Abstract Interpretation–based Formal Verification of Complex Computer Systems

Patrick Cousot

Jerome C. Hunsaker Visiting Professor
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology

cousot@mit.edu   www.mit.edu/~cousot

École normale supérieure, Paris
cousot@ens.fr   www.di.ens.fr/~cousot

Minta Martin Lecture, May 13th, 2005

Software is everywhere

Software is replacing humans

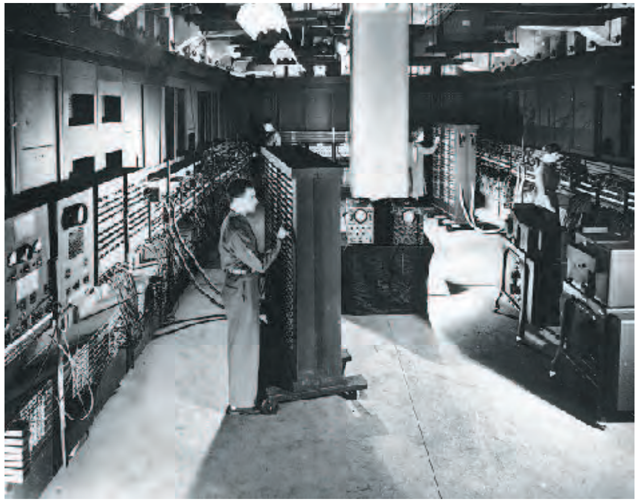
– Paris métro line 12 accident [1]: the driver was going too fast
– New high-speed métro line 14 (Météor): fully automated, no operators
– Software is in all mission-critical and safety-critical industrial infrastructures

[1] On August 30th, 2000, at the Notre-Dame-de-Lorette métro station in Paris, a car flipped over on its side and slid to a stop just a few feet from a train stopped on the opposite platform (24 injured).

Why bugs in software?

(1) Software gets huge

As computer hardware capacity grows...



ENIAC

5,000 flops [2]

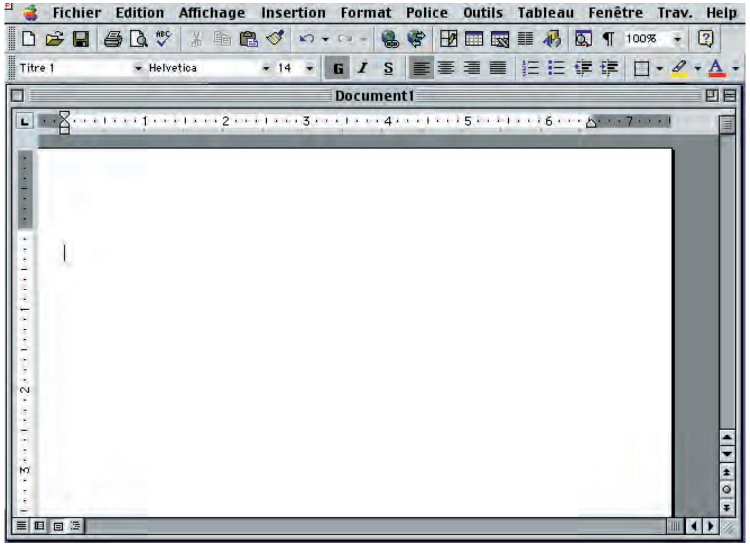
NEC Earth Simulator

35×10^{12} flops [3]

2 Floating point operations per second

3 10^{12} = Thousand Billion

Software size grows. . .



Text editor

1,700,000 lines of C [4]

Operating system

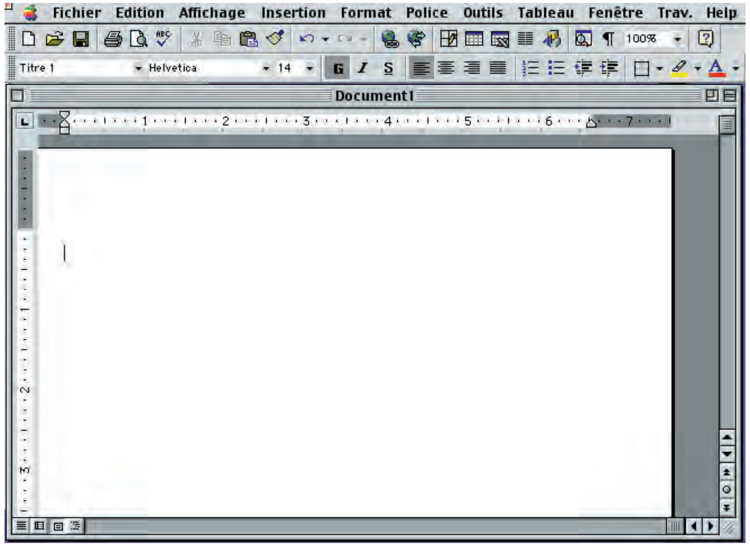
35,000,000 lines of C [5]

---

4 3 months for full-time reading of the code

5 5 years for full-time reading of the code

. . . and so does the number of bugs



Text editor

1,700,000 lines of C [4]

1,700 bugs (estimation)

Operating system

35,000,000 lines of C [5]

30,000 known bugs

---

4 3 months for full-time reading of the code

5 5 years for full-time reading of the code

(2) Computers are finite

Computers are finite

- Engineers use mathematics to deal with continuous, infinite structures (e.g. $\mathbb{R}$)
- Computers can only handle discrete, finite structures

Putting big things into small containers

– Numbers are encoded onto a limited number of bits (*binary digits*)
– Some operations may overflow (e.g. integers: 32 bits $\times$ 32 bits = 64 bits)
– Using different number sizes (32, 64, ... bits) can also be the source of over-flows

The Ariane 5.01 maiden flight

– June 4th, 1996 was the maiden flight of Ariane 5

The Ariane 5.01 maiden flight failure

- June 4th, 1996 was the maiden flight of Ariane 5
- The launcher was detroyed after 40 seconds of flight because of a software overflow [6]

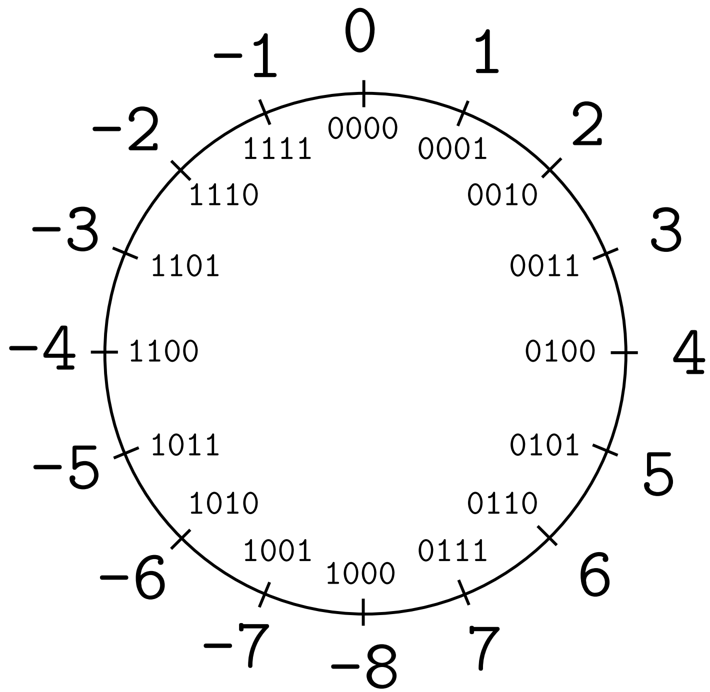
[6] A 16 bit piece of code of Ariane 4 had been reused within the new 32 bit code for Ariane 5. This caused an uncaught overflow, making the launcher uncontrolable.

(3) Computers go round

Modular arithmetic...

– Todays, computers avoid integer overflows thanks to modular arithmetic

– Example: integer 2's complement encoding on 8 bits



. . . can be contrary to common sense

```
# 1073741823 + 1;;
- : int = -1073741824
# -1073741824 - 1;;
- : int = 1073741823
# -1073741824 ÷ -1;;
- : int =
```

... can be contrary to common sense

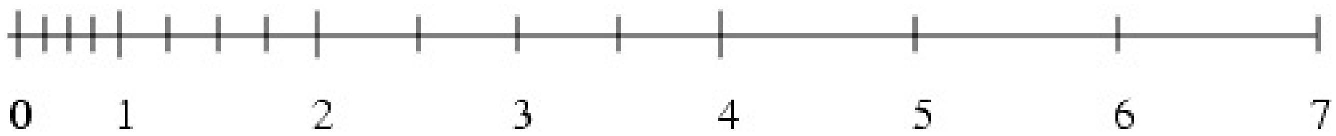
```
# 1073741823 + 1;;
- : int = -1073741824
# -1073741824 - 1;;
- : int = 1073741823
# -1073741824 ÷ -1;;
- : int = -1073741824
```

Mapping many to few

– Reals are mapped to floats (floating-point arithmetic)

$$\pm d_0.d_1 d_2 \ldots d_{p-1} \beta^e$$ [7]

– For example on 6 bits (with $p = 3$, $\beta = 2$, $e_{\min} = -1$, $e_{\max} = 2$), there are 32 normalized floating-point numbers. The 16 positive numbers are



7 where - $d_0 \neq 0$,
- p is the number of significative digits,
- β is the basis (2), and
- e is the exponant ($e_{\min} \leq e \leq e_{\max}$)

Rounding

- Computations returning reals that are not floats, must be rounded
- Most mathematical identities on $\mathbb{R}$ are no longer valid with floats
- Rounding errors may either compensate or accumulate in long computations
- Computations converging in the reals may diverge with floats (and ultimately overflow)

Example of rounding error

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

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

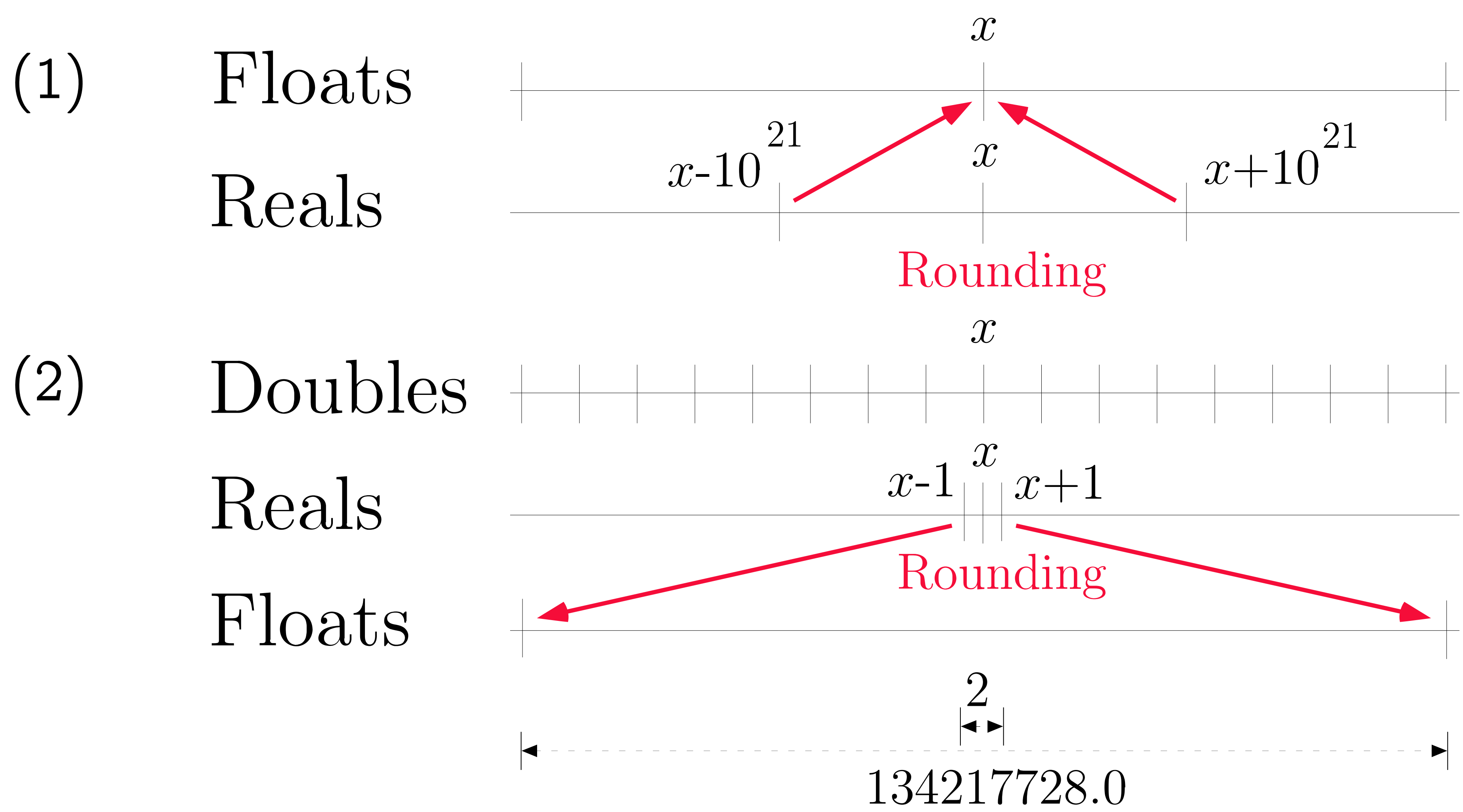
Example of rounding error

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

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

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

Explanation of the huge rounding error



Example of accumulation of small rounding errors

```
% ocaml
        Objective Caml version 3.08.1
# let x = ref 0.0;;
val x : float ref = {contents = 0.}
# for i = 1 to 1000000000 do
      x := !x +. 1.0/.10.0
  done; x;;
- : float ref = {contents = 99999998.7454178184}
```

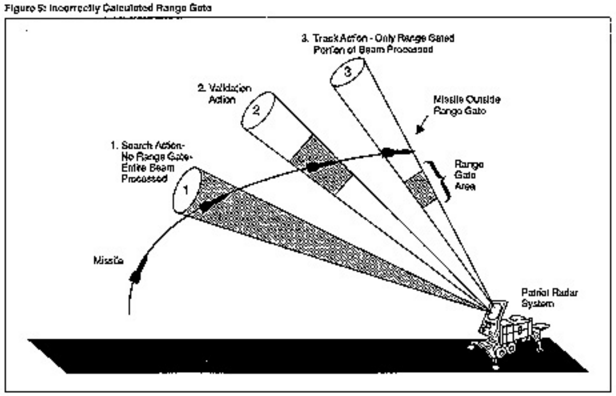
since $(0.1)_{10} = (0.0001100110011001100\ldots)_2$

The Patriot missile failure

– "On February 25th, 1991, a Patriot missile … failed to track and intercept an incoming Scud[8]."

