Software Verification by Abstract Interpretation: Current Trends and Perspectives

Patrick COUSOT

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

Jerome C. Hunsaker Visiting Professor
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
cousot@mit.edu
www.mit.edu/~cousot

Meeting with Ford representatives, MIT, 15 March 2005

All Computer Scientists Have Experienced Bugs

It is preferable to verify that safety-critical programs do not go wrong before running them.

Static Analysis by Abstract Interpretation

Static analysis: analyse the program at compile-time to verify a program runtime property (e.g. the absence of some categories of bugs)

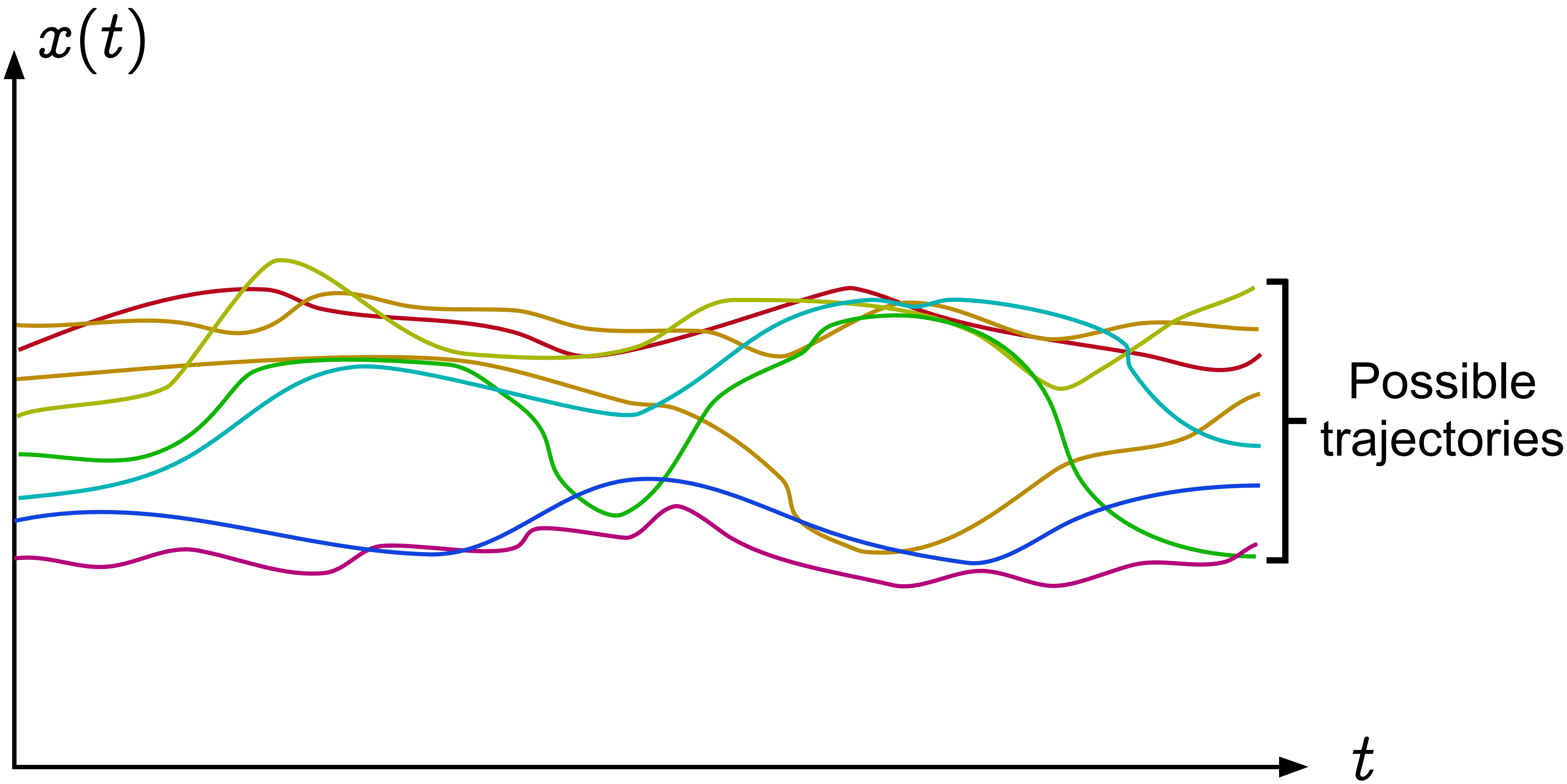
Undecidability $\longrightarrow$

Abstract interpretation: effectively compute an abstraction/ sound approximation of the program semantics,

