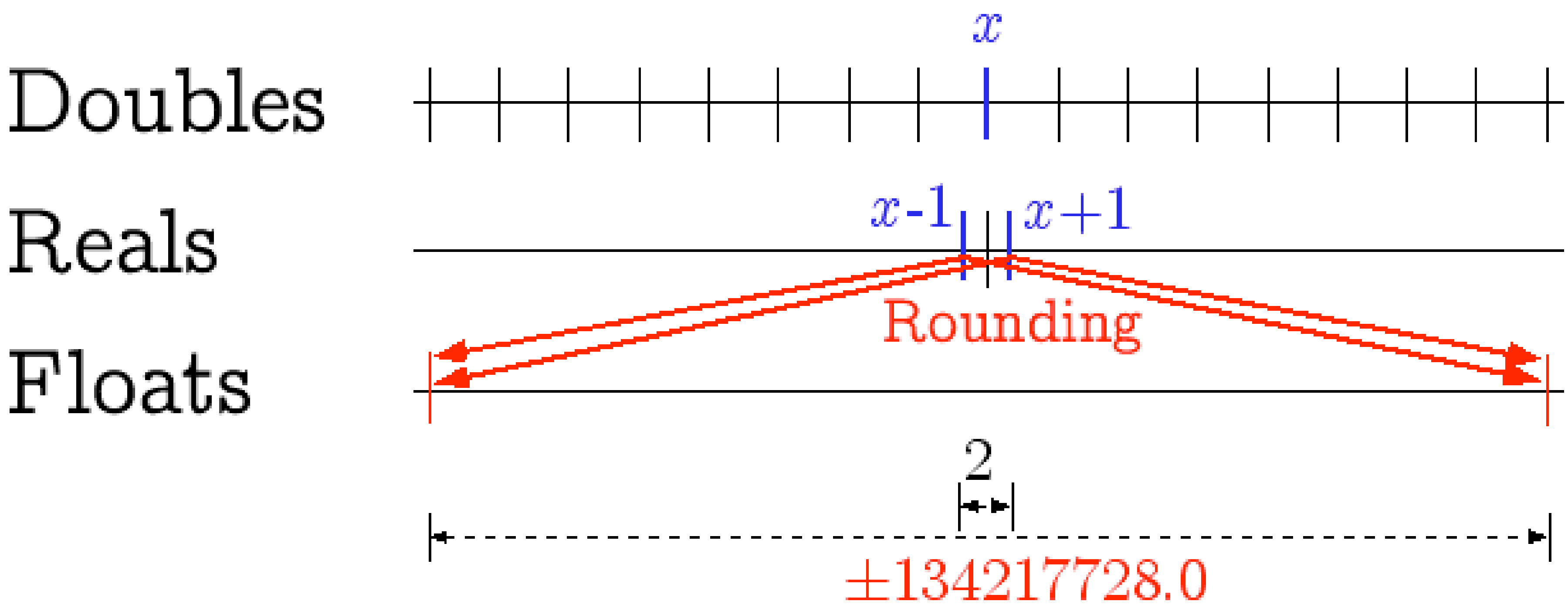


Proof of absence of runtime error by static analysis

```
% 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.

Bugs in the everyday numerical world

Bugs are frequent in everyday life

– Bugs proliferate in banks, cars, telephons, washing machines, ...

– Example (bug in an ATM machine located at 19 Boulevard Sébastopol in Paris, on 21 November 2006 at 8:30):

– Hypothesis (Gordon Moore's law revisited): the number of software bugs in the world double every 18 months??? :-(

2. Program verification

Principle of program verification

- Define a semantics of the language (that is the effect of executing programs of the language)

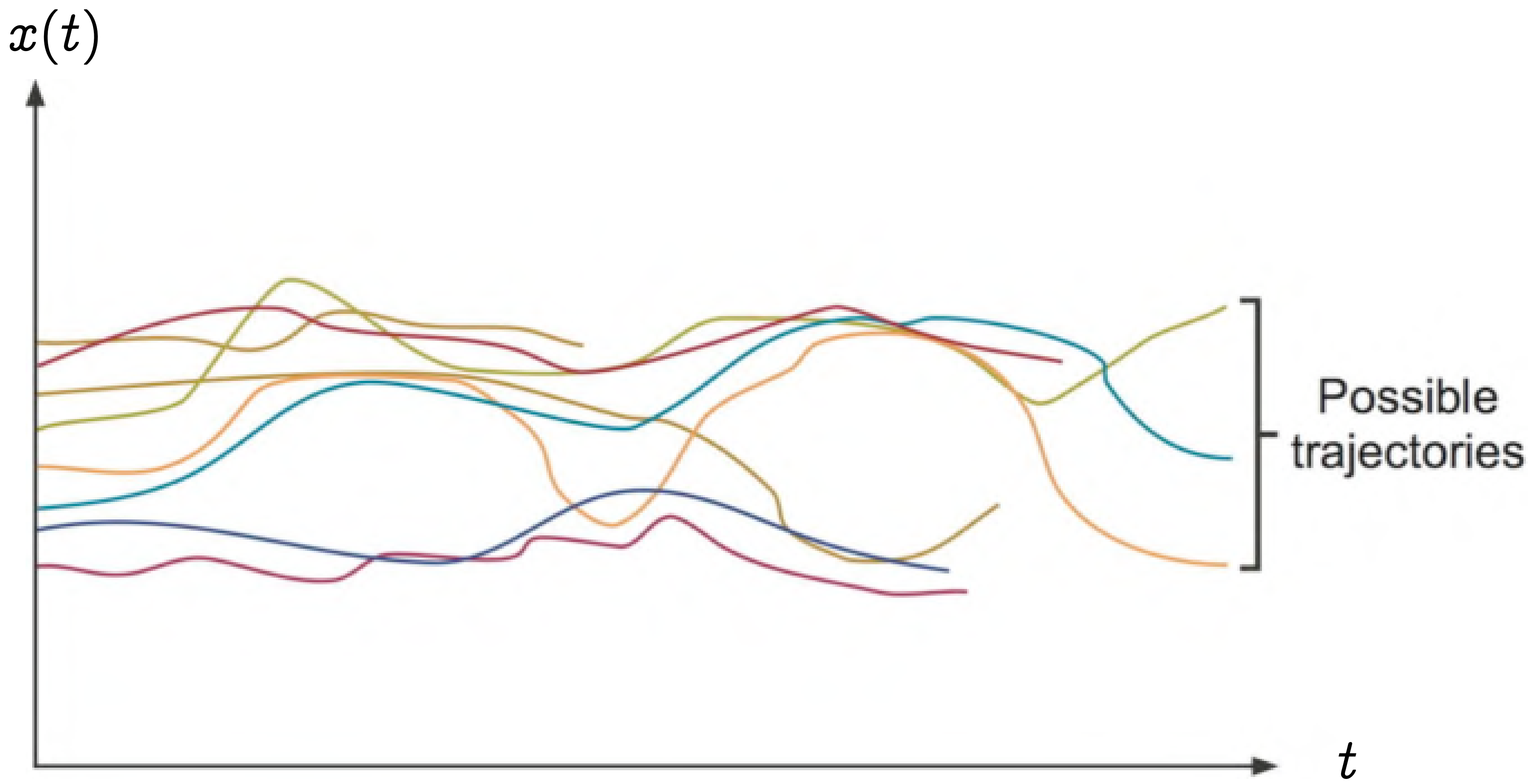
- Define a specification (example: absence of runtime errors such as division by zero, un arithmetic overflow, etc)

- Make a formal proof that the semantics satisfies the specification

- Use a computer to automate the proof

Semantics of programs

Operational semantics of program P



Semantics$[\![P]\!]$

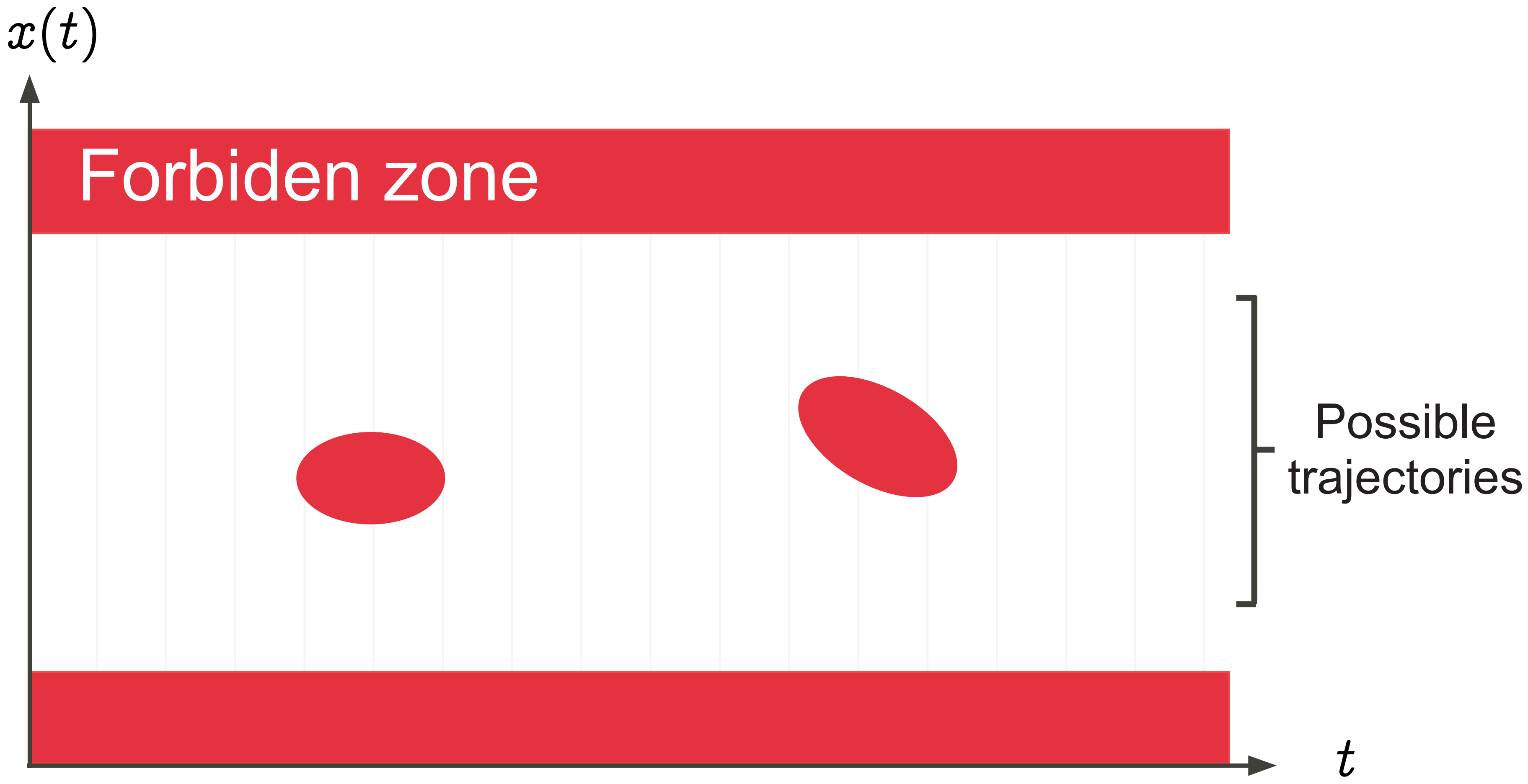
Example: execution trace of `fact(4)`