– The software failure was due to a cumulated rounding error[9]

Figure 5: Incorrectly Calculated Range Gate



---

[8] This Scud subsequently hit an Army barracks, killing 28 Americans.

[9]
- "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"

What can be done about bugs?

Warranty

Excerpt from an GPL open software licence:

NO WARRANTY. . . . *BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.*

Warranty

Excerpt from an GPL open software licence:

> **NO WARRANTY.** . . . *BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.*

You get nothing for free!

Warranty

Excerpt from Microsoft software licence:

DISCLAIMER OF WARRANTIES. ... *MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE, AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND MICROSOFT AND ITS SUPPLIERS HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. ...*

Warranty

Excerpt from Microsoft software licence:

DISCLAIMER OF WARRANTIES. . . . *MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE, AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND MICROSOFT AND ITS SUPPLIERS HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE.* . . .

You get nothing for your money either!

Traditional software validation methods

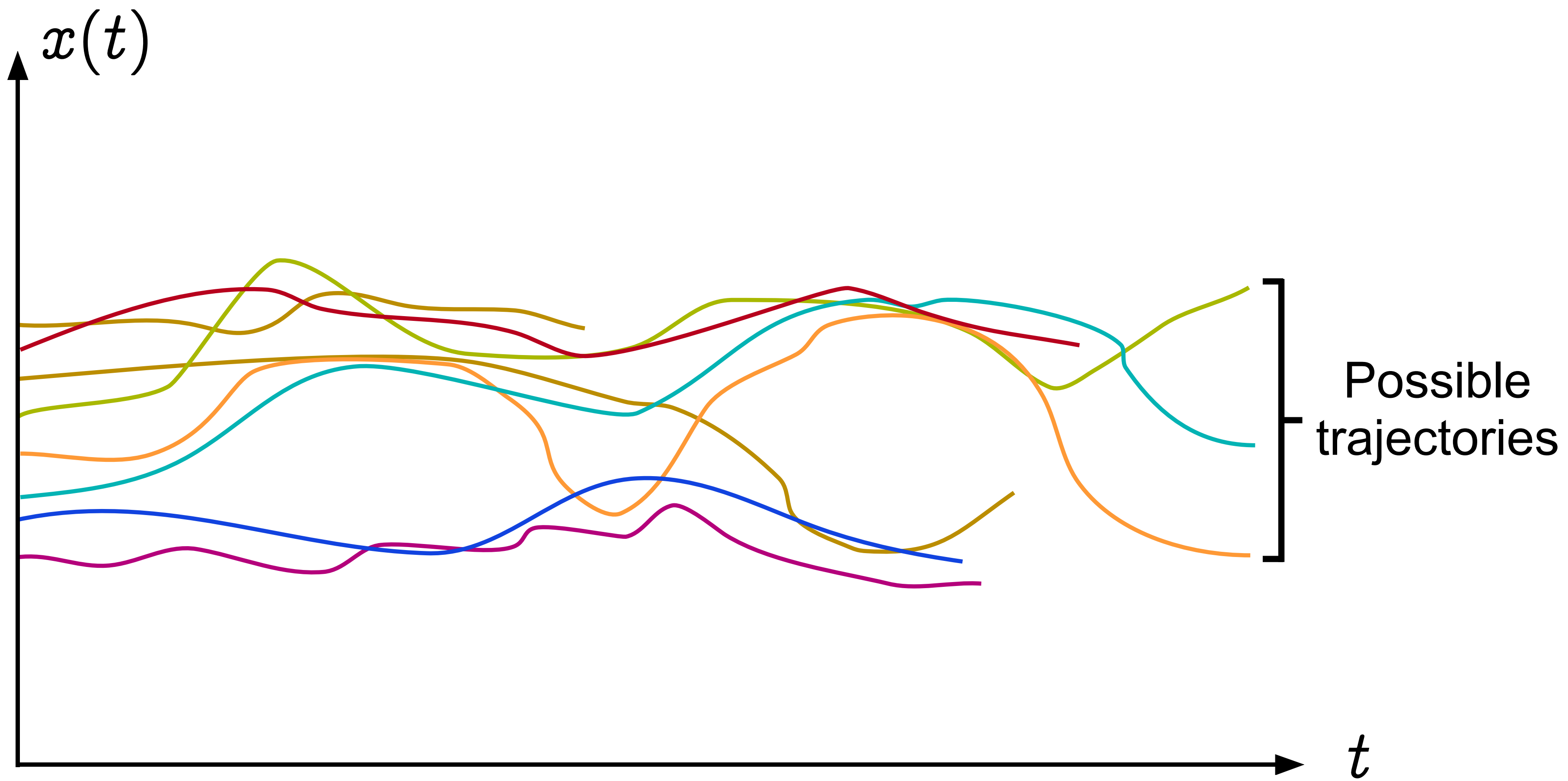
- The law cannot enforce more than "best practice"
- Manual software validation methods (code reviews, simulations, tests, etc.) do not scale up
- The capacity of programmers/computer scientists remains essentially the same
- The size of software teams cannot grow significantly without severe efficiency losses

Mathematics and computers can help

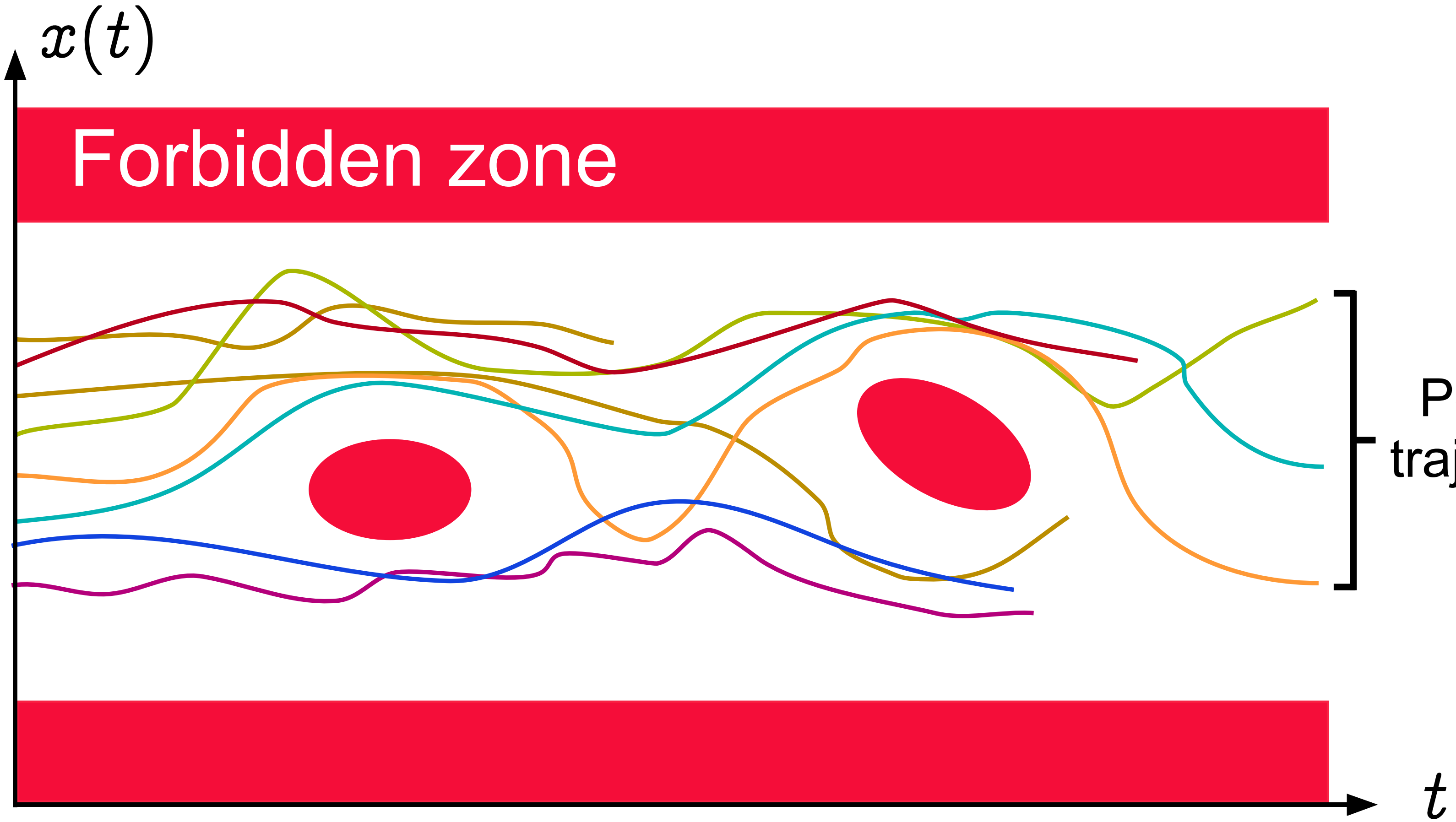
- Software behavior can be mathematically formalized $\rightarrow$ semantics
- Computers can perform semantics-based program analyses to realize verification $\rightarrow$ static analysis
  - but computers are finite so there are intrinsic limitations $\rightarrow$ undecidability, complexity
  - which can only be handled by semantics approximations $\rightarrow$ abstract interpretation