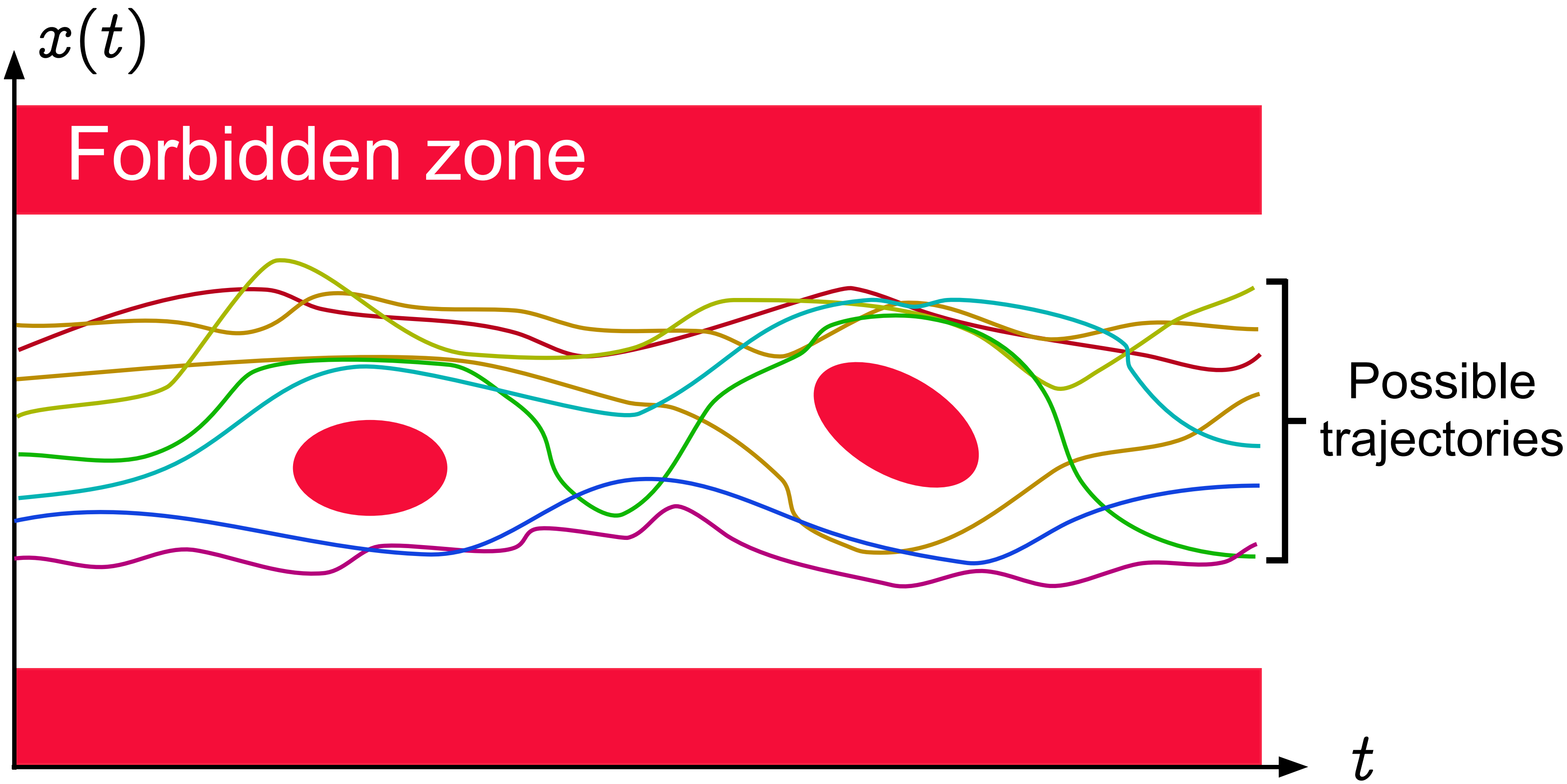
- which is precise enough to imply the desired property, and
- coarse enough to be efficiently computable.