```
int fact (int n ) {
  int r = 1, i;
  for (i=2; i<=n; i++) {
    r = r*i;
  }
  return r;
}
```

- $\mathtt{n} \leftarrow 4$; $\mathtt{r} \leftarrow 1$;
- $\mathtt{i} \leftarrow 2$; $\mathtt{r} \leftarrow 1 \times 2 = 1$;
- $\mathtt{i} \leftarrow 3$; $\mathtt{r} \leftarrow 2 \times 3 = 6$;
- $\mathtt{i} \leftarrow 4$; $\mathtt{r} \leftarrow 6 \times 4 = 24$;
- $\mathtt{i} \leftarrow 5$;
- `return` 24;

Program specification

Specification of program P



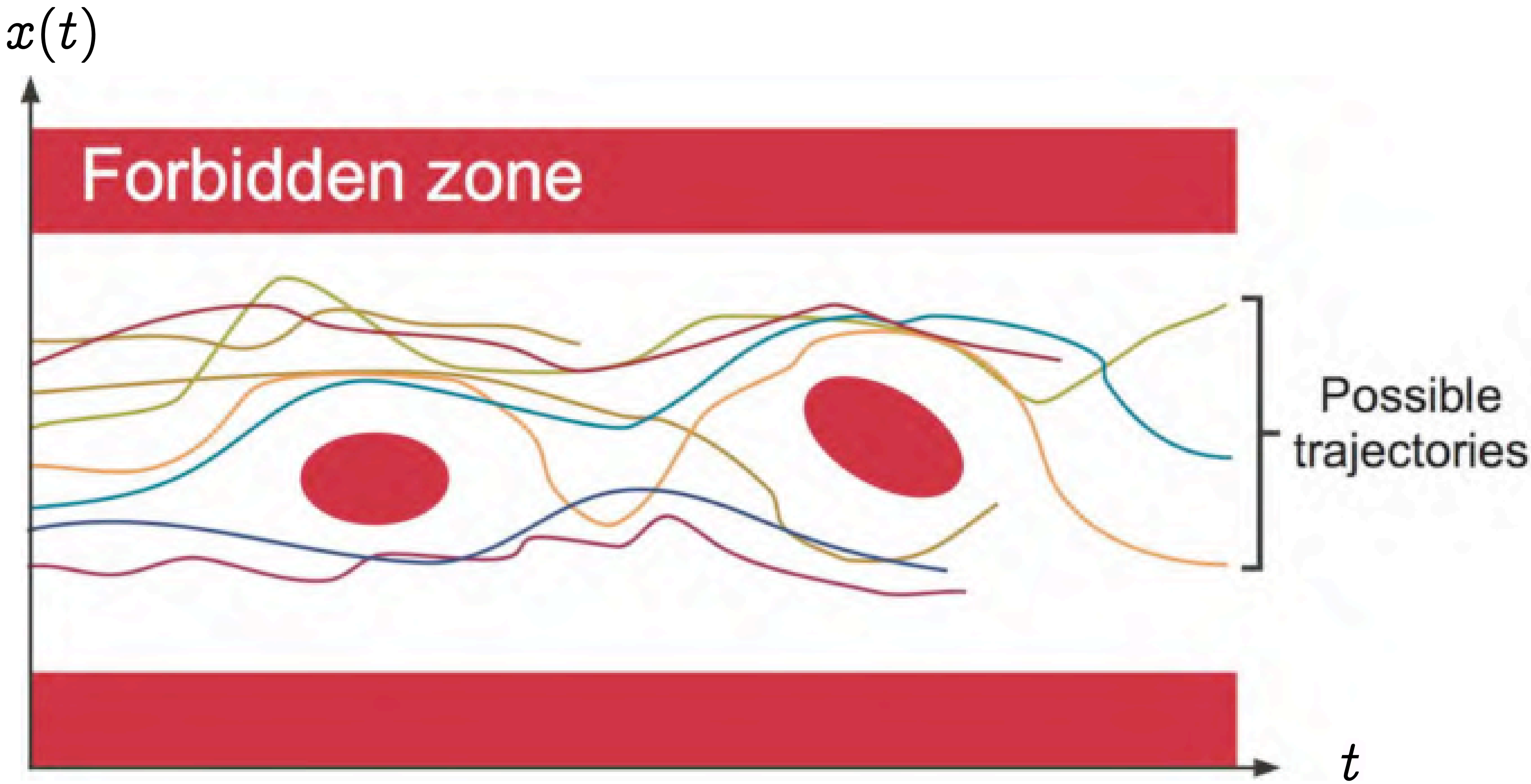
Specification$[\![P]\!]$

Example of specification