Abstract interpretation (1) very informally

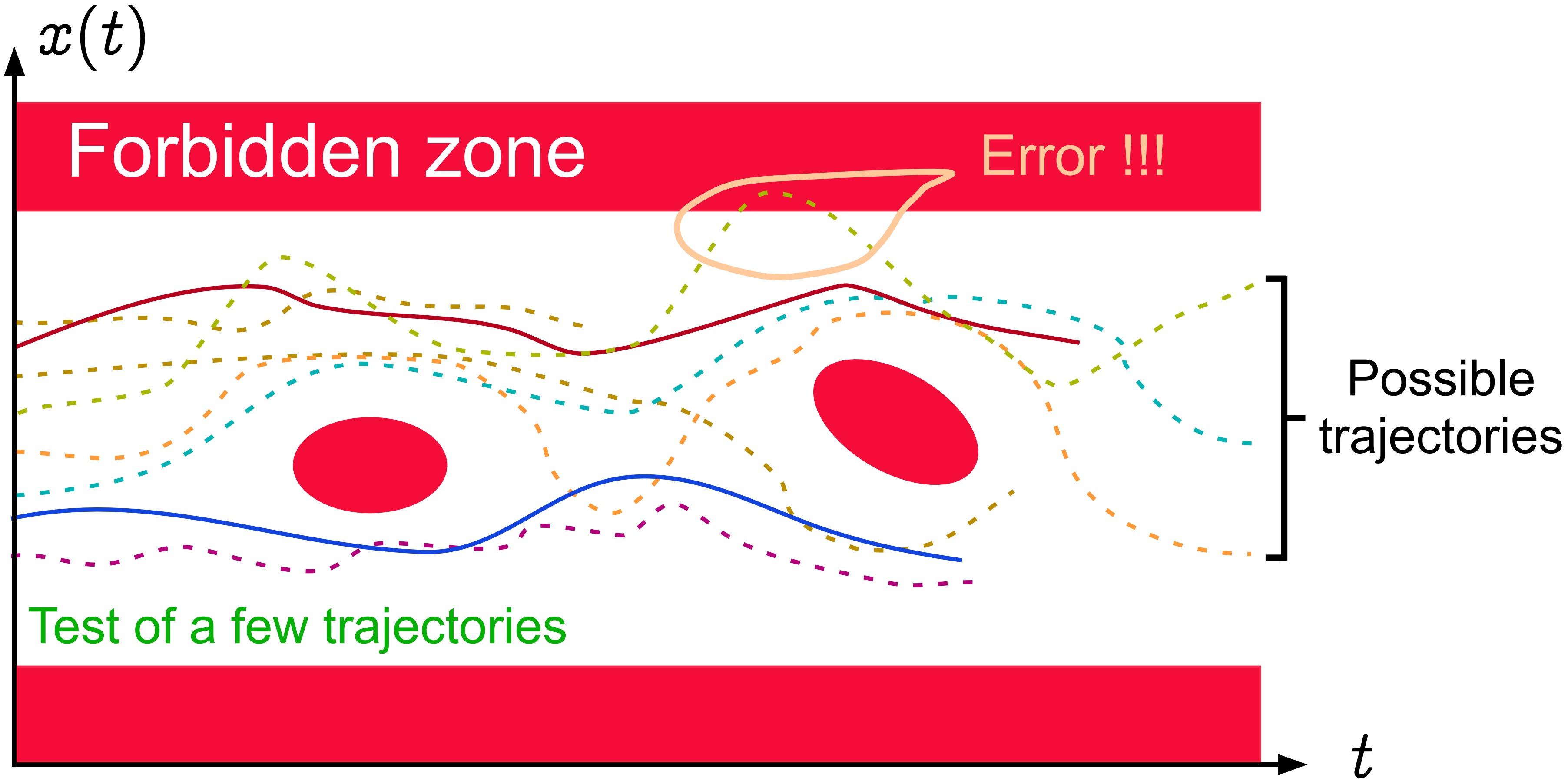
Operational semantics



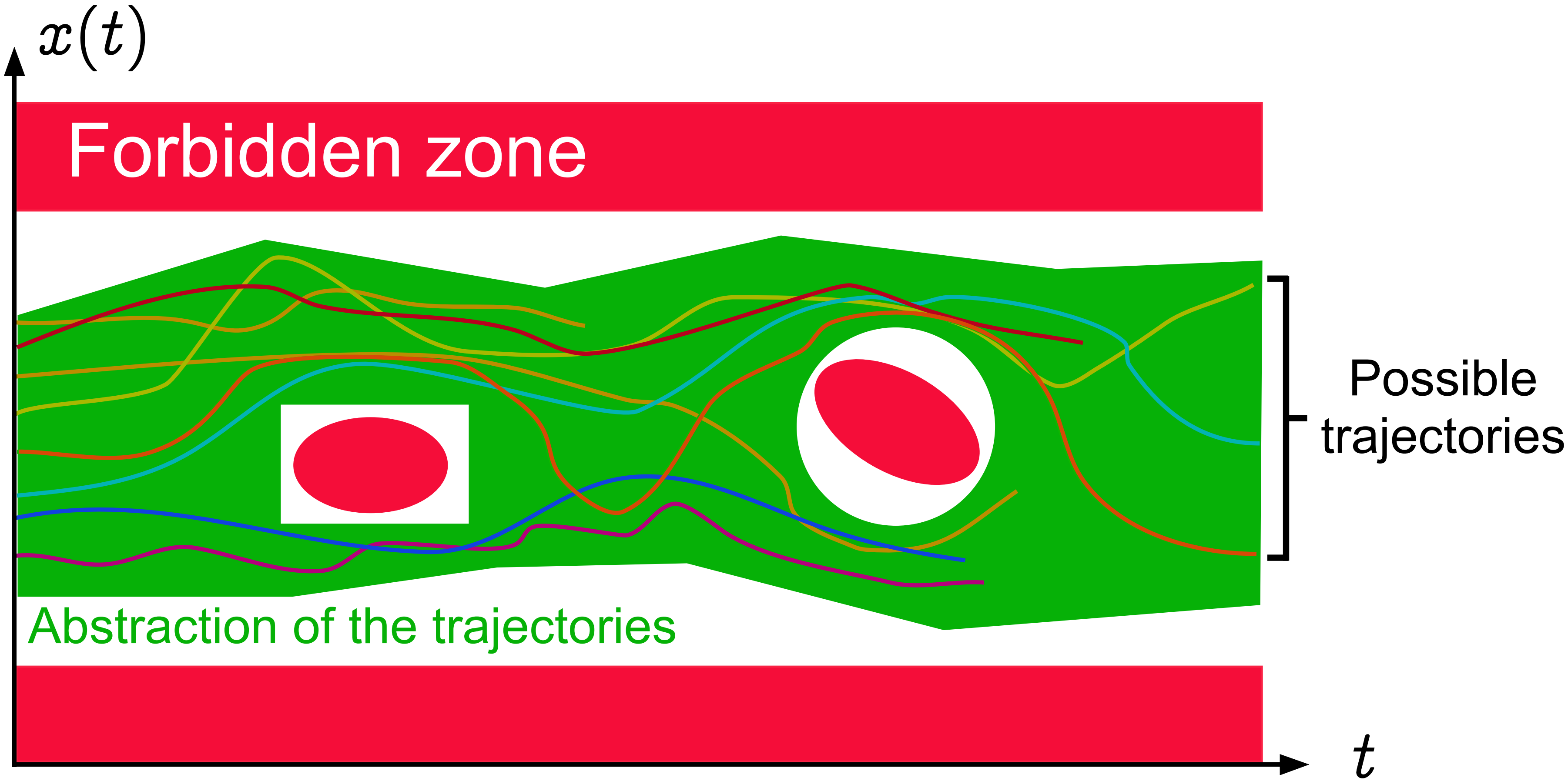
Safety property



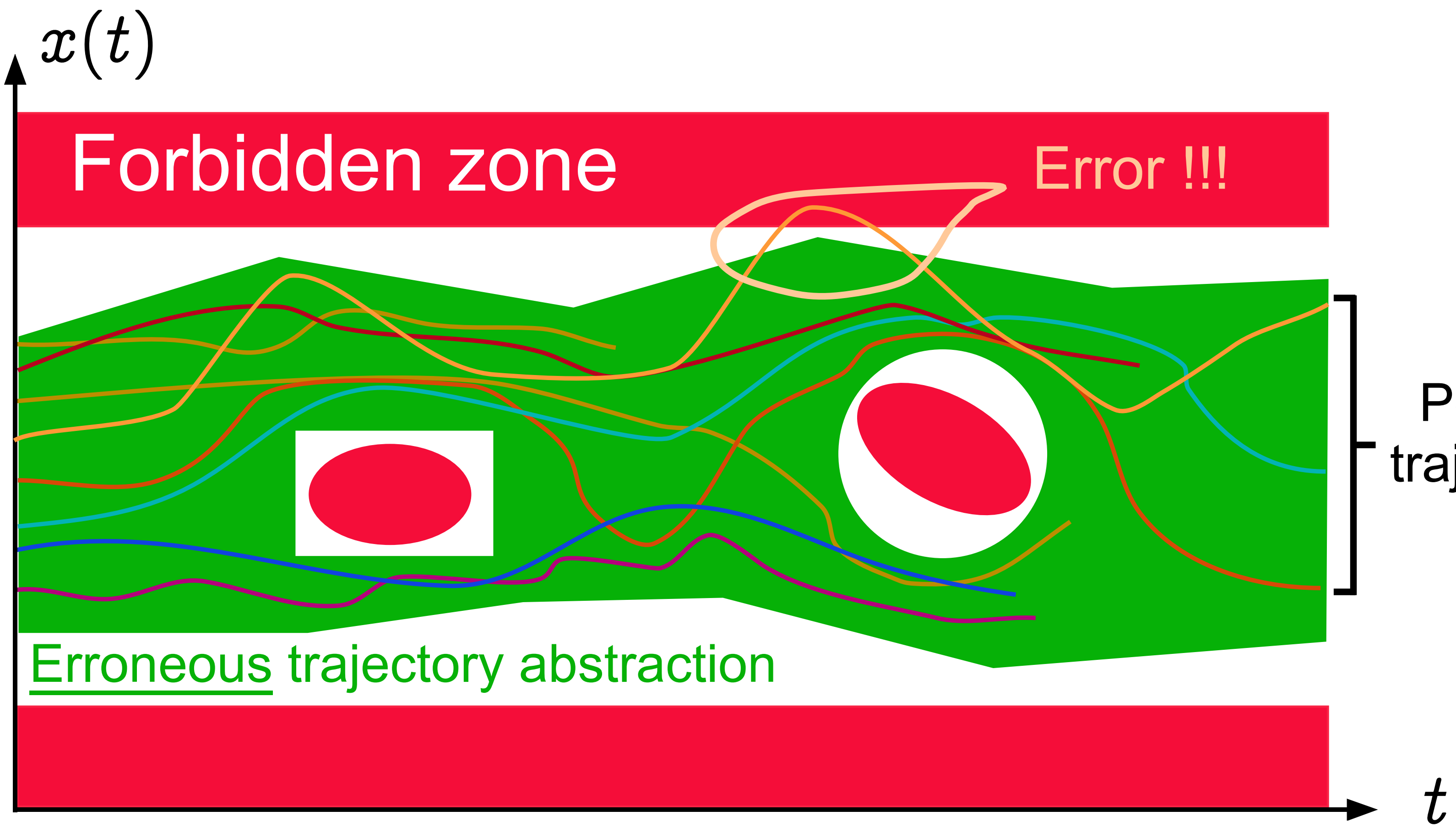
Test/debugging is unsafe



Abstract interpretation is safe



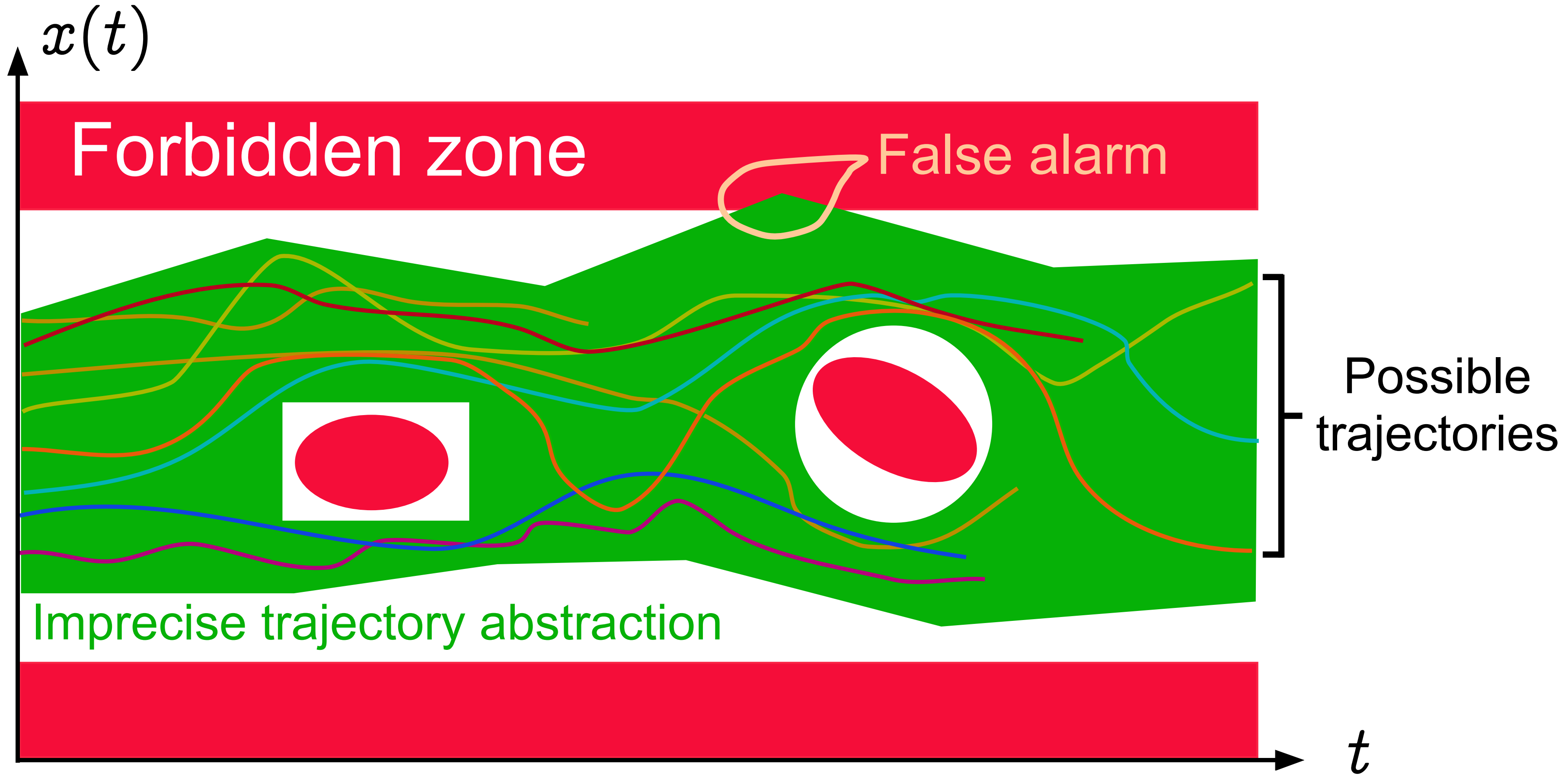
Soundness requirement: erroneous abstraction [10]