Abstract Interpretation, Informally

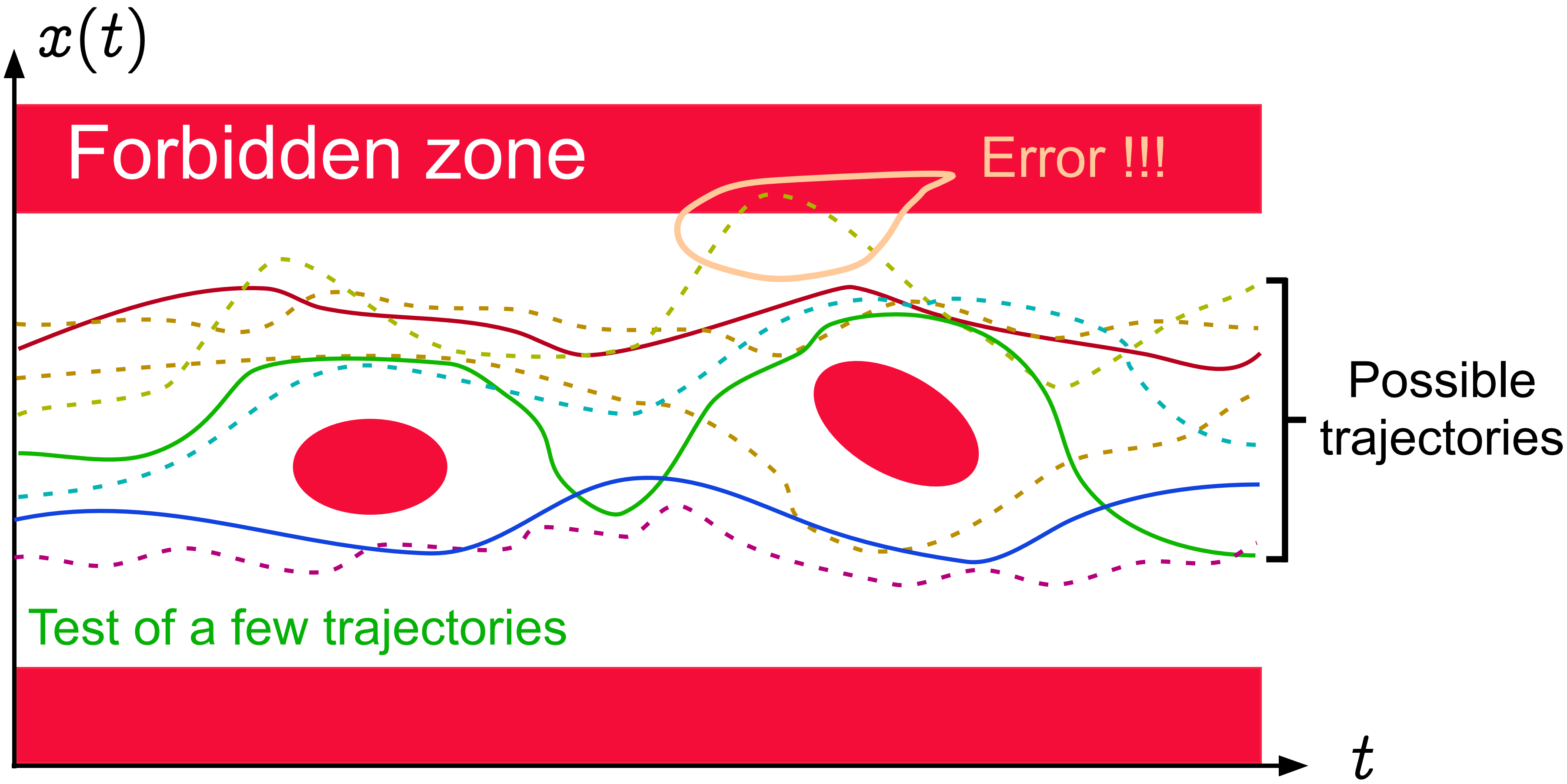