```
int fact (int n ) {
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {      ← no overflow of i++
    r = r*i;                  ← no overflow of r*i
  }
  return r;
}
```

Formal proofs

Formal proof of program P



$$\text{Semantics}[\![P]\!] \subseteq \text{Specification}[\![P]\!]$$

Undecidability and complexity

– The mathematical proof problem is undecidable [(3)]

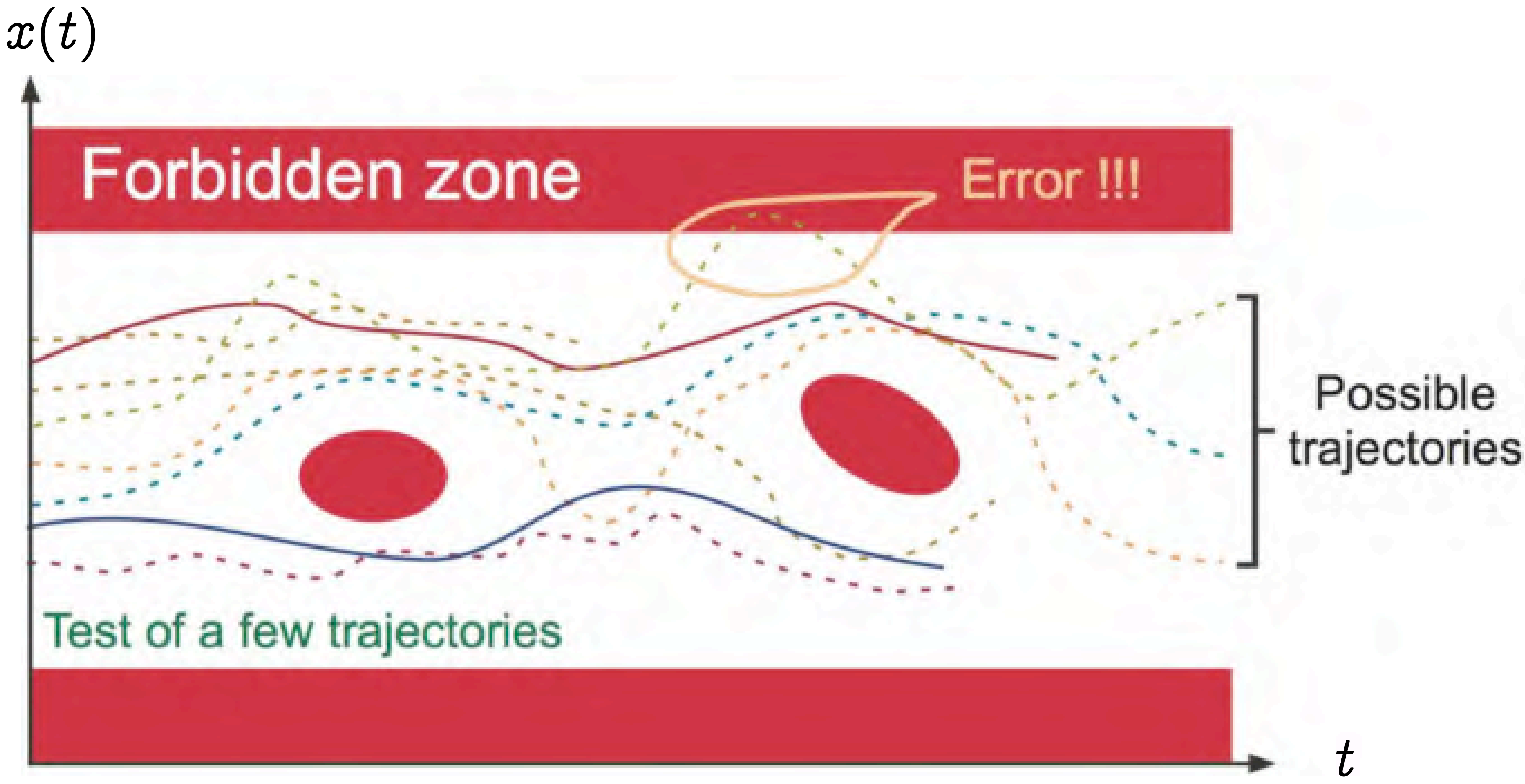
– Even assuming finite states, the complexity is much too high for combinatorial exploration to succeed

– 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.

Testing is incomplete

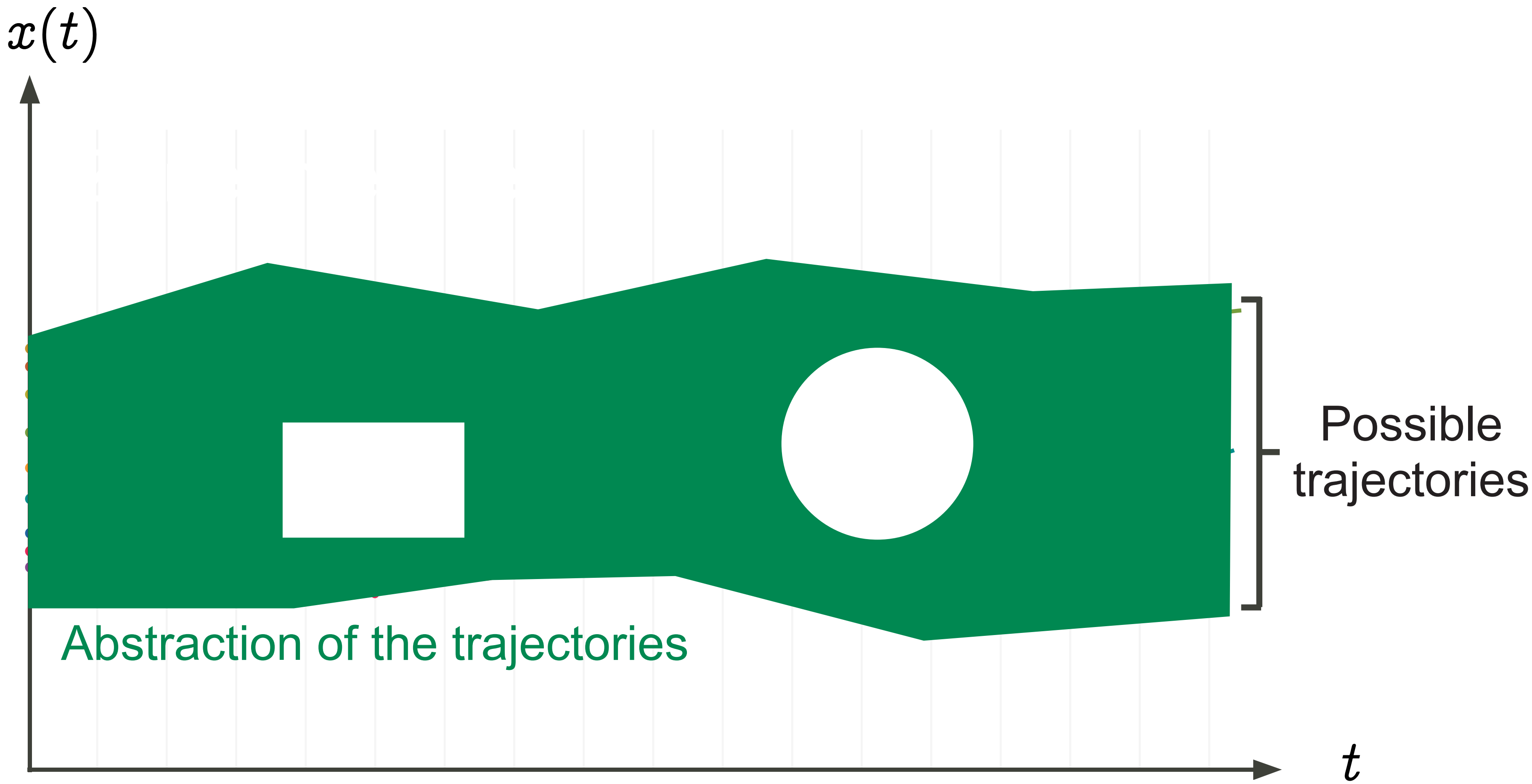


3. Abstract interpretation [1]

Reference

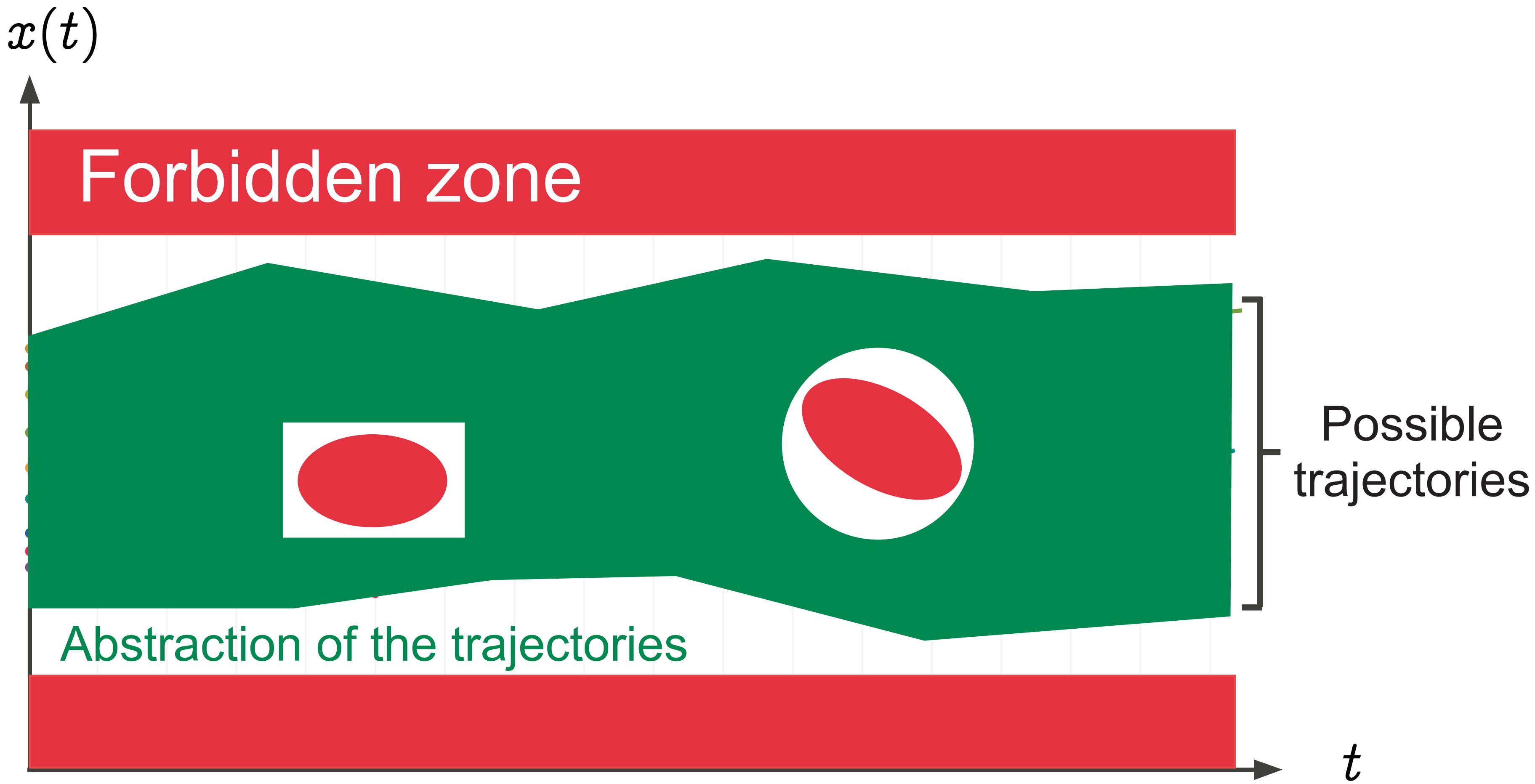
[1] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse d'État ès sciences mathématiques. Université scientifique et médicale de Grenoble. 1978.