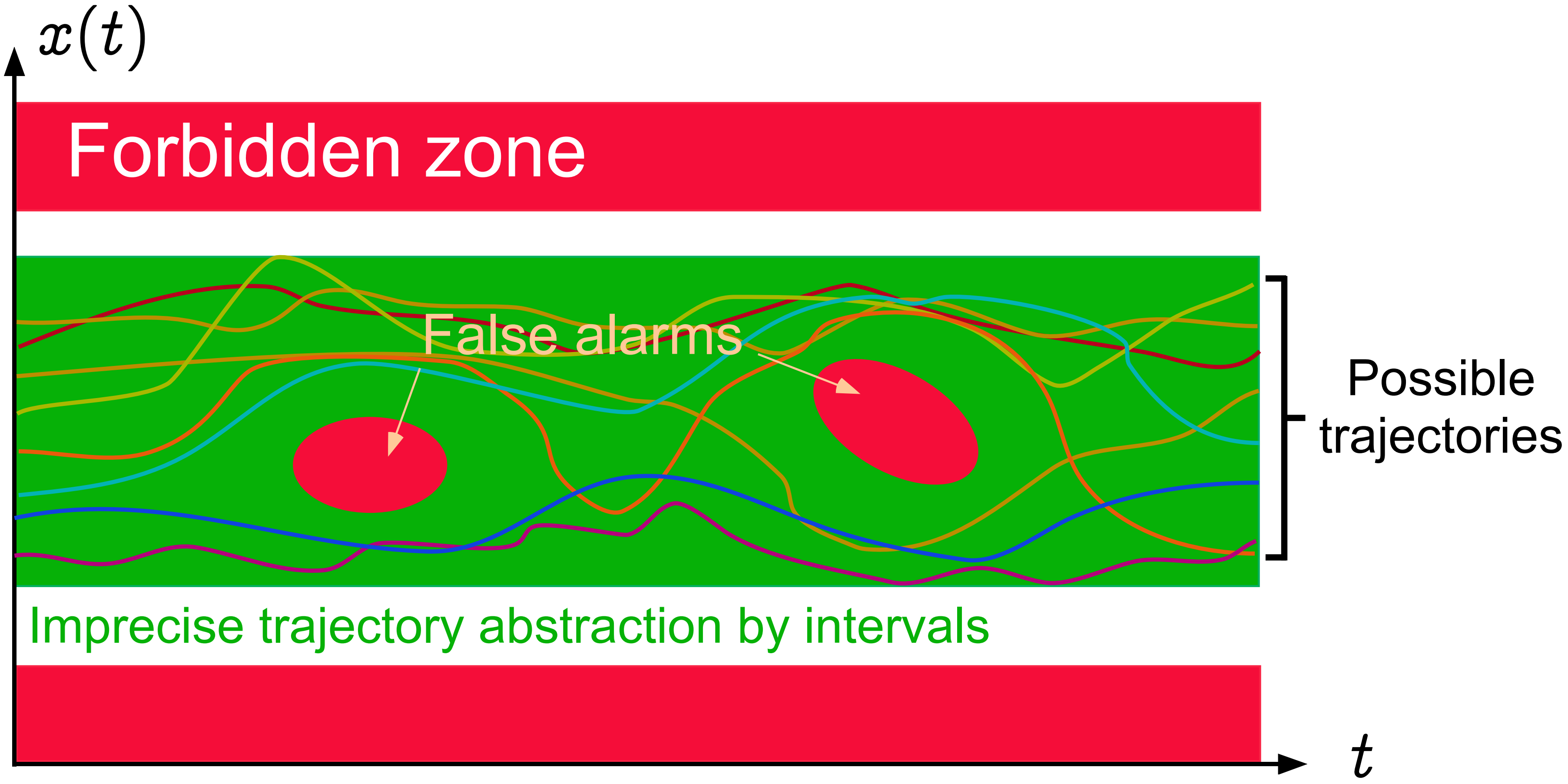
---

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

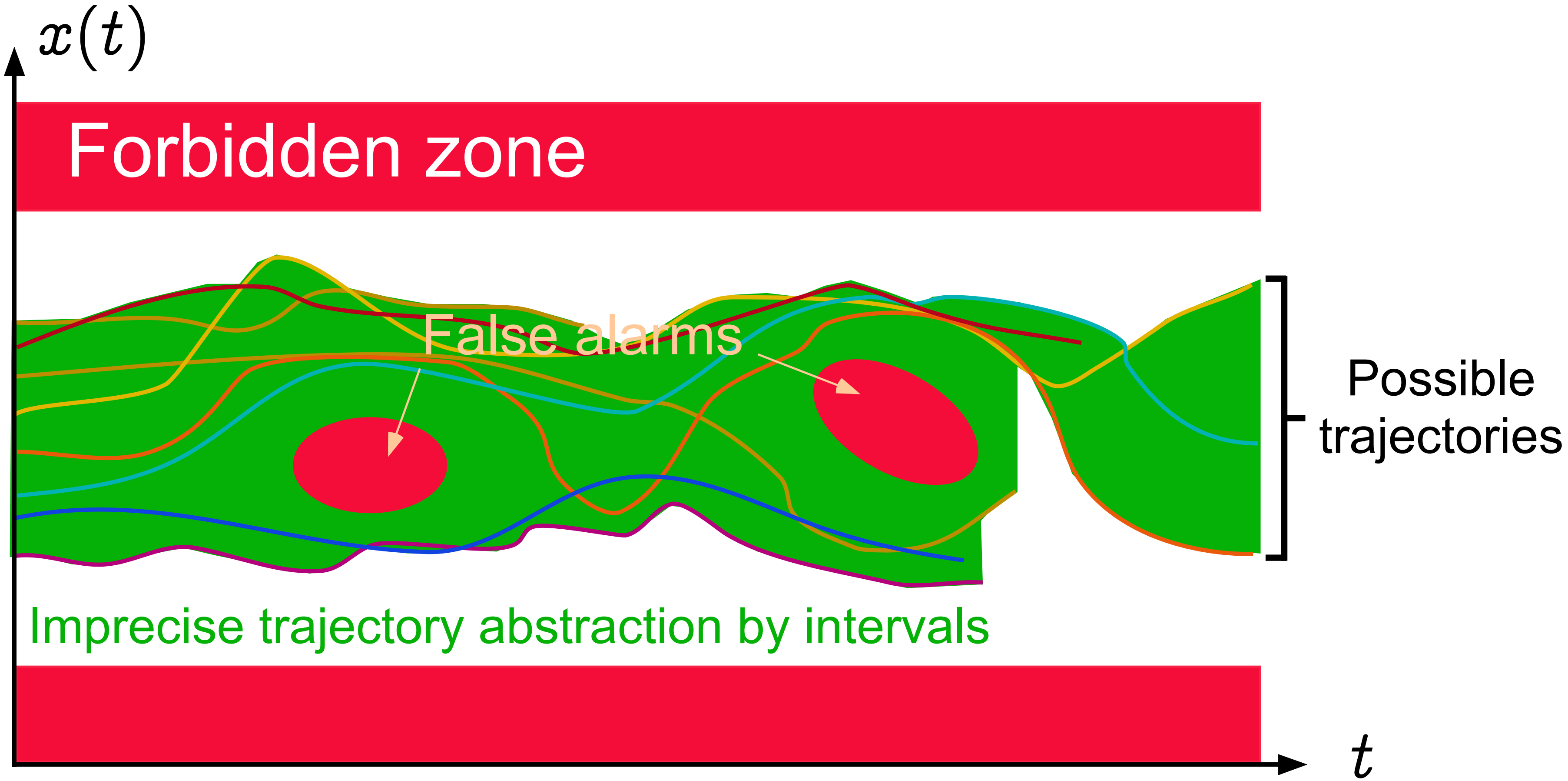
Imprecision $\Rightarrow$ false alarms



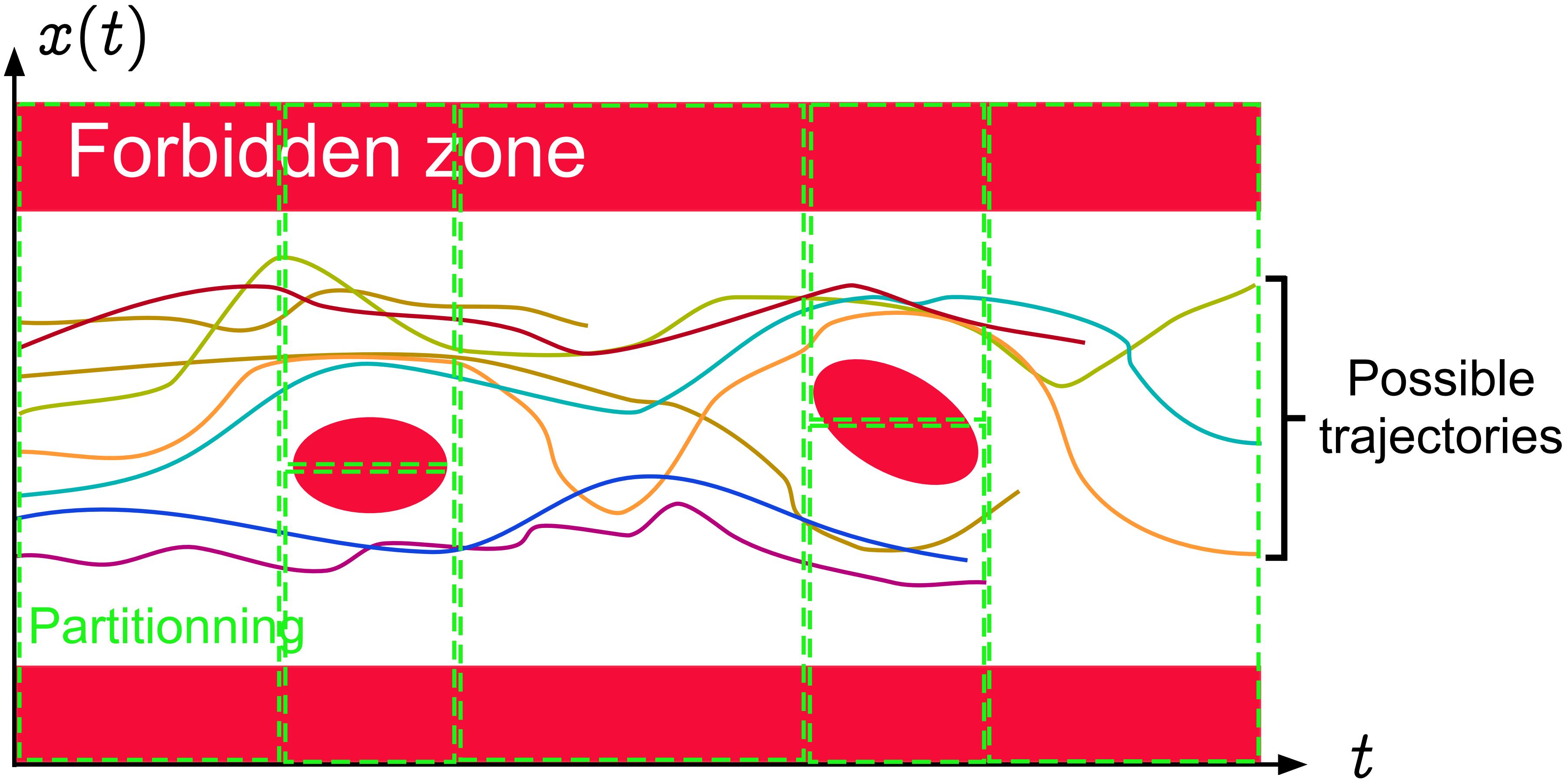
Global interval abstraction → false alarms