Operational Semantics



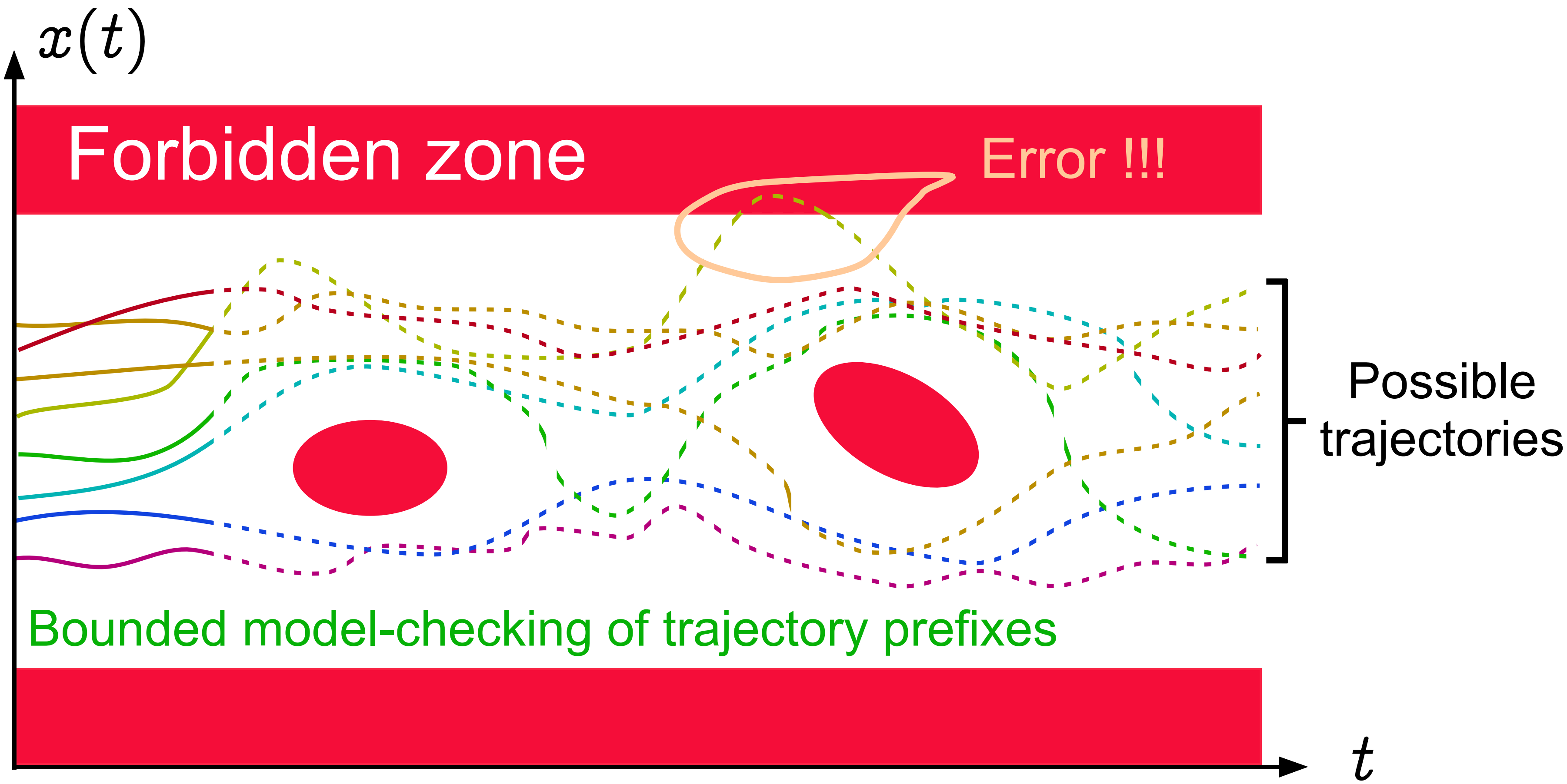
Safety property