Abstraction of program P



Abstraction(Semantics$[\![P]\!]$)

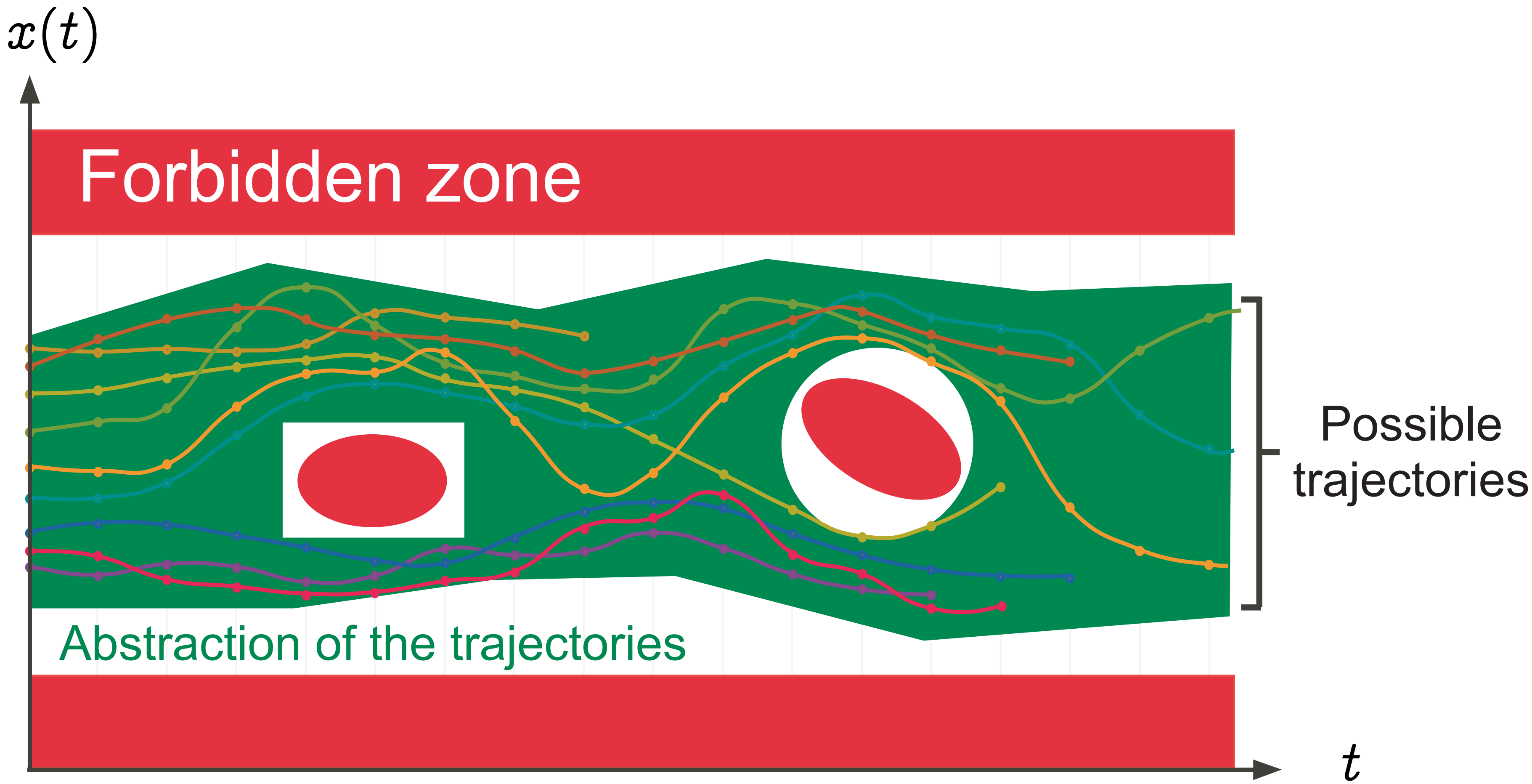
Proof by abstraction



$\text{Abstraction}(\text{Semantics}[\![P]\!]) \subseteq \text{Specificaton}[\![P]\!]$

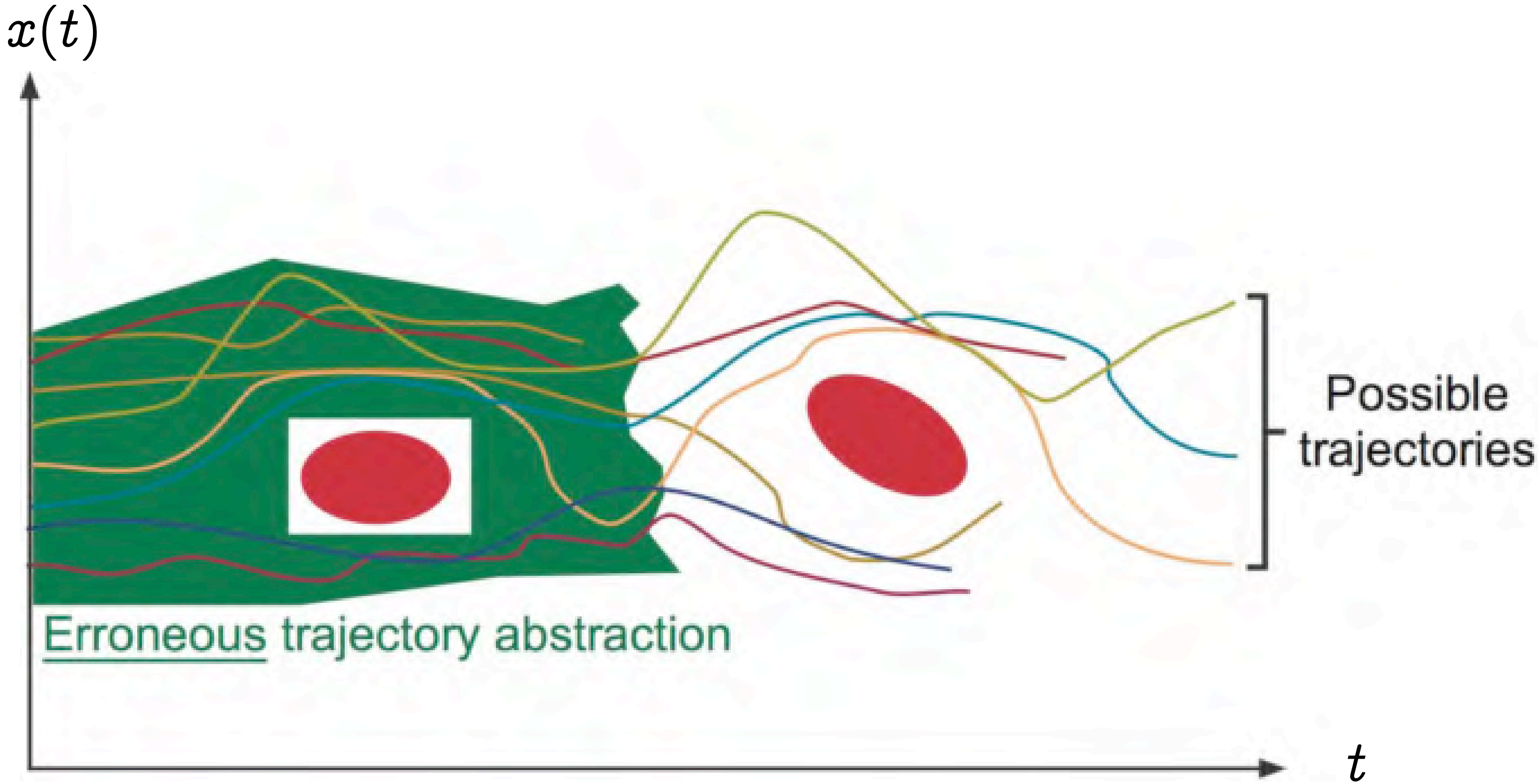
Soundness of abstract interpretation

Abstract interpretation is sound



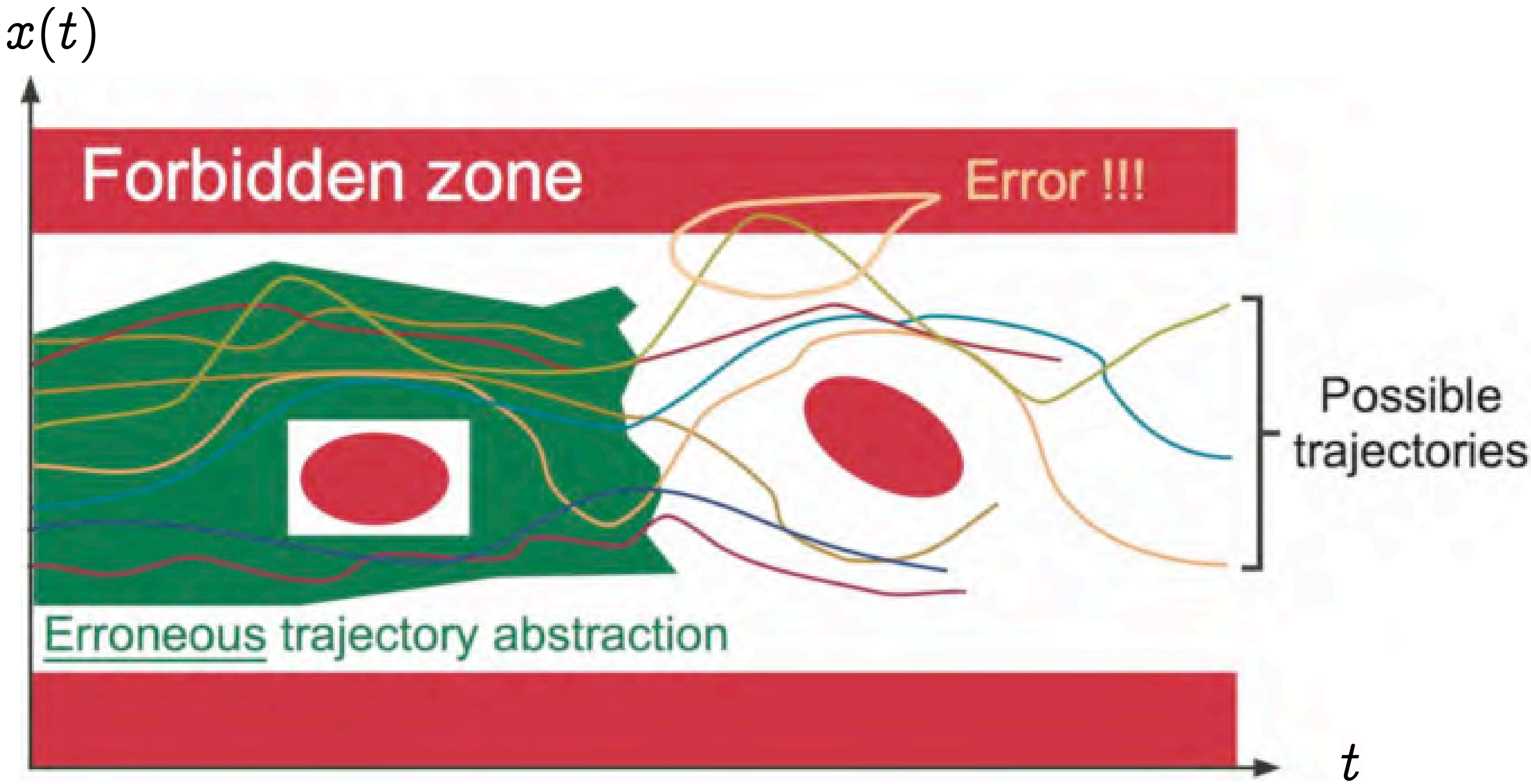
$\text{Semantics}[\![P]\!] \subseteq \text{Abstraction}(\text{Semantics}[\![P]\!])$

Example of unsound abstraction [4]



[4] Unsoundness is always excluded by abstract interpretation theory.

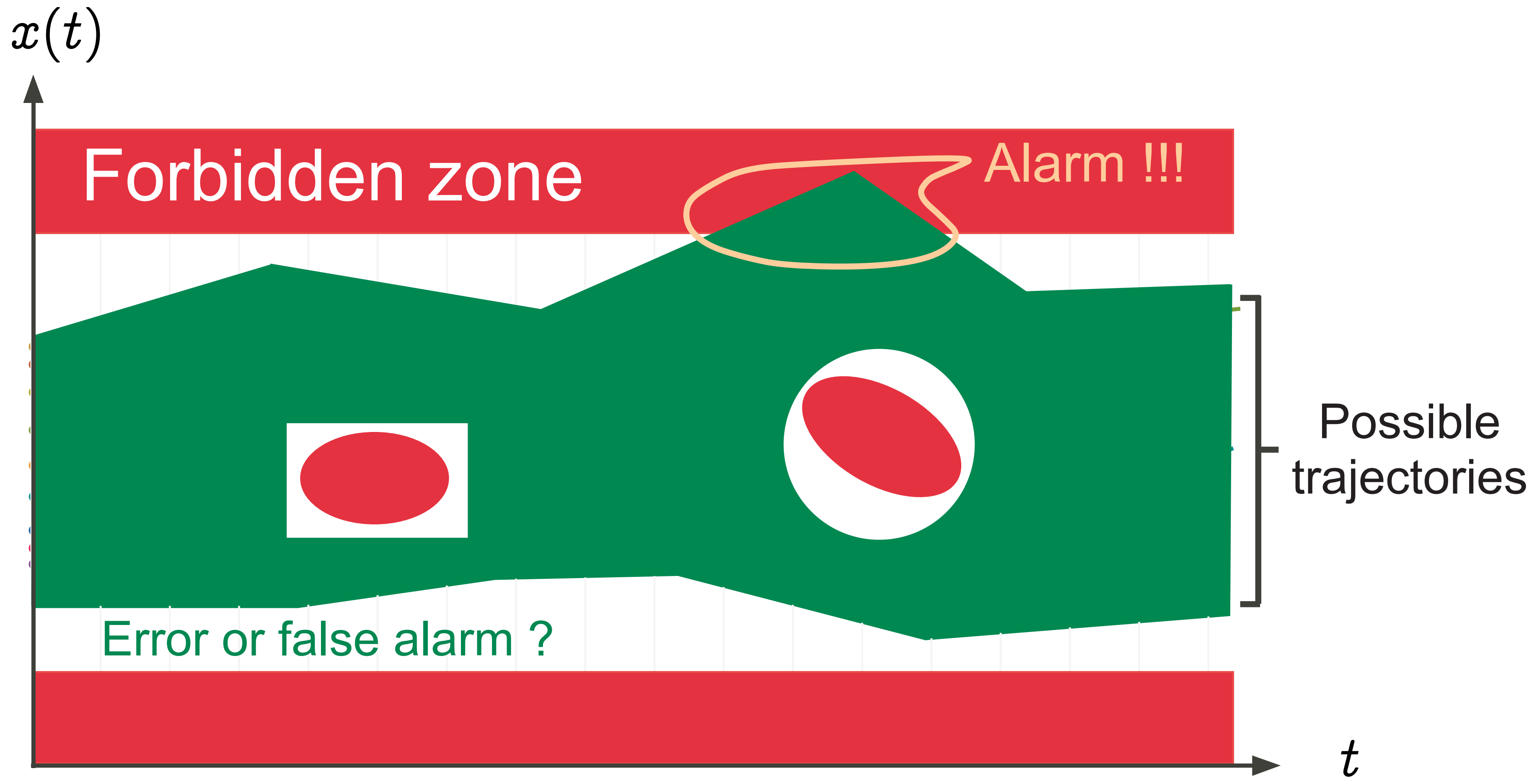
Unsound abstractions are inconclusive (false negatives) [4]



(4) Unsoundness is always excluded by abstract interpretation theory.

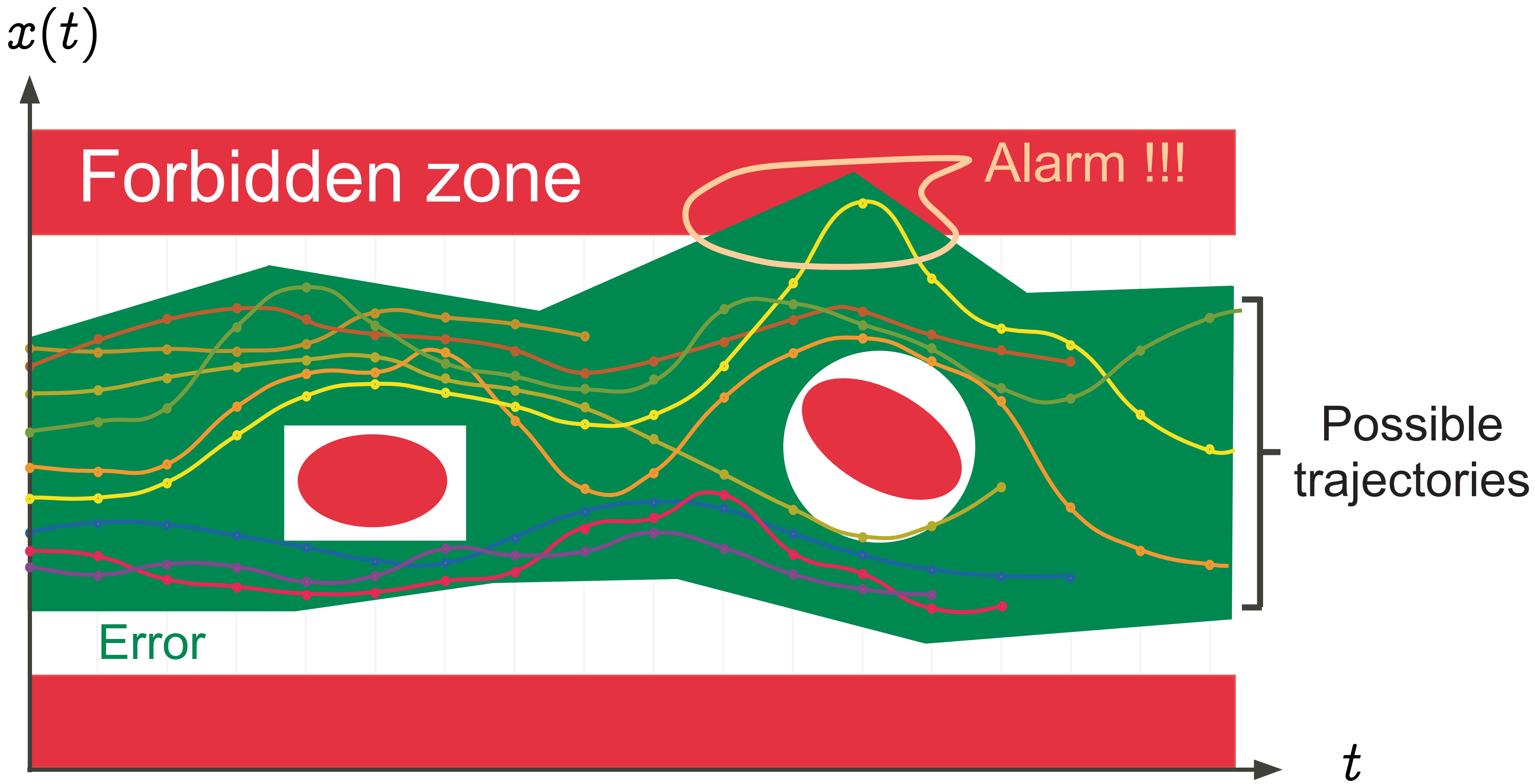
Incompleteness of abstract interpretation

Alarm