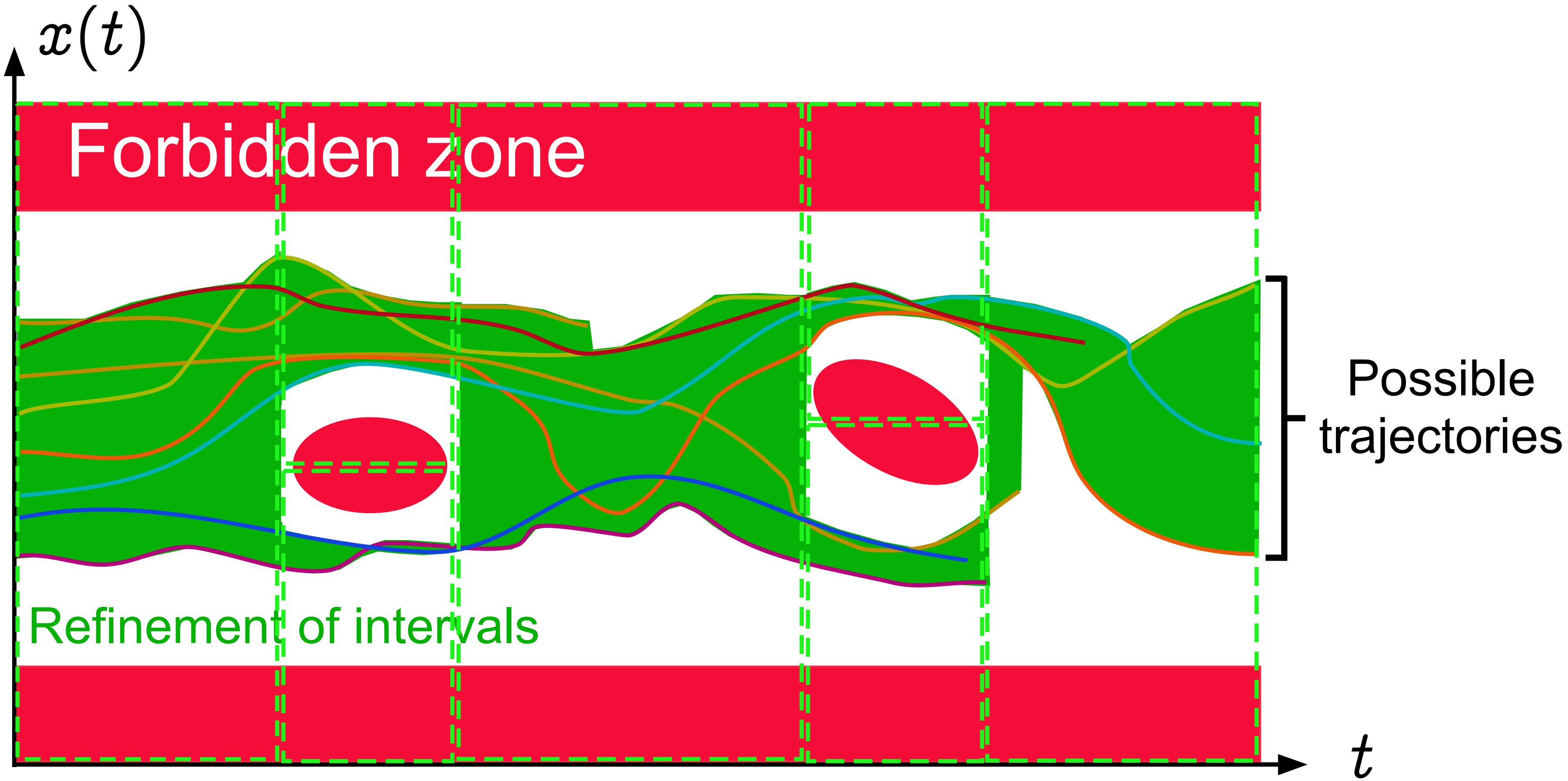
Local interval abstraction $\rightarrow$ false alarms



Refinement by partitionning



Intervals with partitionning



The ASTRÉE static analyzer

C programming language

with:

- boolean, integer & floating point computations
- pointers (on functions, etc), structures & arrays
- tests, loops and function calls
- limited branching (forward `goto`, `break`, `continue`)

without:

`union`, dynamic memory allocation, recursive function calls, unstructured backward branching, conflicting side effects [11], C libraries

[11] The ASTRÉE analyzer checks the absence of ambiguous side effects since otherwise the semantics of the C program would not be defined deterministically

Operational semantics

- International norm of C (ISO/IEC 9899:1999)
- *restricted by* implementation-specific behaviors depending upon the machine and compiler [12]
- *restricted by* user-defined programming guidelines [13]
- *restricted by* program specific user requirements [14]
- *restricted by* a volatile environment as specified by a *trusted* configuration file.

---

12 e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles

13 e.g. no modular arithmetic for signed integers, even though this might be the hardware choice

14 e.g. `assert`

Implicit specification: absence of runtime errors

– No violation of the norm of C [15]

– No implementation-specific undefined behaviors [16]

– No violation of the programming guidelines [17]

– No violation of the programmer assertions [18]

---

15 e.g. array index out of bounds

16 e.g. maximum short integer is 32767, no float overflow

17 e.g. static variables are not be assumed to be initialized to 0

18 must all be statically verified

Application domain

- Safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems [19]
- Strictly disciplined programming methodology
- 75% of the code is automatically generated from a high-level specification language [20]
- The external controlled system is unknown (but for the range of a few volatile variables, maximal duration, ... as specified in the configuration file)

---

19 e.g. flight control software, engine control software

20 e.g. S.A.O. (proprietary), Simulink, SCADE

Verification of flight control software

– Primary flight control software of the Airbus A340 family and the A380 digital fly-by-wire systems

– Most critical software on board [21]

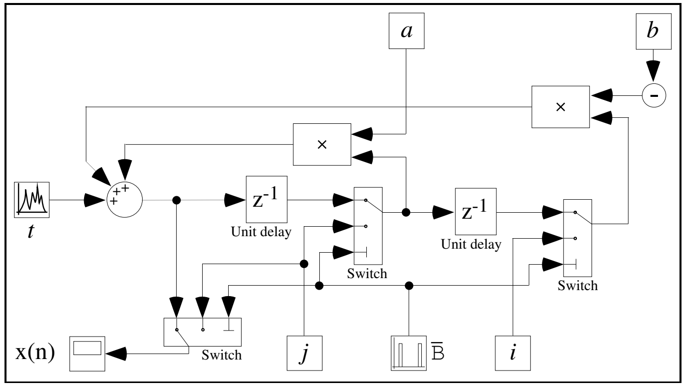
– ASTRÉE verifies the absence of runtime errors without any false alarms!

[21] controls automatically the airplane surface deflections and power settings, performs envelope protection, . . . with precedence over the pilot

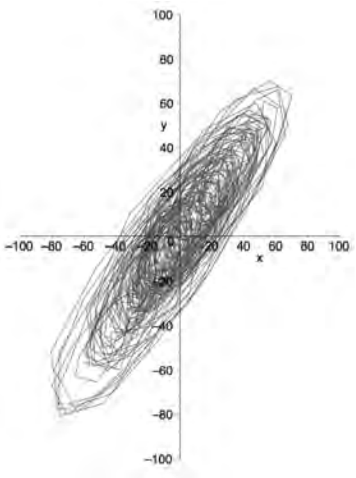
Examples of abstractions in ASTRÉE

Ellipsoid Abstract Domain for Filters

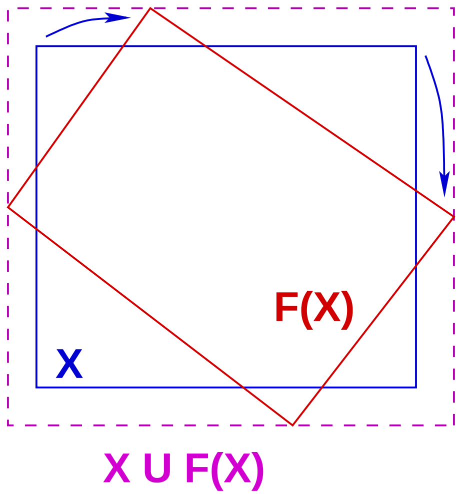
2^d Order Digital Filter:



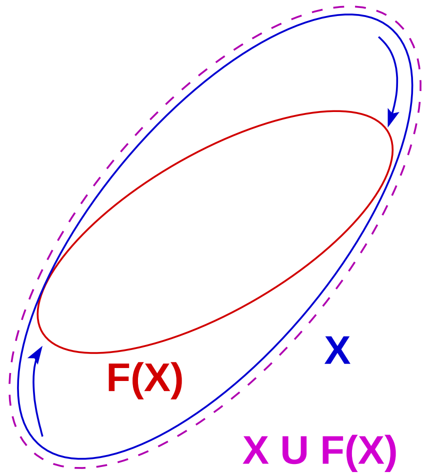
- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract
- Polyhedral approximations are unstable
- The simplest stable surface is an ellipsoid



execution trace



unstable interval



stable ellipsoid

Filter Example

```
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

```
% 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
```

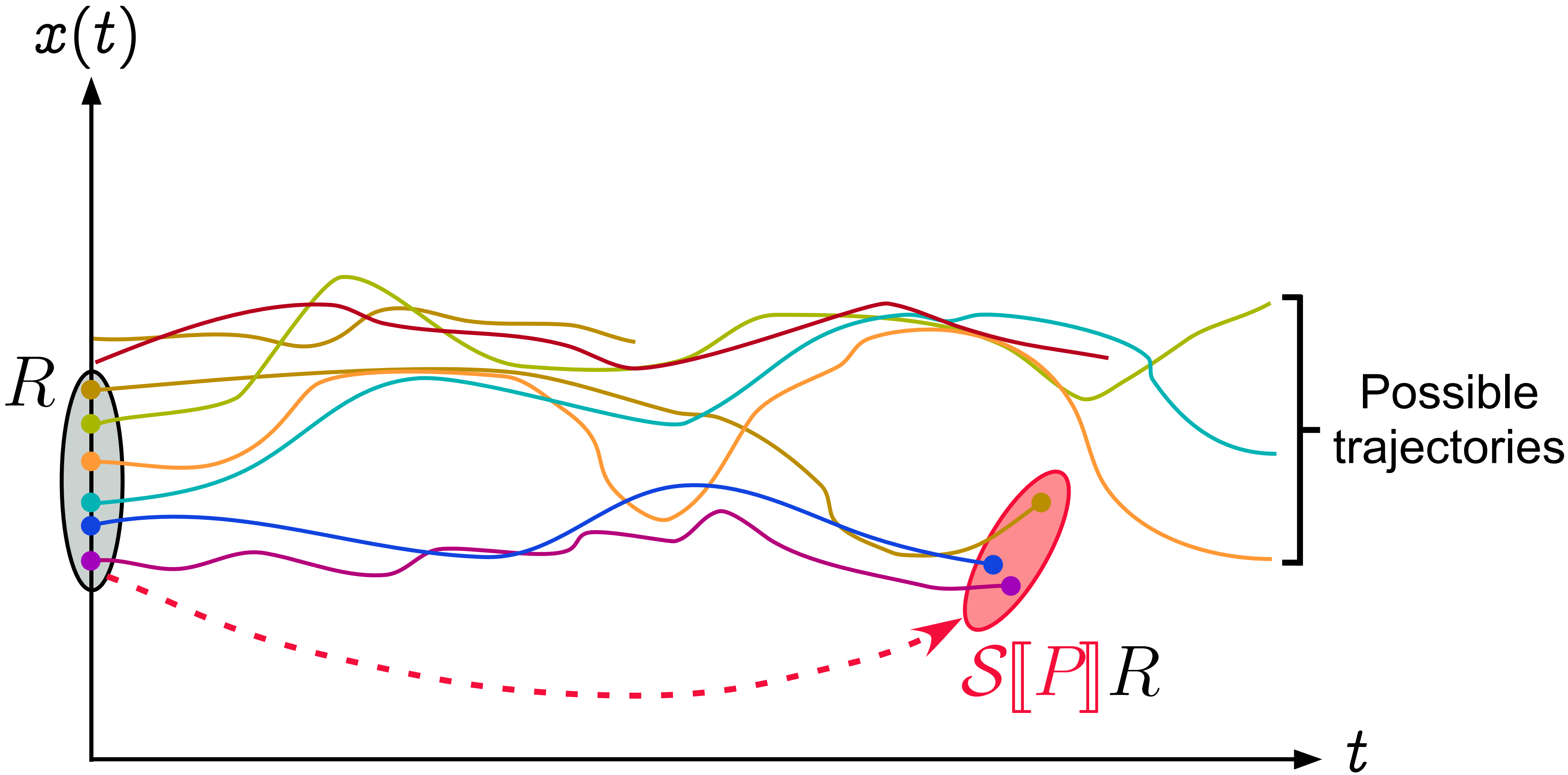
Abstract interpretation (2) with a touch of formalism

Semantics

Syntax of programs

X	variables $X \in \mathbb{X}$
T	types $T \in \mathbb{T}$
E	arithmetic expressions $E \in \mathbb{E}$
B	boolean expressions $B \in \mathbb{B}$
$D ::= T\ X\,;$	
$\quad\mid\ T\ X\ ;\ D'$	
$C ::= X = E;$	commands $C \in \mathbb{C}$
$\quad\mid\ \texttt{while}\ B\ C'$	
$\quad\mid\ \texttt{if}\ B\ C'\ \texttt{else}\ C''$	
$\quad\mid\ \{\ \mathrm{C}_1\ \ldots\ \mathrm{C}_n\ \},\ (n \geq 0)$	
$P ::= D\ C$	program $P \in \mathbb{P}$

Final states semantics



States

Values of given type:

$$\mathcal{V}[\![T]\!] \quad : \quad \text{values of type } T \in \mathbb{T}$$

$$\mathcal{V}[\![\texttt{int}]\!] \stackrel{\text{def}}{=} \{z \in \mathbb{Z} \mid \texttt{min\_int} \leq z \leq \texttt{max\_int}\}$$

Program states $\Sigma[\![P]\!]$ [22]:

$$\Sigma[\![D\ C]\!] \stackrel{\text{def}}{=} \Sigma[\![D]\!]$$

$$\Sigma[\![T\ X\,;]\!] \stackrel{\text{def}}{=} \{X\} \mapsto \mathcal{V}[\![T]\!]$$

$$\Sigma[\![T\ X\,;\ D]\!] \stackrel{\text{def}}{=} (\{X\} \mapsto \mathcal{V}[\![T]\!]) \cup \Sigma[\![D]\!]$$

---

22 States $\rho \in \Sigma[\![P]\!]$ of a program P map program variables X to their values $\rho(X)$

Final states semantics

$$\mathcal{S}[\![X = E;]\!]R \stackrel{\text{def}}{=} \{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in R\}$$

$$\rho[X \leftarrow v](X) \stackrel{\text{def}}{=} v, \qquad \rho[X \leftarrow v](Y) \stackrel{\text{def}}{=} \rho(Y)$$

$$\mathcal{S}[\![\texttt{if } B\ C' \texttt{ else } C'']\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]R) \cup \mathcal{S}[\![C'']\!](\mathcal{B}[\![\neg B]\!]R)$$

$$\mathcal{B}[\![B]\!]R \stackrel{\text{def}}{=} \{\rho \in R \mid B \text{ holds in } \rho\}$$

$$\mathcal{S}[\![\texttt{while } B\ C']\!]R \stackrel{\text{def}}{=} \text{let } \mathcal{W} = \mathsf{lfp}^{\subseteq}\ \lambda \mathcal{X} \cdot R \cup \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]\mathcal{X}) \text{ in } (\mathcal{B}[\![\neg B]\!]\mathcal{W})$$

$$\mathcal{S}[\![\{\}]\!]R \stackrel{\text{def}}{=} R$$

$$\mathcal{S}[\![\{C_1 \ldots C_n\}]\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C_n]\!] \circ \ldots \circ \mathcal{S}[\![C_1]\!]R \quad n > 0$$

$$\mathcal{S}[\![D\ C]\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C]\!](R) \quad (R \subseteq \Sigma[\![D]\!], \text{ initial states})$$

Undecidability

Undecidability

– The program's semantics, which is an infinite object, is not computable by a finite device

– All non-trivial questions about a program's semantics are undecidable (no computer can always answer, for sure, in a finite amount of time)

– Example: termination [23]

---

[23]

- Assume `Termination(P)` is a terminating program answering correctly the following question about any program P (P is a parameter encoded as text): *Are all trajectories of P finite?*
- A contradiction immediately appears when considering the program which text is:

```
program Goedel(P);
while termination(P) do {} od
```

- So termination is undecidable (whence so is any interesting semantic program property)

Complexity

Polynomial Time Complexity

– Polynomial-time computability is identified with the intuitive notion of algorithmic efficiency

– Intuitively valid only for small powers:

n	Execution time at 10^9 ops/s			
	$\mathcal{O}(n)$	$\mathcal{O}(n.log(n))$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
1	ϵ	ϵ	ϵ	ϵ
10	ϵ	ϵ	$0.1\mu s$	$1\mu s$
10^3	$1\mu s$	$6\mu s$	1ms	1s
10^6	1ms	13ms	16mn	32 years
10^9	1s	20s	32 years	300 000 000 centuries
10^{12}	16mn	7.7h	300 000 centuries	—
10^{15}	11.6 days	1 year	—	—

Abstract interpretation

Property abstraction

- $\langle \wp(\Sigma[\![P]\!]),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L,\ \sqsubseteq\rangle$
- L encodes abstractions of properties in $\wp(\Sigma[\![P]\!])$
- $\sqsubseteq$ abstracts implication $\subseteq$ [24]
- $\alpha(I)$ encodes an overapproximation of property I [25]
- $\gamma(\overline{I})$ is the meaning of the abstract property $\overline{I}$
- Approximation is from above $I \subseteq \gamma \circ \alpha(I)$
- In case of best approximation ($\alpha \circ \gamma(\overline{I}) \sqsubseteq \overline{I}$), $\langle \alpha,\ \gamma\rangle$ is a Galois connection

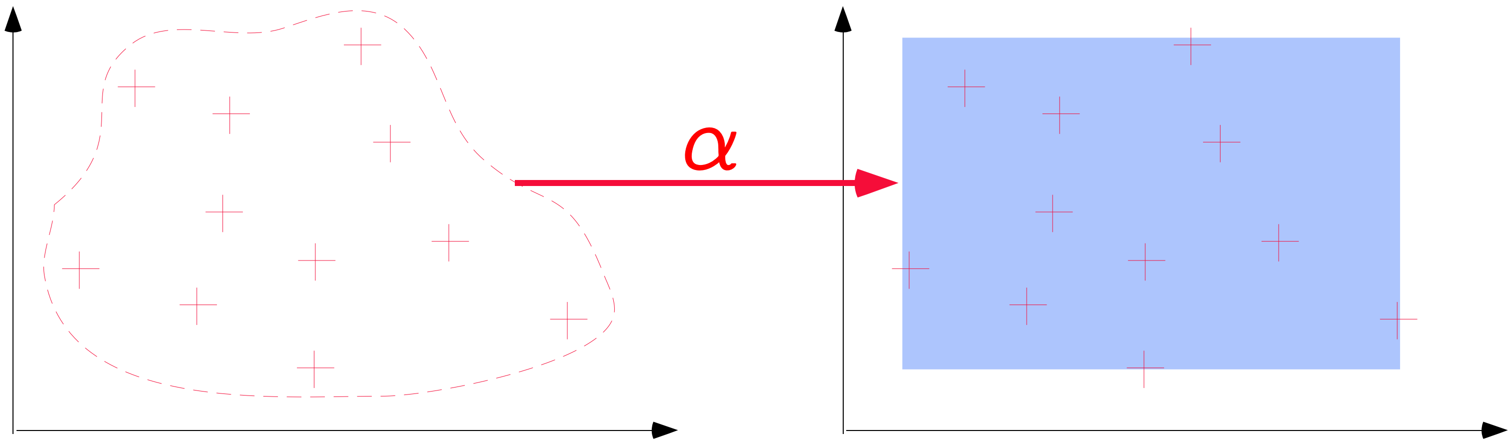
---

24 α and γ order preserving

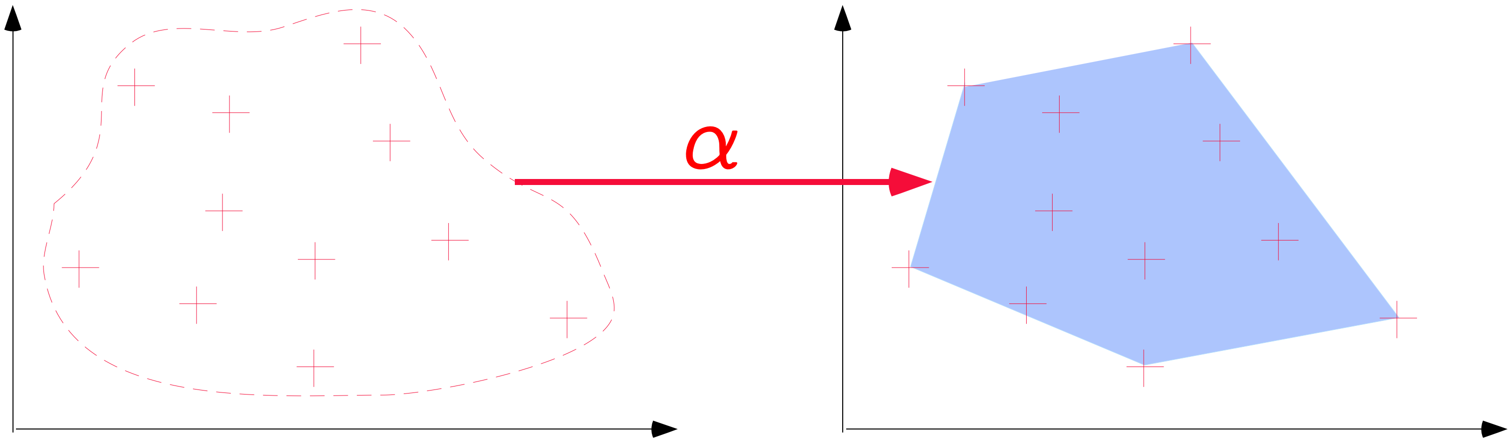
25 e.g. α(set of points) = polyhedron and γ(polyhedron) = set of interior points

Examples

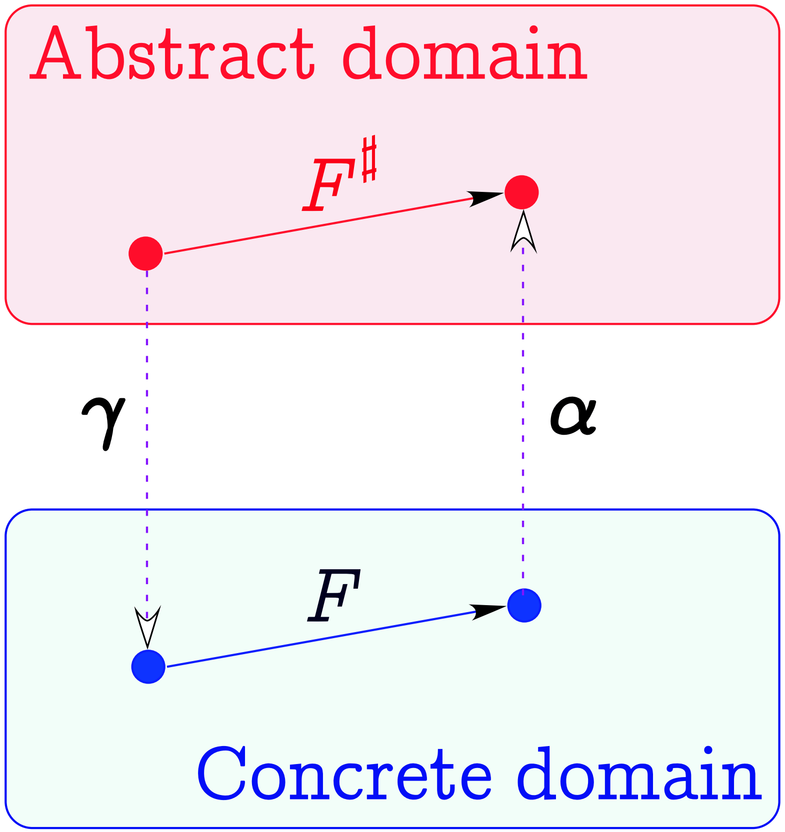
Interval abstraction:



Polyhedral abstraction:



Function Abstraction

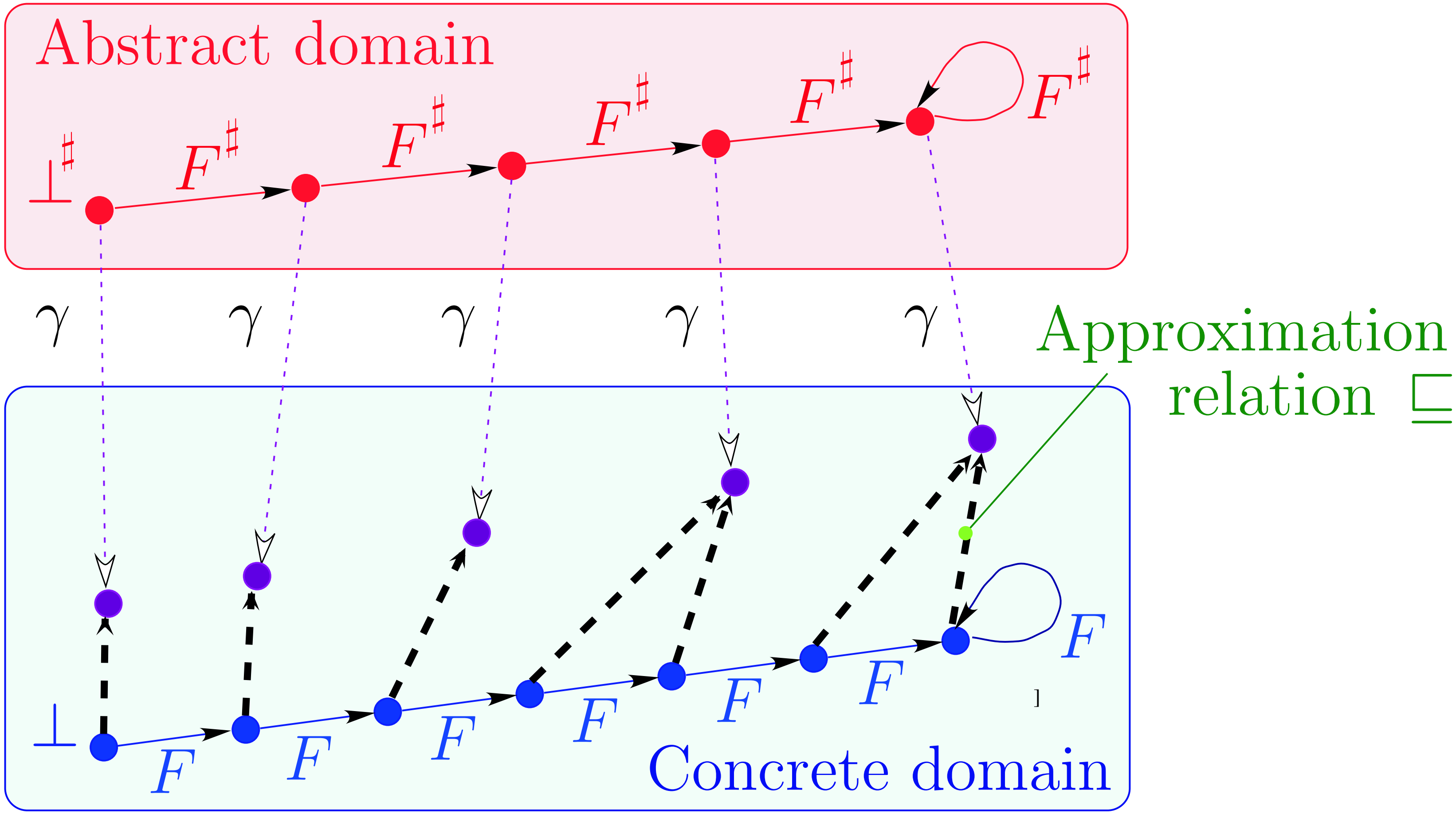


$$F^\sharp = \alpha \circ F \circ \gamma$$

$$\langle P,\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q,\ \sqsubseteq\rangle \Rightarrow$$

$$\langle P \xrightarrow{\text{mon}} P,\ \dot{\subseteq}\rangle \underset{\lambda F \cdot \alpha \circ F \circ \gamma}{\overset{\lambda F^\sharp \cdot \gamma \circ F^\sharp \circ \alpha}{\leftrightarrows}} \langle Q \xrightarrow{\text{mon}} Q,\ \dot{\sqsubseteq}\rangle$$

Fixpoint abstraction



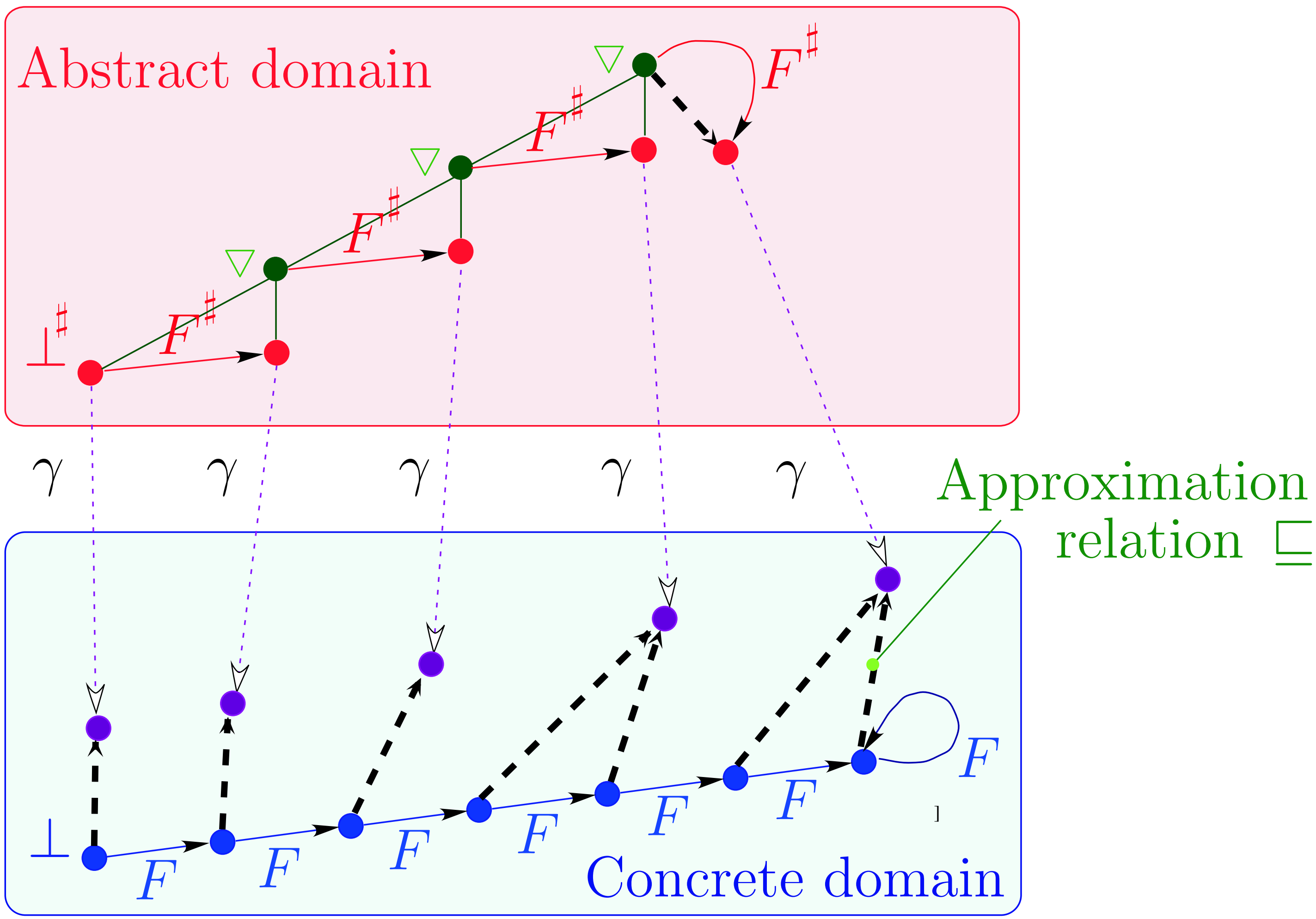
$$F \circ \gamma \sqsubseteq \gamma \circ F^\sharp \Rightarrow \mathsf{lfp}\, F \sqsubseteq \gamma(\mathsf{lfp}\, F^\sharp)$$

Abstract final state semantics

$$
\begin{aligned}
\mathcal{S}^\sharp[\![X = E;]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in \gamma(R)\})\\
\mathcal{S}^\sharp[\![\texttt{if } B\ C' \texttt{ else } C'']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{S}^\sharp[\![C'']\!](\mathcal{B}^\sharp[\![\neg B]\!]R)\\
\mathcal{B}^\sharp[\![B]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \mid B \text{ holds in } \rho\})\\
\mathcal{S}^\sharp[\![\texttt{while } B\ C']\!]R &\stackrel{\text{def}}{=} \text{let } \mathcal{W} = \textsf{lfp}^{\sqsubseteq} \lambda \mathcal{X} \cdot R \sqcup \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]\mathcal{X})\\
&\qquad \text{in } (\mathcal{B}^\sharp[\![\neg B]\!]\mathcal{W})\\
\mathcal{S}^\sharp[\![\{\}]\!]R &\stackrel{\text{def}}{=} R\\
\mathcal{S}^\sharp[\![\{C_1 \ldots C_n\}]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C_n]\!] \circ \ldots \circ \mathcal{S}^\sharp[\![C_1]\!] \quad n > 0\\
\mathcal{S}^\sharp[\![D\ C]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C]\!](\alpha(R)) \quad \text{(initial states)}
\end{aligned}
$$