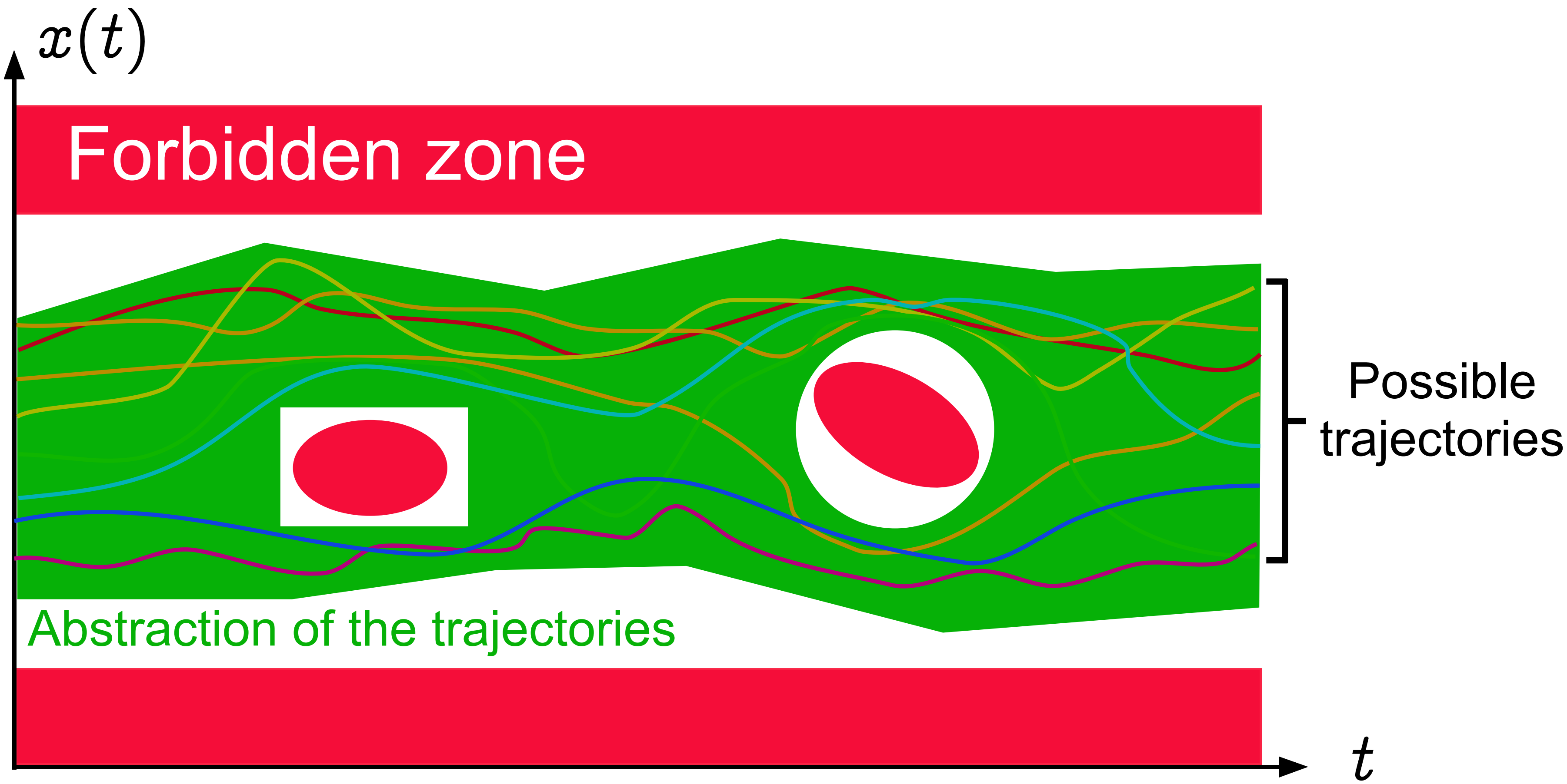
Test/Debugging is Unsafe



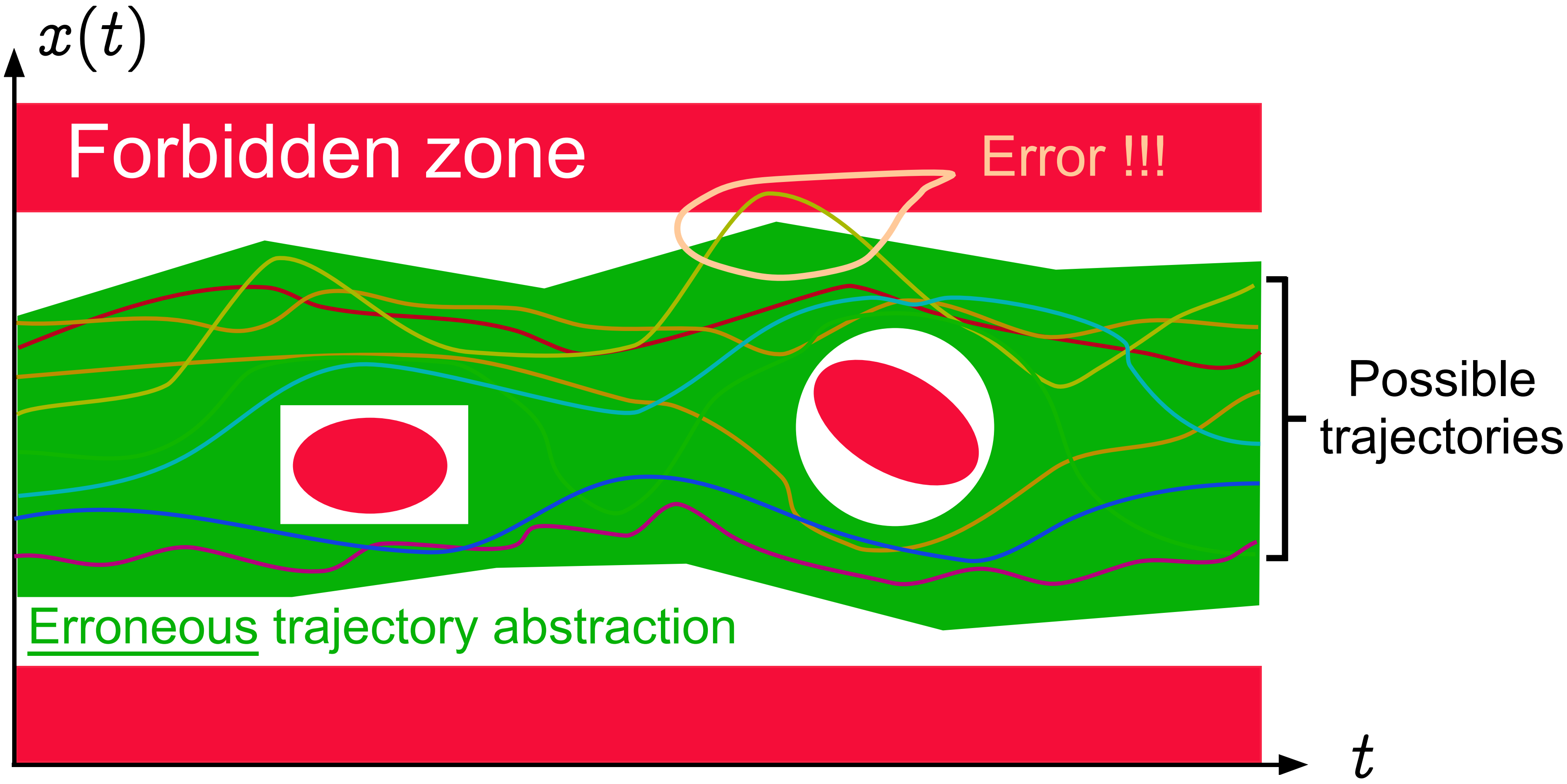