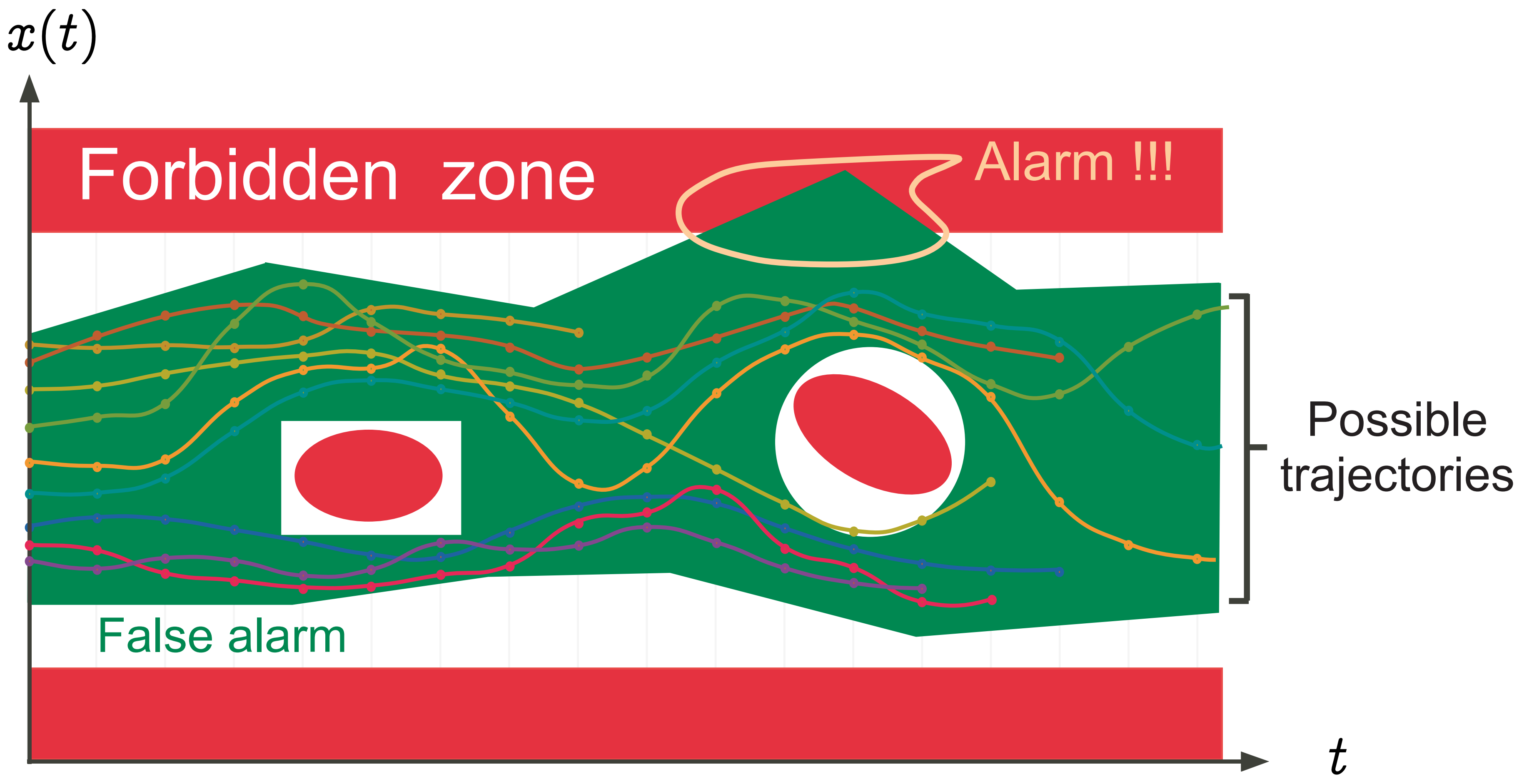


$x(t)$
Forbidden zone
Alarm !!!
Possible trajectories
Error or false alarm ?
t

An alarm can originate from an error



An alarm can originate from an over-approximation

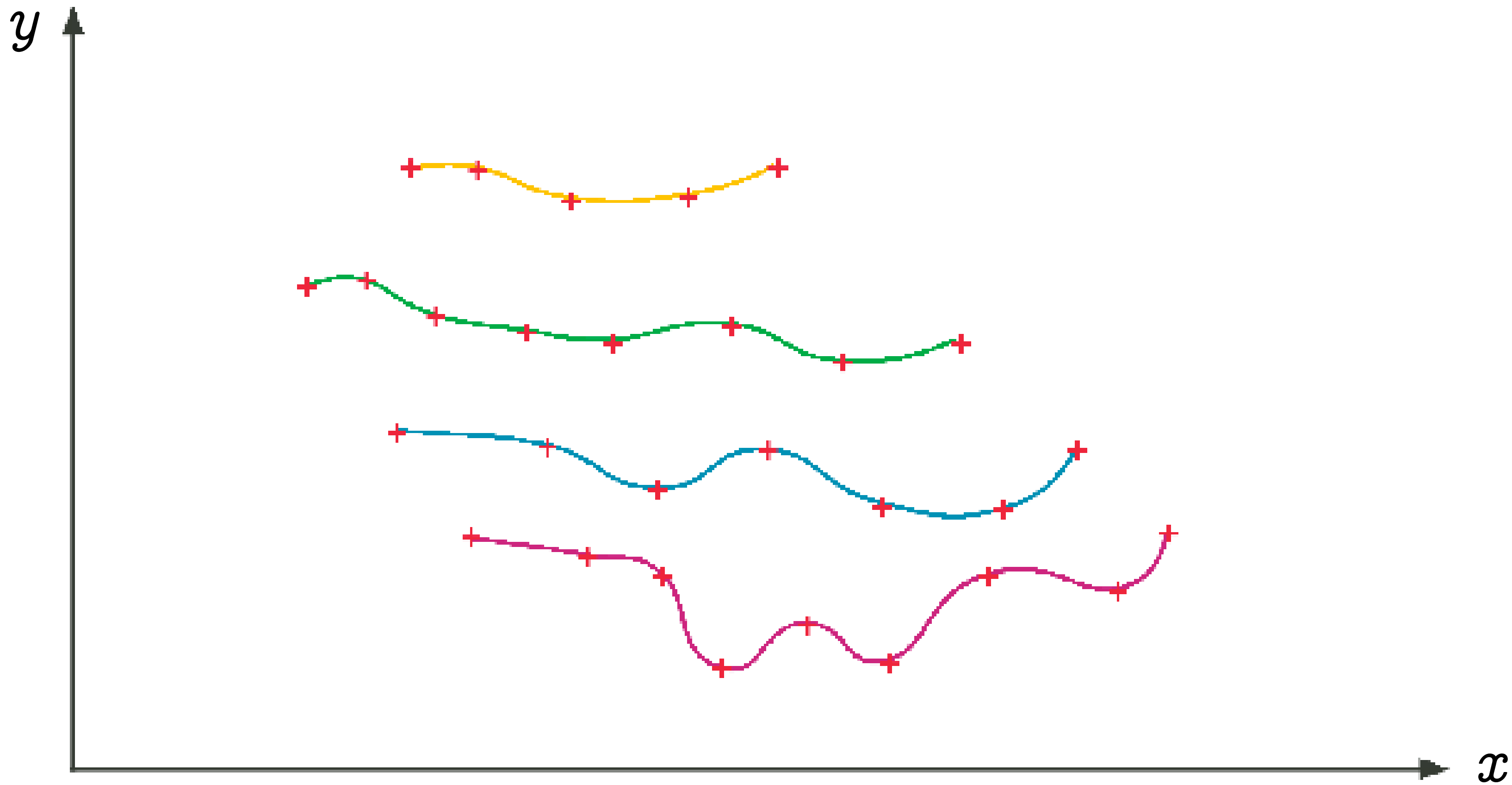


Examples of applications of abstract interpretation

- Typing [Cou97]
- Abstract model-checking [CC00]
- Program transformation (for example for program optimization during compilation, partial evaluation) [CC02]
- The definition of semantics at various levels of abstraction [Cou02]
- static analysis (or semantics-checking) to prove the absence of bugs [BCC+03]
- ...

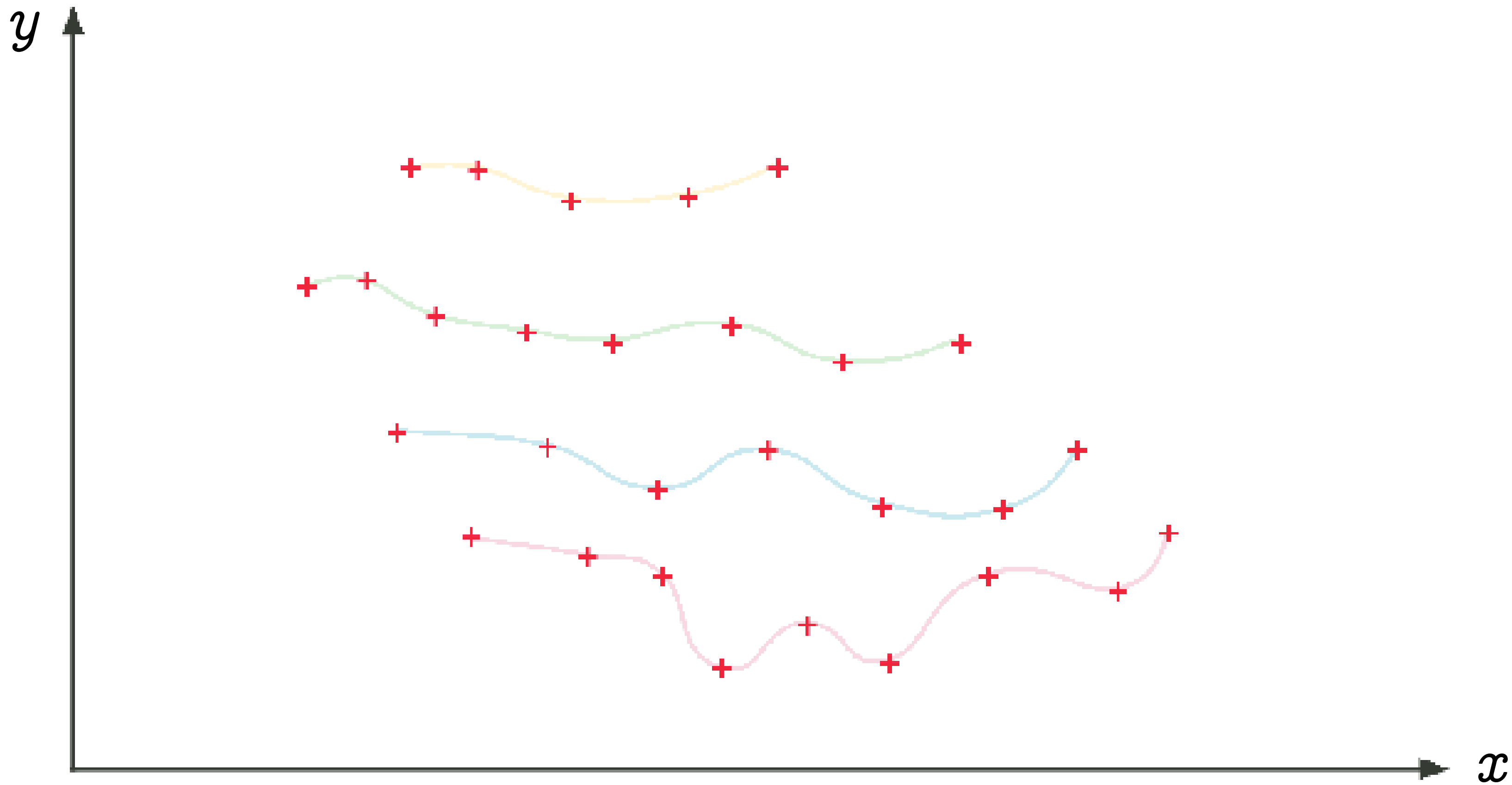
4. Application of abstract interpretation to static analysis

Semantics



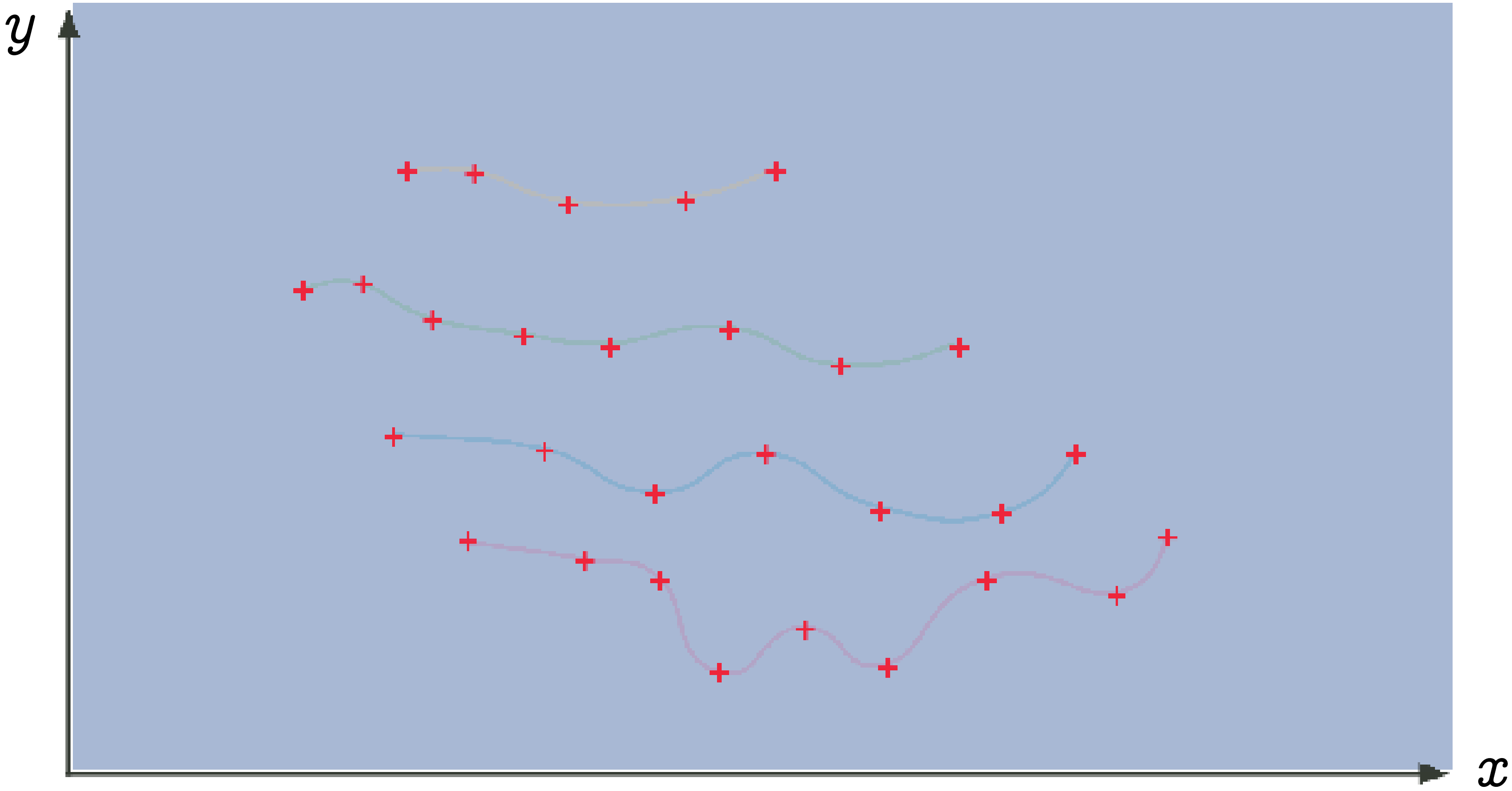
(Infinite) set of traces (finite ou infinite)

Abstraction to a set of states (invariant)