The $\sqsubseteq$-least fixpoint can be computed by elimination methods or by chaotic/asynchronous iteration methods but rapid convergence may not be guaranteed in infinite or very large abstract domains.

Convergence acceleration by extrapolation [26]



26 ∇ is a *widening* operator

Abstract semantics with convergence acceleration [27]

$$\mathcal{S}^\sharp[\![X = E;]\!]R \stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in \gamma(R)\})$$

$$\mathcal{S}^\sharp[\![\texttt{if } B\ C' \texttt{ else } C'']\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{S}^\sharp[\![C'']\!](\mathcal{B}^\sharp[\![\neg B]\!]R)$$

$$\mathcal{B}^\sharp[\![B]\!]R \stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \mid B \text{ holds in } \rho\})$$

$$\mathcal{S}^\sharp[\![\texttt{while } B\ C']\!]R \stackrel{\text{def}}{=} \text{let } \mathcal{F}^\sharp = \lambda \mathcal{X} \cdot \text{let } \mathcal{Y} = R \sqcup \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]\mathcal{X})$$
$$\text{in if } \mathcal{Y} \sqsubseteq \mathcal{X} \text{ then } \mathcal{X} \text{ else } \mathcal{X} \triangledown \mathcal{Y}$$
$$\text{and } \mathcal{W} = \mathsf{lfp}^{\sqsubseteq} \mathcal{F}^\sharp \text{ in } (\mathcal{B}^\sharp[\![\neg B]\!]\mathcal{W})$$

$$\mathcal{S}^\sharp[\![\{\}]\!]R \stackrel{\text{def}}{=} R$$

$$\mathcal{S}^\sharp[\![\{C_1 \ldots C_n\}]\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C_n]\!] \circ \ldots \circ \mathcal{S}^\sharp[\![C_1]\!] \quad n > 0$$

$$\mathcal{S}^\sharp[\![D\ C]\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C]\!](\alpha(R)) \quad \text{(initial states)}$$

---

[27] Note: $\mathcal{F}^\sharp$ not monotonic!

Applications of Abstract Interpretation

Applications of Abstract Interpretation

Abstract interpretation formalizes sound approximations as found everywhere in computer science:

- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92], [TCS 277(1–2) 2002]
- **Program Transformation** [POPL '02]
- **Typing & Type Inference** [POPL '97]
- **(Abstract) Model Checking** [POPL '00]

Applications of Abstract Interpretation (Cont'd)

– **Bisimulations** [RT-ESOP '04]

– **Software Watermarking** [POPL '04]

– **Code obfuscation** [DPG-ICALP '05]

– **Static Program Analysis** [POPL '77], [POPL '78], [POPL '79] including

  - **Dataflow Analysis** [POPL '79], [POPL '00],

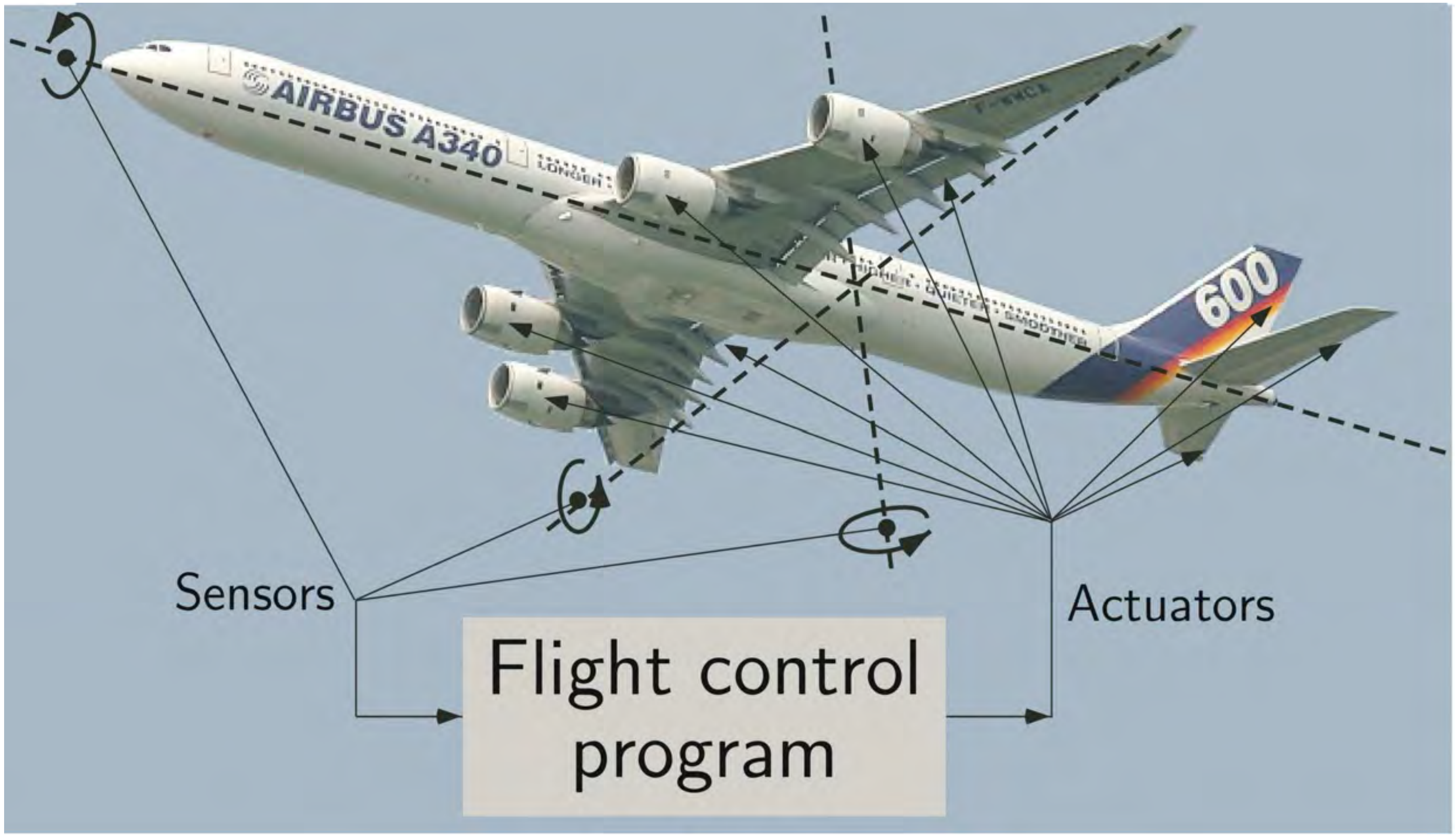
  - **Set-based Analysis** [FPCA '95],

  - **Predicate Abstraction** [Manna's festschrift '03], ...

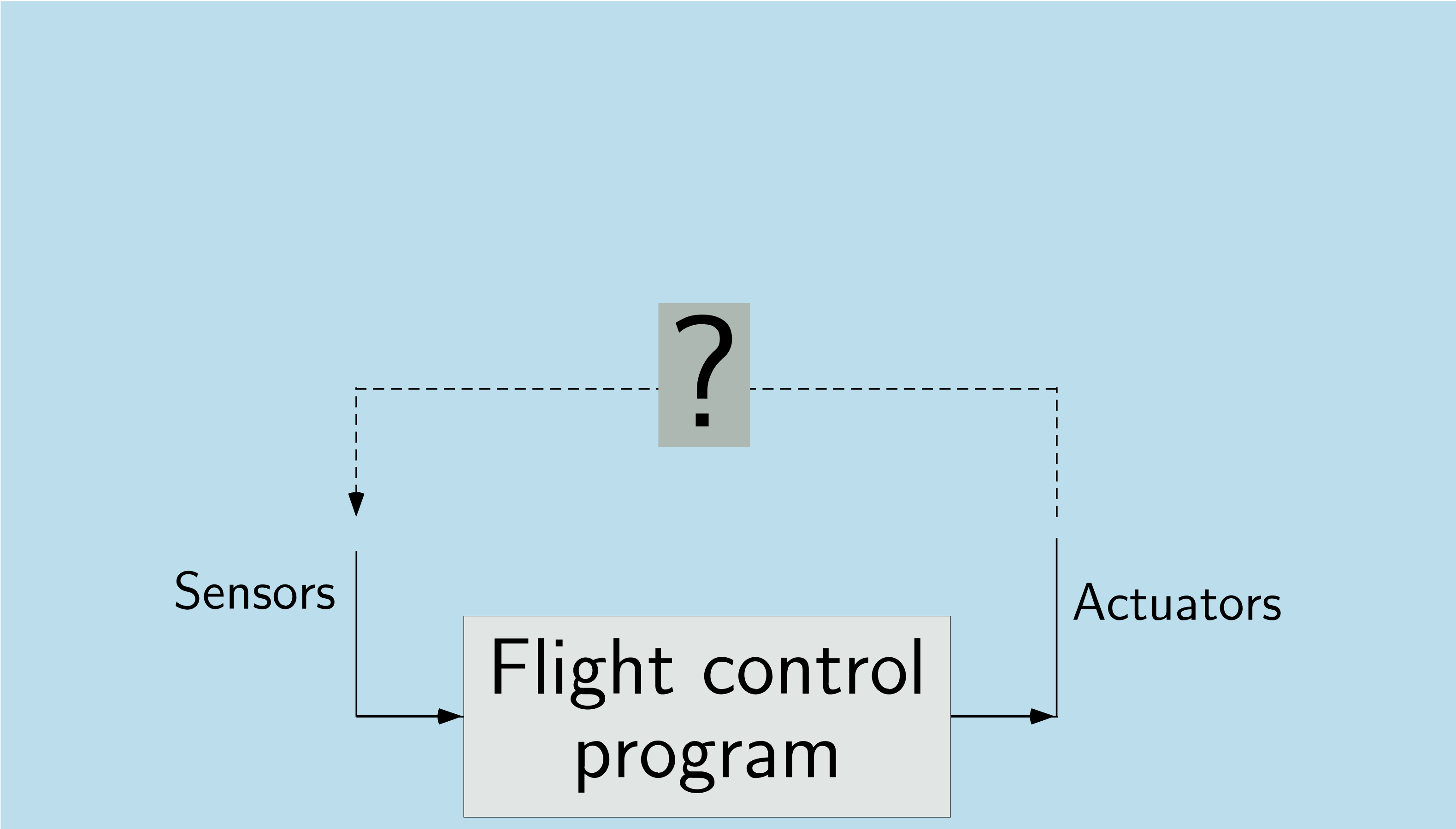
  - **WCET** [EMSOFT '01], ...

Project while visiting MIT

Computer controlled systems

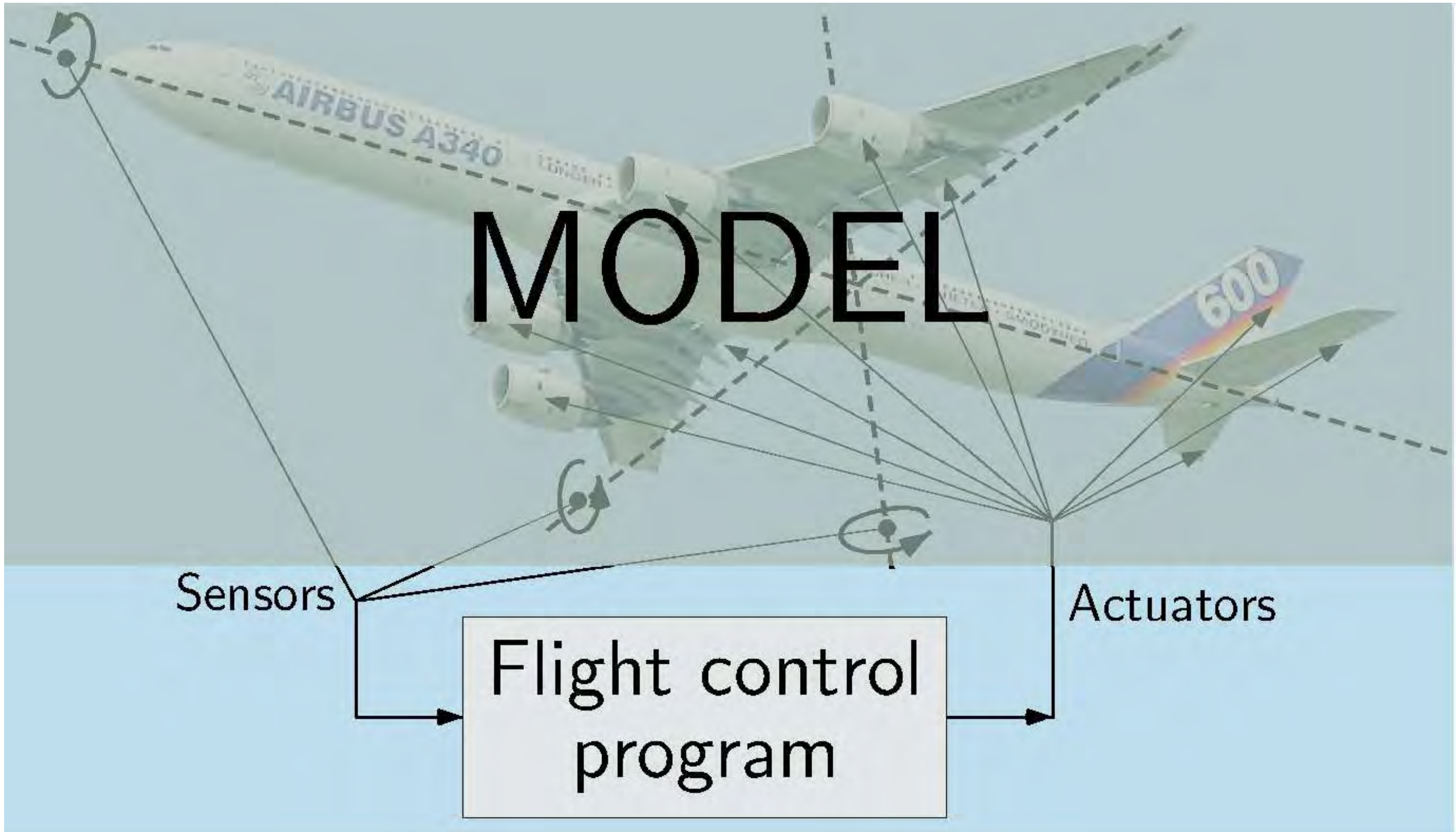


Software analysis & verification



Abstractions: program $\longrightarrow$ precise, system $\longrightarrow$ coarse

System analysis & verification



Abstractions: program → precise, system → precise

Conclusion

Grand challenge

Software verification

– is the grand challenge for computer scientists and engineers in the next 15 years

– will not be convincing without global system verification

THE END

My MIT web site is www.mit.edu/~cousot, where these slides are available
My ENS web site is www.di.ens.fr/~cousot

For more technical details, see the MIT course 16.399 on *Abstract interpretation*
web.mit.edu/16.399/

References

[1] `www.astree.ens.fr` [3, 4, 5, 6, 7, 8, 9, 10]

[2] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 March 1978.

[3] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pp. 85–108. Springer, 2002.