Bounded Model Checking is Unsafe



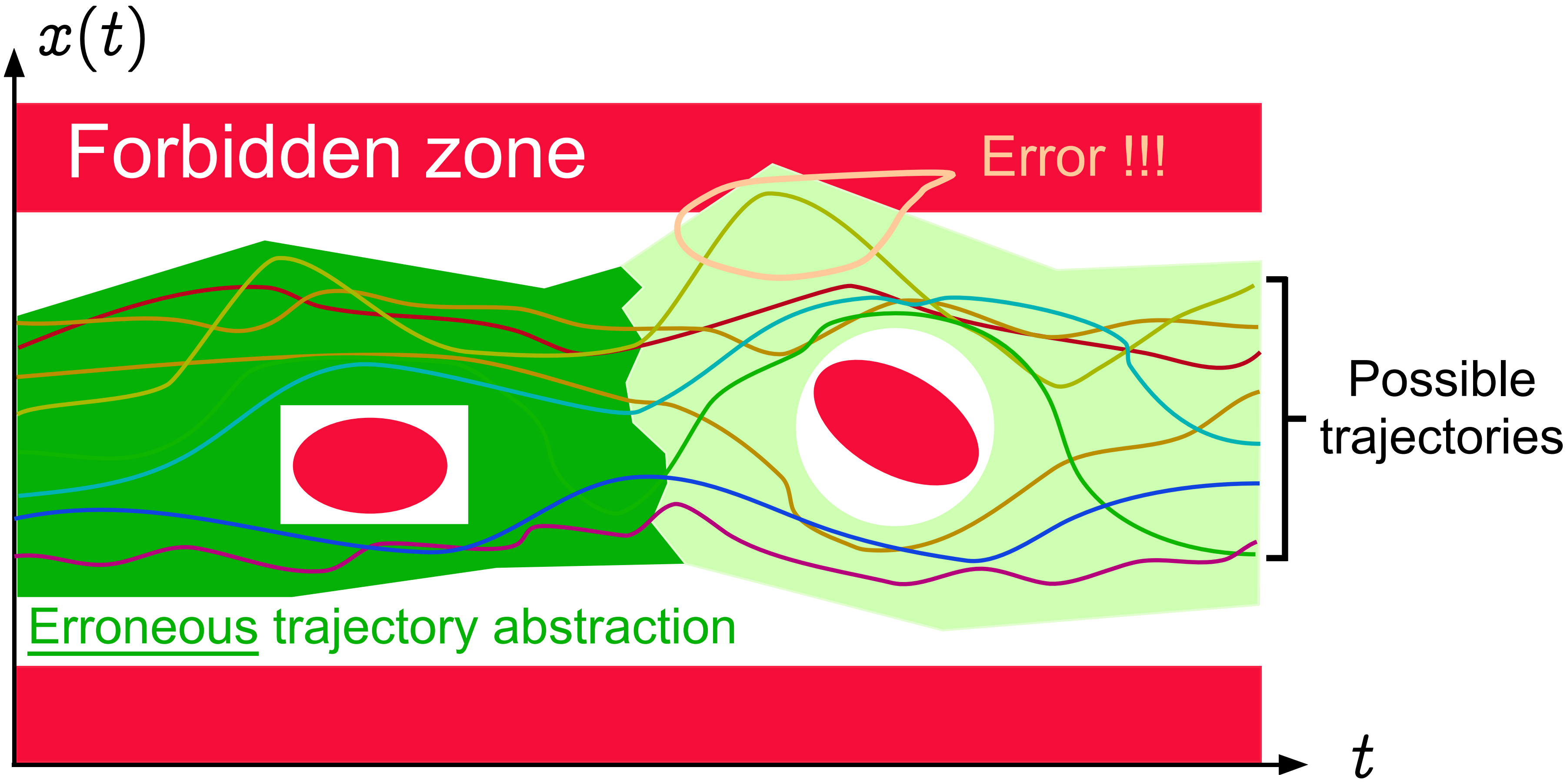