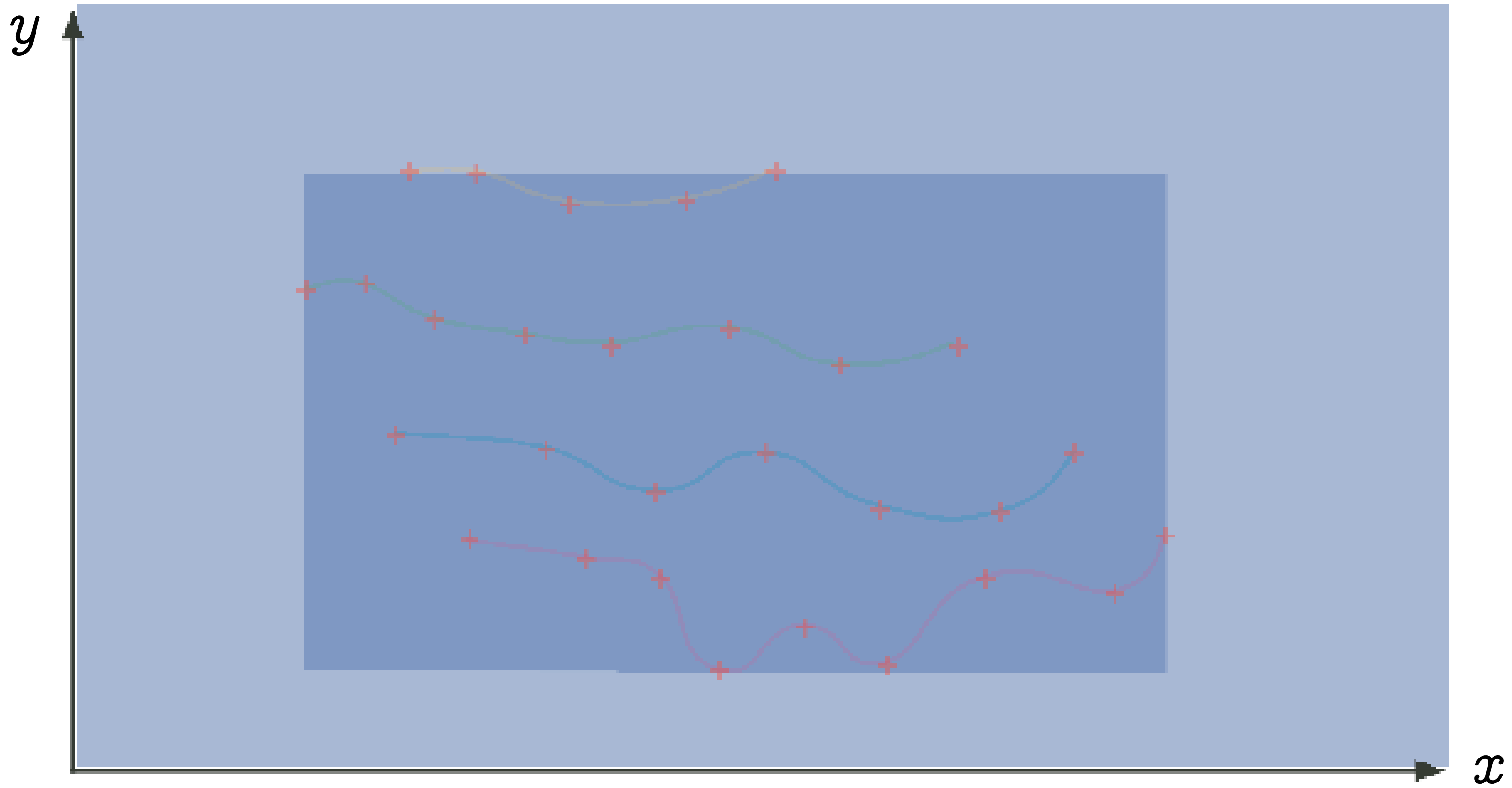
Set of points $\{(x_i, y_i) : i \in \Delta\}$, Floyd/Hoare/Naur invariance proof method [Cou02]

Abstraction by signs



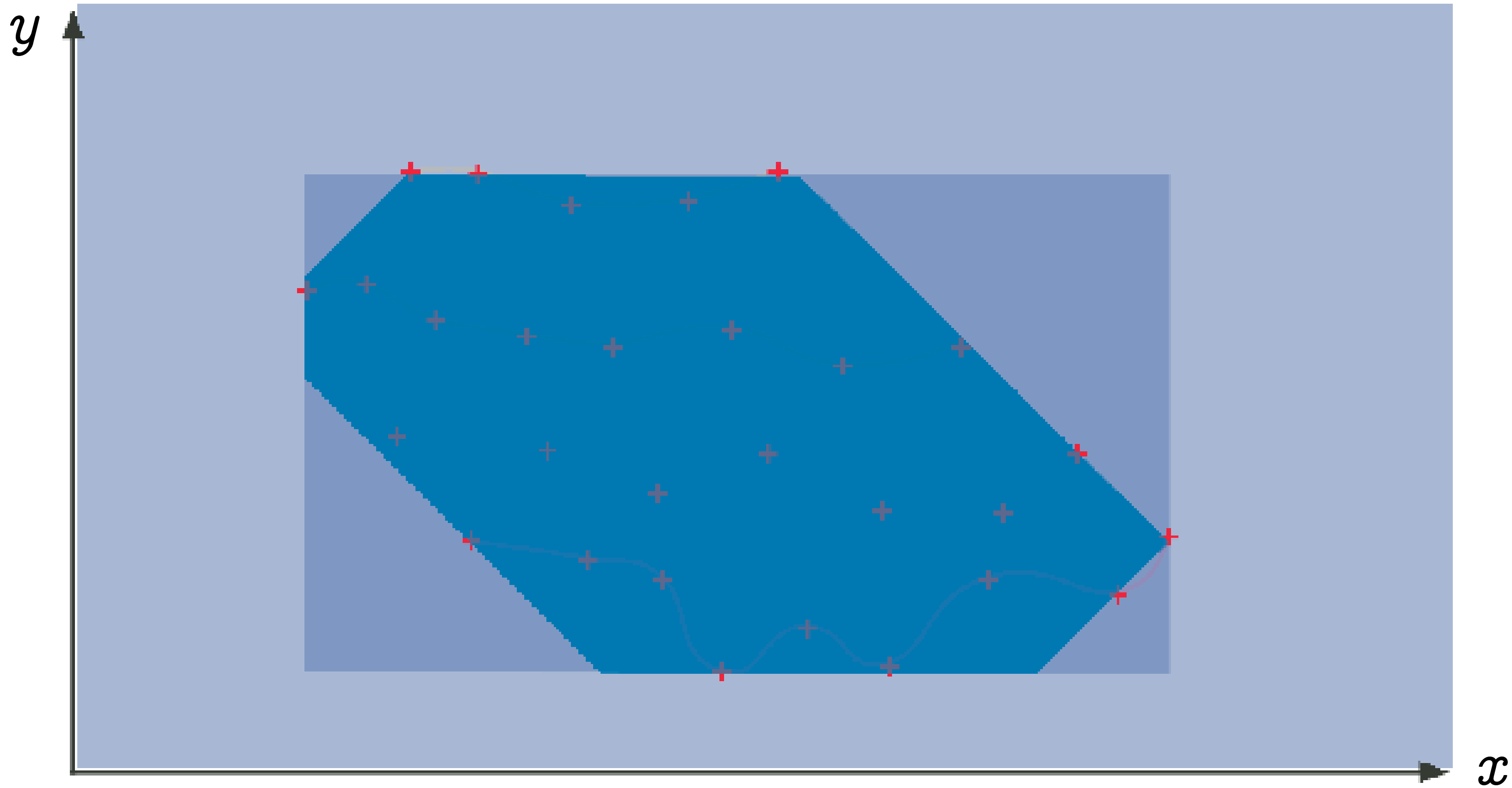
Signs $x \geq 0$, $y \geq 0$ [CC79]

Abstraction by intervals



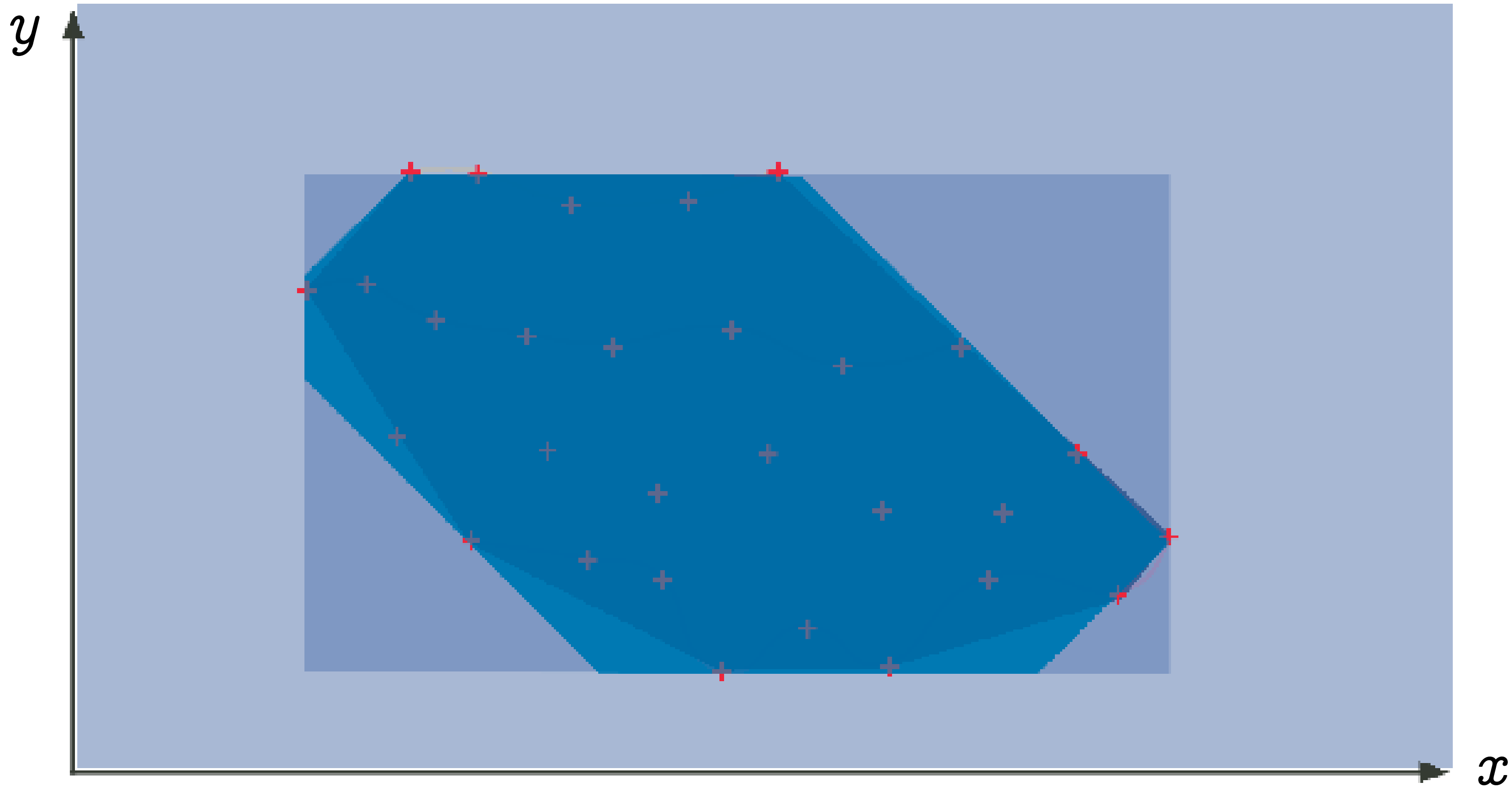
Intervals $a \leq x \leq b$, $c \leq y \leq d$ [CC77]

Abstraction by octagons



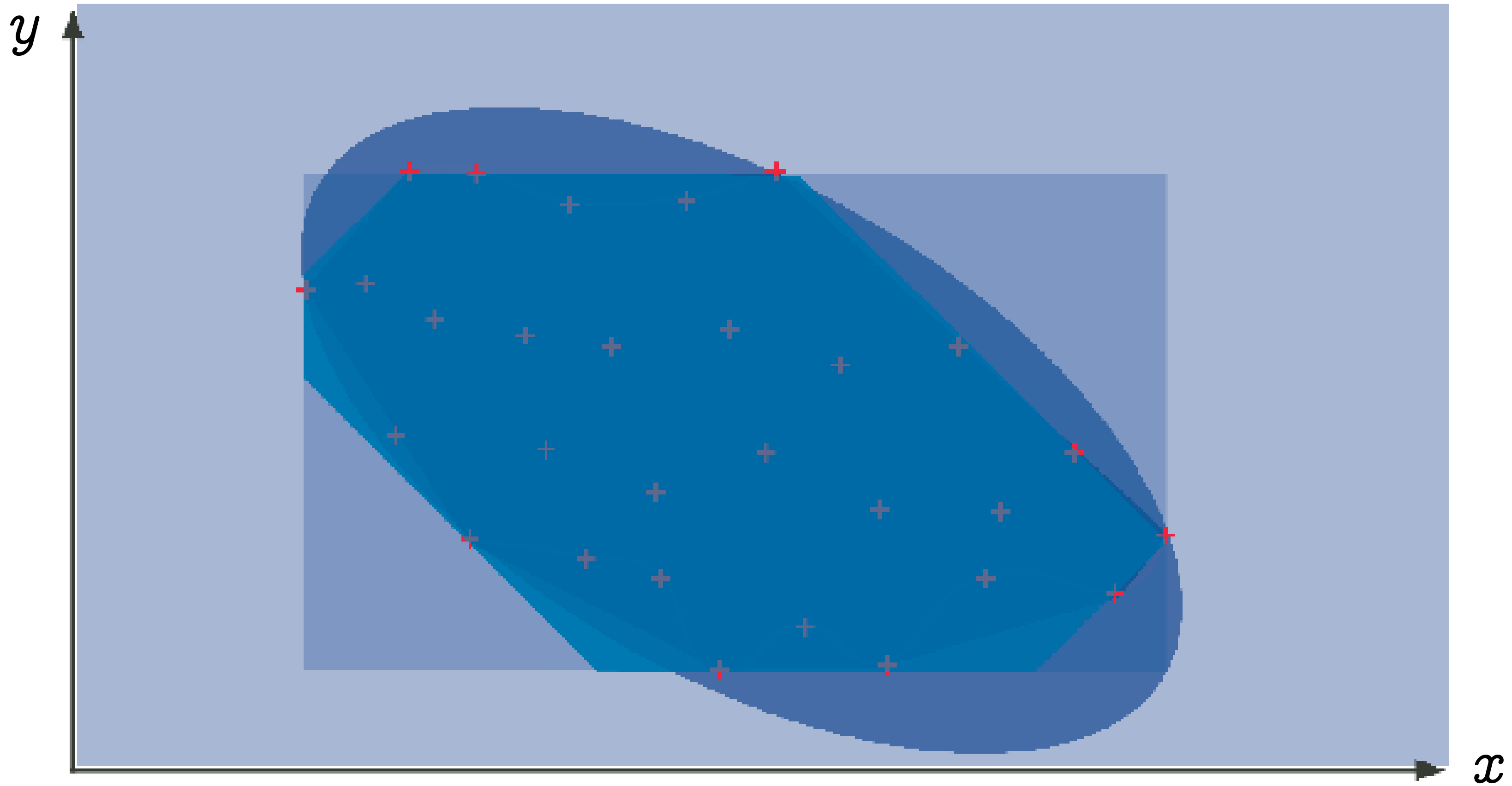
Octagons $x - y \leq a$, $x + y \leq b$ [Min06]

Abstraction by polyedra



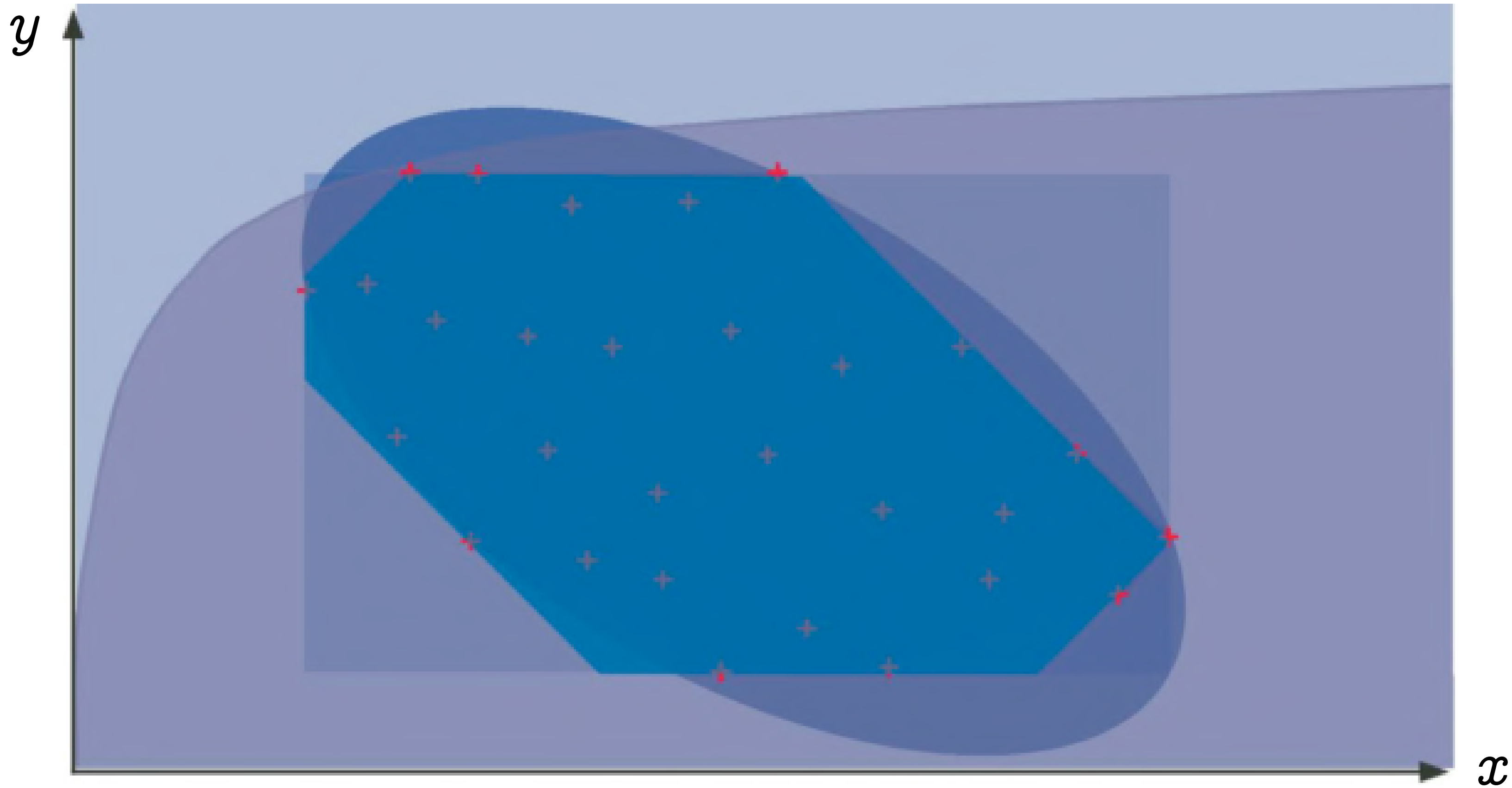
Polyedra $a.x + b.y \leq c$ [CH78]

Abstraction by ellipsoids



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

Abstraction by exponentials



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

5. Invariant computation by fixpoint approximation [CC77]

Fixpoint equation

$\{y \geqslant 0\}$ ← hypothesis