[4] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. *PLDI'03*, San Diego, pp. 196–207, ACM Press, 2003.

[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY, USA.

[PACJM '79] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. Pacific Journal of Mathematics 82(1):43–57 (1979).

[POPL '78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY, U.S.A.

[POPL '79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY, U.S.A.

[POPL '92] P. Cousot and R. Cousot. Inductive Definitions, Semantics and Abstract Interpretation. In Conference Record of the 19$^{\text{th}}$ ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.

[FPCA '95] P. Cousot and R. Cousot. Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation. In *SIGPLAN/SIGARCH/WG2.8 7$^{\text{th}}$ Conference on Functional Programming and Computer Architecture, FPCA'95*. La Jolla, California, U.S.A., pages 170–181. ACM Press, New York, U.S.A., 25-28 June 1995.

[POPL '97] P. Cousot. Types as Abstract Interpretations. In Conference Record of the 24$^{\text{th}}$ ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, pages 316–331, Paris, France, 1997. ACM Press, New York, U.S.A.

[POPL '00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.

[POPL '02] P. Cousot and R. Cousot. Systematic Design of Program Transformation Frameworks by Abstract Interpretation. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, January 2002. ACM Press, New York, NY.

[TCS 277(1–2) 2002] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science* 277(1–2):47–103, 2002.

[TCS 290(1) 2002] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoret. Comput. Sci.*, 290:531–544, 2003.

[Manna's festschrift '03] P. Cousot. Verification by Abstract Interpretation. *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, N. Dershowitz (Ed.), Taormina, Italy, June 29 – July 4, 2003. Lecture Notes in Computer Science, vol. 2772, pp. 243–268. © Springer-Verlag, Berlin, Germany, 2003.

[5] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyser. *ESOP 2005*, Edinburgh, LNCS 3444, pp. 21–30, Springer, 2005.

[6] J. Feret. Static analysis of digital filters. *ESOP'04*, Barcelona, LNCS 2986, pp. 33—-48, Springer, 2004.

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

[8] Laurent Mauborgne & Xavier Rival. Trace Partitioning in Abstract Interpretation Based Static Analyzers. *ESOP'05*, Edinburgh, LNCS 3444, pp. 5–20, Springer, 2005.

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

[10] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. *ESOP'04*, Barcelona, LNCS 2986, pp. 3—17, Springer, 2004.

[POPL '04] P. Cousot and R. Cousot. An Abstract Interpretation-Based Framework for Software Watermarking. In *Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 173–185, Venice, Italy, January 14-16, 2004. ACM Press, New York, NY.

[DPG-ICALP '05] M. Dalla Preda and R. Giacobazzi. Semantic-based Code Obfuscation by Abstract Interpretation. In Proc. 32nd Int. Colloquium on Automata, Languages and Programming (ICALP'05 – Track B). LNCS, 2005 Springer-Verlag. July 11-15, 2005, Lisboa, Portugal. To appear.

[EMSOFT '01] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. *ESOP (2001)*, LNCS 2211, 469–485.

[RT-ESOP '04] F. Ranzato and F. Tapparo. Strong Preservation as Completeness in Abstract Interpretation. ESOP 2004, Barcelona, Spain, March 29 - April 2, 2004, D.A. Schmidt (Ed), LNCS 2986, Springer, 2004, pp. 18–32.