Abstract Interpretation



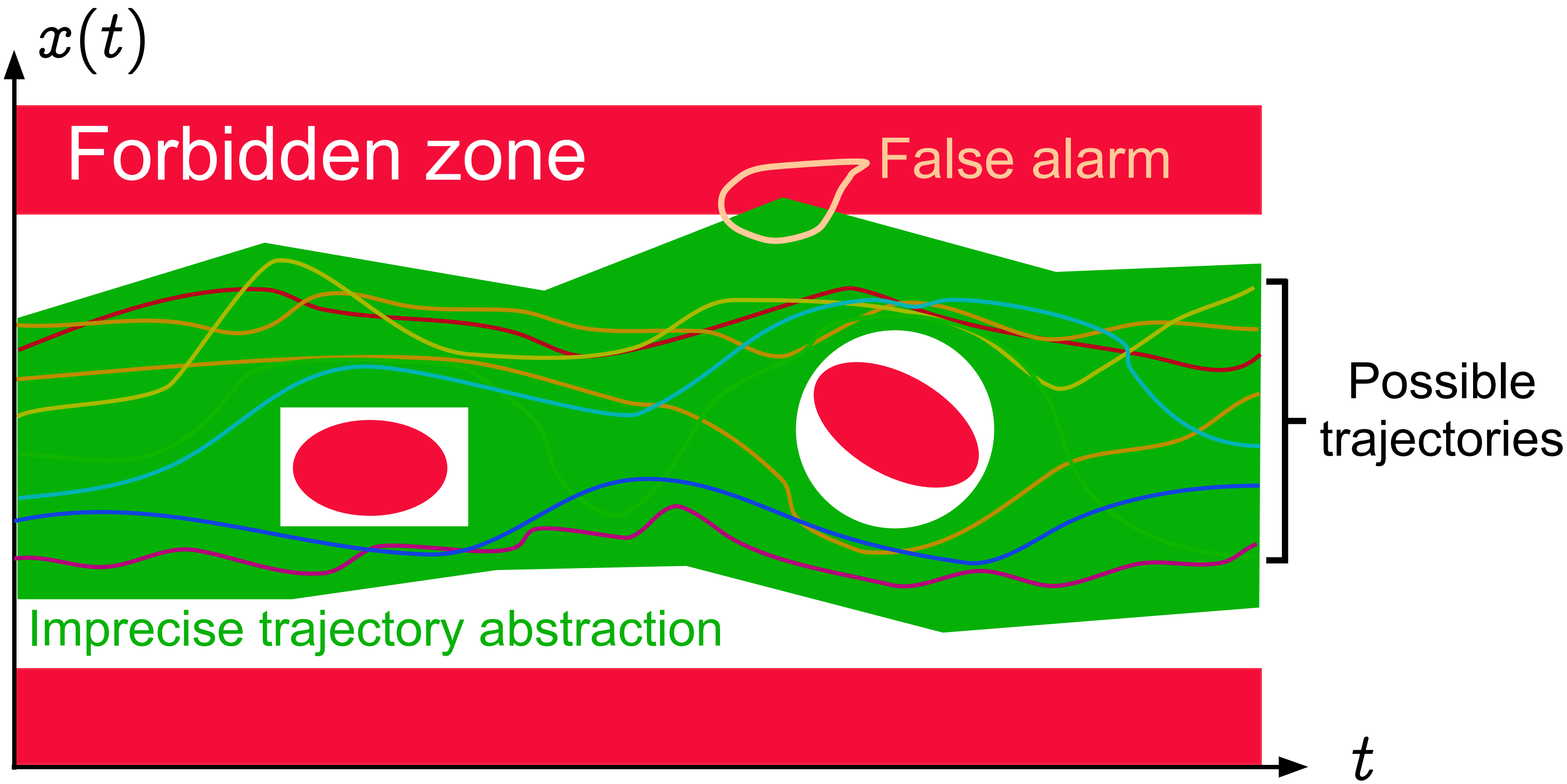
Soundness: Erroneous Abstraction — I