```
x = y
```

$\{I(x, y)\}$ ← 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:

$$I(x, y) \;=\; x \geqslant 0 \wedge (x = y \vee I(x + 1, y)) \qquad (\textit{i.e. } I = F(I)\,^{(5)})$$

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

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

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

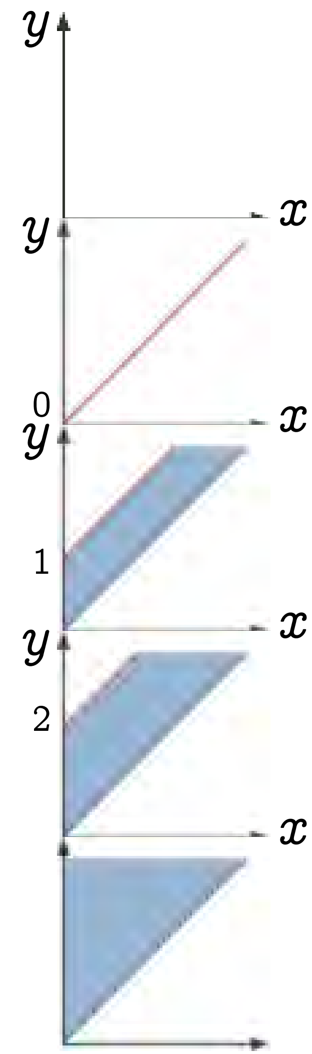
$$\begin{aligned} I^1(x,y) &= x \geqslant 0 \wedge (x = y \vee I^0(x+1,y)) \\ &= 0 \leqslant x = y \end{aligned}$$

$$\begin{aligned} I^2(x,y) &= x \geqslant 0 \wedge (x = y \vee I^1(x+1,y)) \\ &= 0 \leqslant x \leqslant y \leqslant x+1 \end{aligned}$$

$$\begin{aligned} I^3(x,y) &= x \geqslant 0 \wedge (x = y \vee I^2(x+1,y)) \\ &= 0 \leqslant x \leqslant y \leqslant x+2 \end{aligned}$$

$$\begin{aligned} I^4(x,y) &= I^2(x,y) \nabla I^3(x,y) \leftarrow \text{widening} \\ &= 0 \leqslant x \leqslant y \end{aligned}$$

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



The invariants are computer representable with octagons!

6. Scaling up

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* (when considering the analysis of a programming language) [CC92]
- Abstractions are ultimately incomplete (even intrinsically for some semantics and specifications [CC00])

Problems with software verification by abstraction completion

– Completion [CC79, GRS00] is the process of refining an abstraction of a semantics until a specification can proved (e.g. [CGJ+00, CGR07])

– Software verification by abstraction completion/refinement has serious problems:

  - completion involves computations in the infinite domain of the concrete semantics (with undecidable implication) so refinement algorithms assuming a finite concrete domain [CGJ+00, CGR07] are inapplicable

  - Completion does not provide an effective computer representation of refined abstract properties

  - Completion is an infinite iterative process (in general not convergent)

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 <u>not</u> uniform.

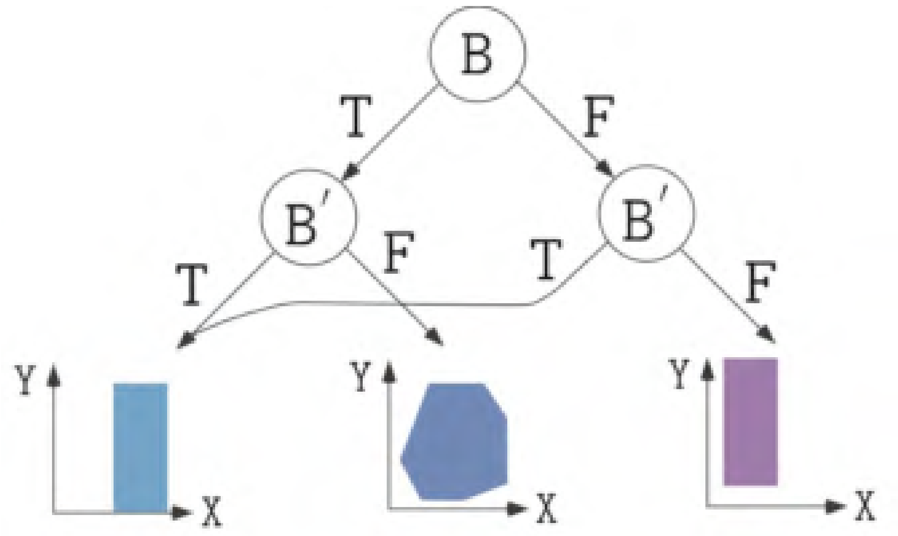
Example of abstract domain choice in ASTRÉE

```
/* 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 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

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 : $X = 0 \mod 4 \Longrightarrow X \in \{132\}$.

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" ]
```

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 be automated in ASTRÉE (if the considered family of programs has "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;
}
...
%
```

Adding new abstract domains

- The weakest invariant to prove the specification may not be expressible with the current refined abstractions $\Rightarrow$ false alarms cannot be solved

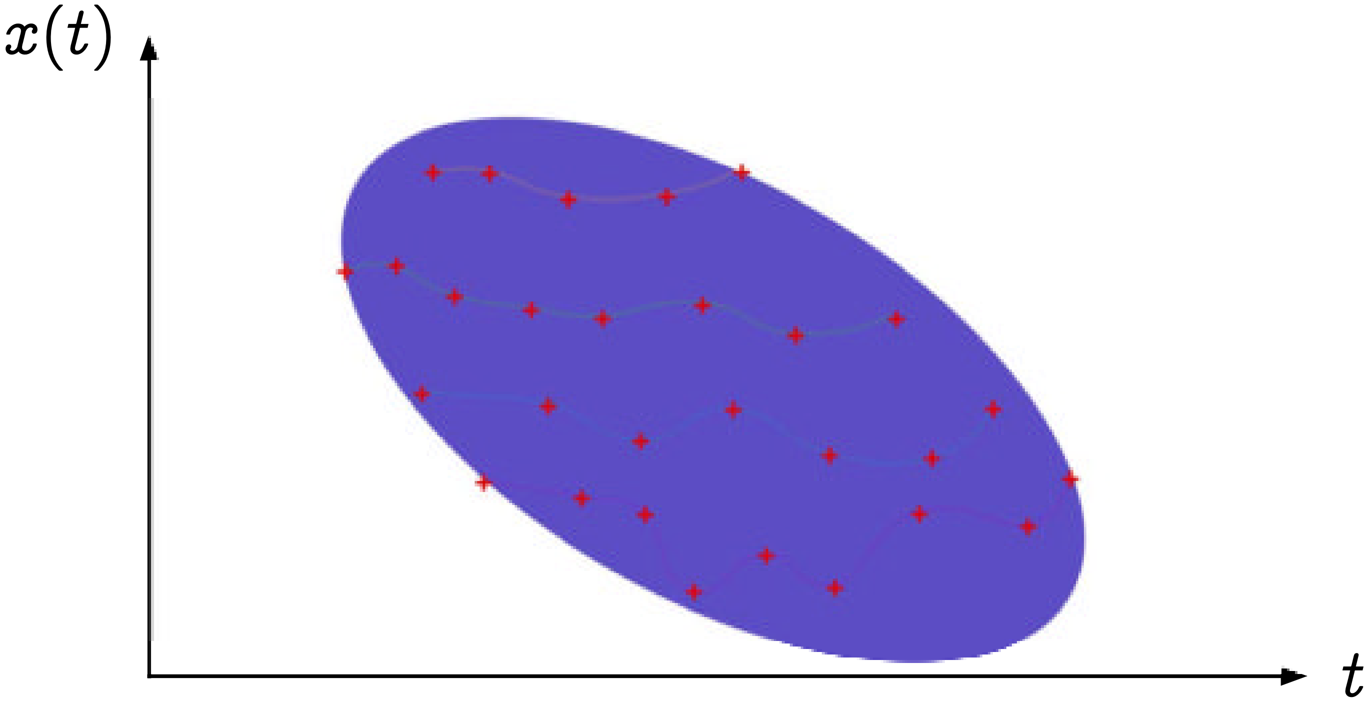
- No solution, but adding a new abstract domain:

  - representation of the abstract properties

  - abstract property transformers for language primitives
  - widening
  - reduction with other abstractions

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

Abstraction by ellipsoid for filters

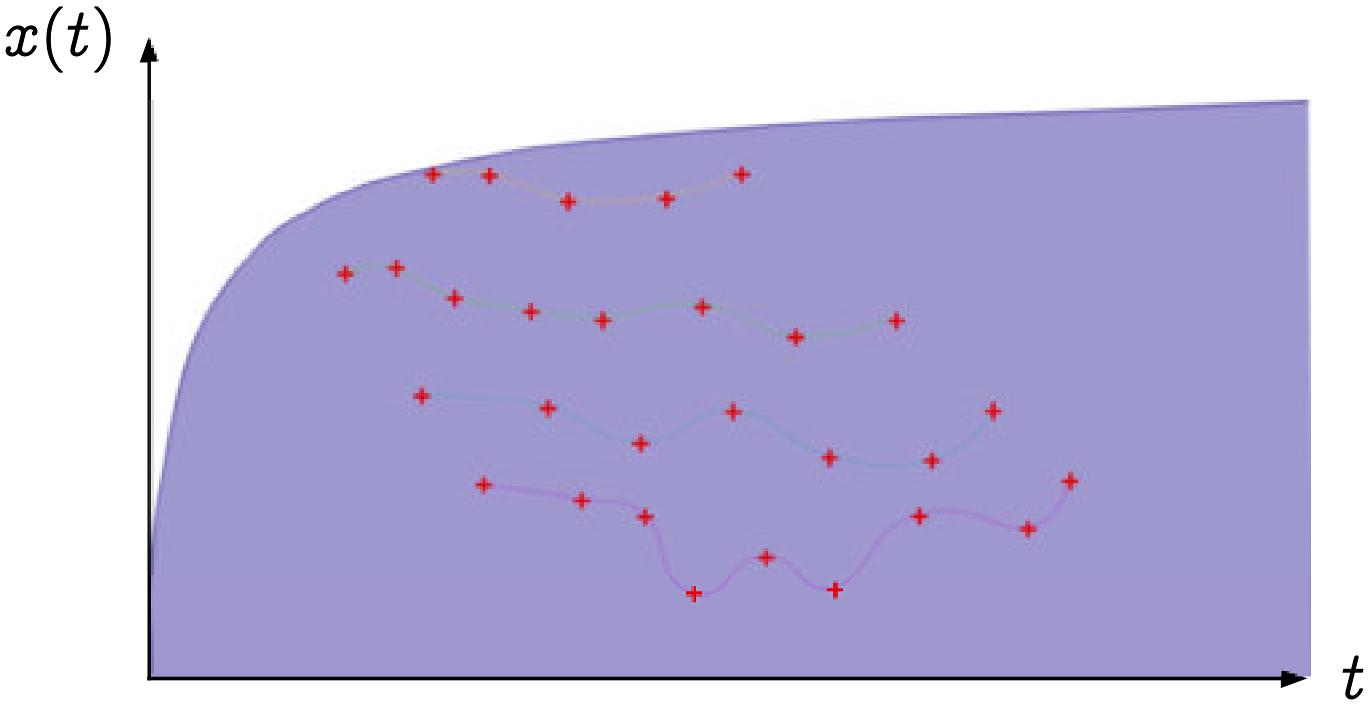


Ellipsoids $(x-a)^2 + (y-b)^2 \leq c$ [Fer05b]

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 $a^x \leq y$

Example of analysis by ASTRÉE

```
% 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));
```

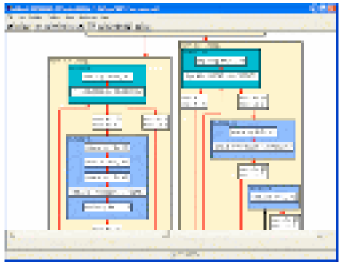
```
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1 +
1.19209290217e-07)^clock - 5.87747175411e-39
/ 1.19209290217e-07 <= 23.0393526881
```

7. Industrial application of abstract interpretation

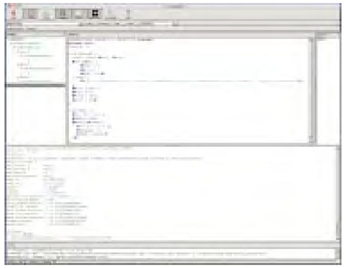
Examples of static analyzers in industrial use

– For C critical synchronous embedded control/command programs (for example for Electric Flight Control Software)

– aiT [FHL+01] is a static analyzer to determine the Worst Case Execution Time (to guarantee synchronization in due time)



– ASTRÉE [BCC+03] is a static analyzer to verify the absence of runtime errors



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 !

8. Conclusion

Conclusion

- Vision: to understand the numerical world, different levels of abstraction must be considered

- Theory: abstract interpretation ensures the coherence between abstractions and offers effective approximation techniques to cope with infinite systems

- Applications: the choice of effective abstraction which are coarse enough to be *computable* and precise enough to be *avoid false alarms* is central to master undecidability and complexity in model and program verification

The futur

- Software engineering : Manual validation by control of the software design process will be complemented by the verification of the final product

- Complex systems : abstract interpretation applies equally well to the analysis of systems with discrete evolution (image analysis [Ser94], biological systems [DFFK07, DFFK08, Fer07], quantum computation [JP06], etc)

THE END

Thank you for your attention

9. 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.

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

[Cou97] P. Cousot. Types as abstract interpretations, invited paper. In *24th 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.

[DFFK07] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable simulation of cellular signaling networks. In Zhong Shao, editor, *Proc. 5th APLAS '2007*, pages 139–157, Singapore, 29 Nov. –1 Dec. 2007. LNCS 4807, Springer.

[DFFK08] V. Danos, J. Feret, W. Fontana, and J. Krivine. Abstract interpretation of cellular signalling networks. In F. Loggozzo, D. Peled, and L.D. Zuck, editors, *Proc. 9th Int. Conf. VMCAI 2008*, pages 83–97, San Francisco, CA, US, 7–9 Jan. 2008. LNCS 4905, Springer.

[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.

[Fer07] J. Feret. Reachability analysis of biological signalling pathways by abstract interpretation. In T.E. Simos and G. Maroulis, editors, *Computation in Modern Science and Engineering: Proc. 6th Int. Conf. on Computational Methods in Sciences and Engineering (ICCMSE'07)*, volume American Institute of Physics Conf. Proc. 963 (2, Part A & B), pages 619–622. AIP, Corfu, GR, 25–30 Sep. 2007.

[FHL$^+$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. 1st Int. Work. EMSOFT '2001*, volume 2211 of *LNCS*, pages 469–485. Springer, 2001.

[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.

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

[Min06] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.

[Ser94] J. Serra. Morphological filtering: An overview. *Signal Processing*, 38:3–11, 1994.

Answers to questions

– The integers are encoded on 32 bits in C and on 31 bits in OCaml (one bit is used for garbage collection)

– The call of `fact(-1)` calls `fact(-2)` which calls `fact(-3)`, etc. For each call, it is necessary to stack the parameter and return address, which ends by a stack overflow:

```
% ocaml
        Objective Caml version 3.10.0
# let rec fact n = if (n = 1) then 1 else n * fact(n-1);;
val fact : int -> int = <fun>
# fact(-1);;
Stack overflow during evaluation (looping recursion?).
```