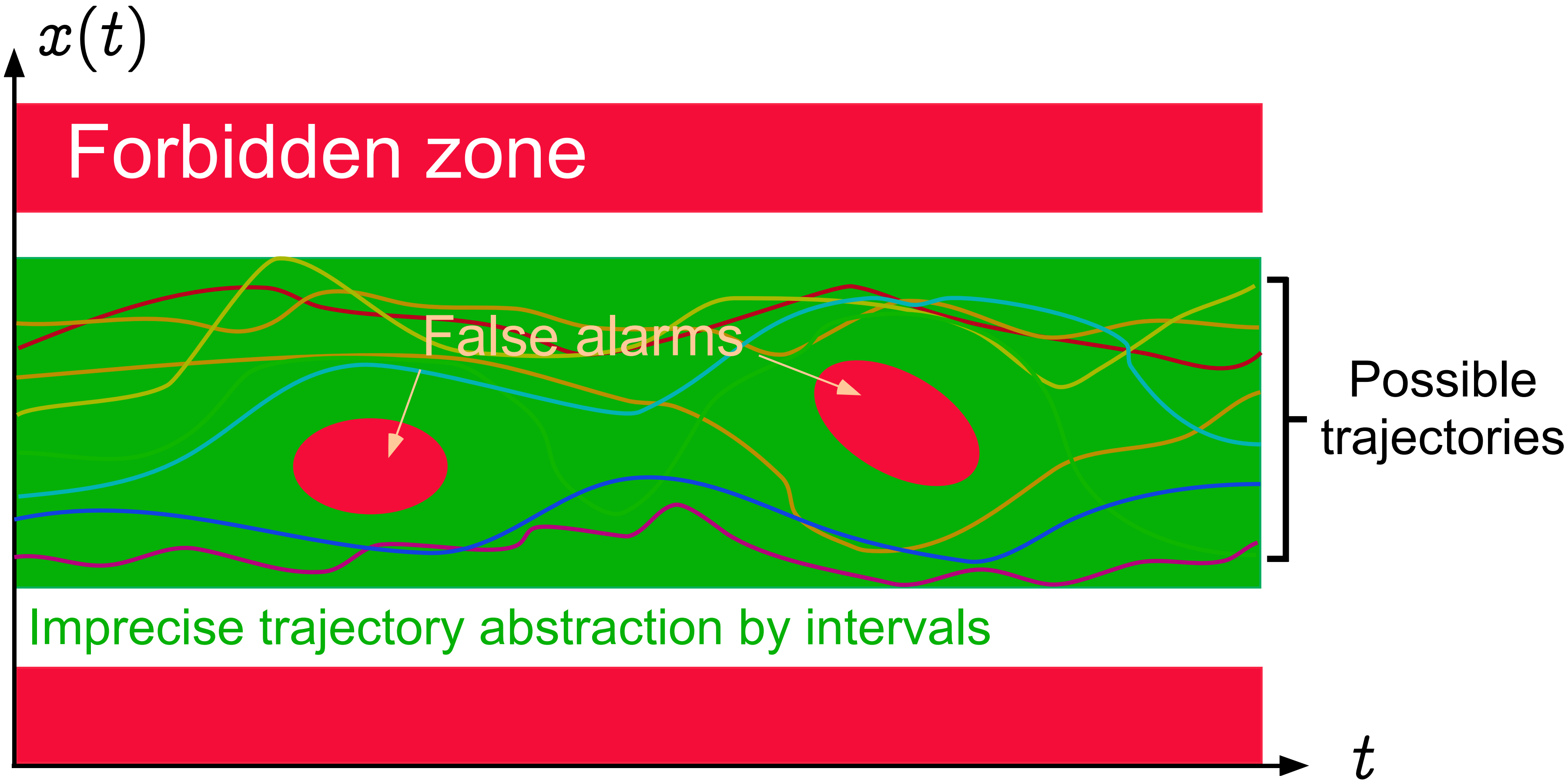
Soundness: Erroneous Abstraction — II



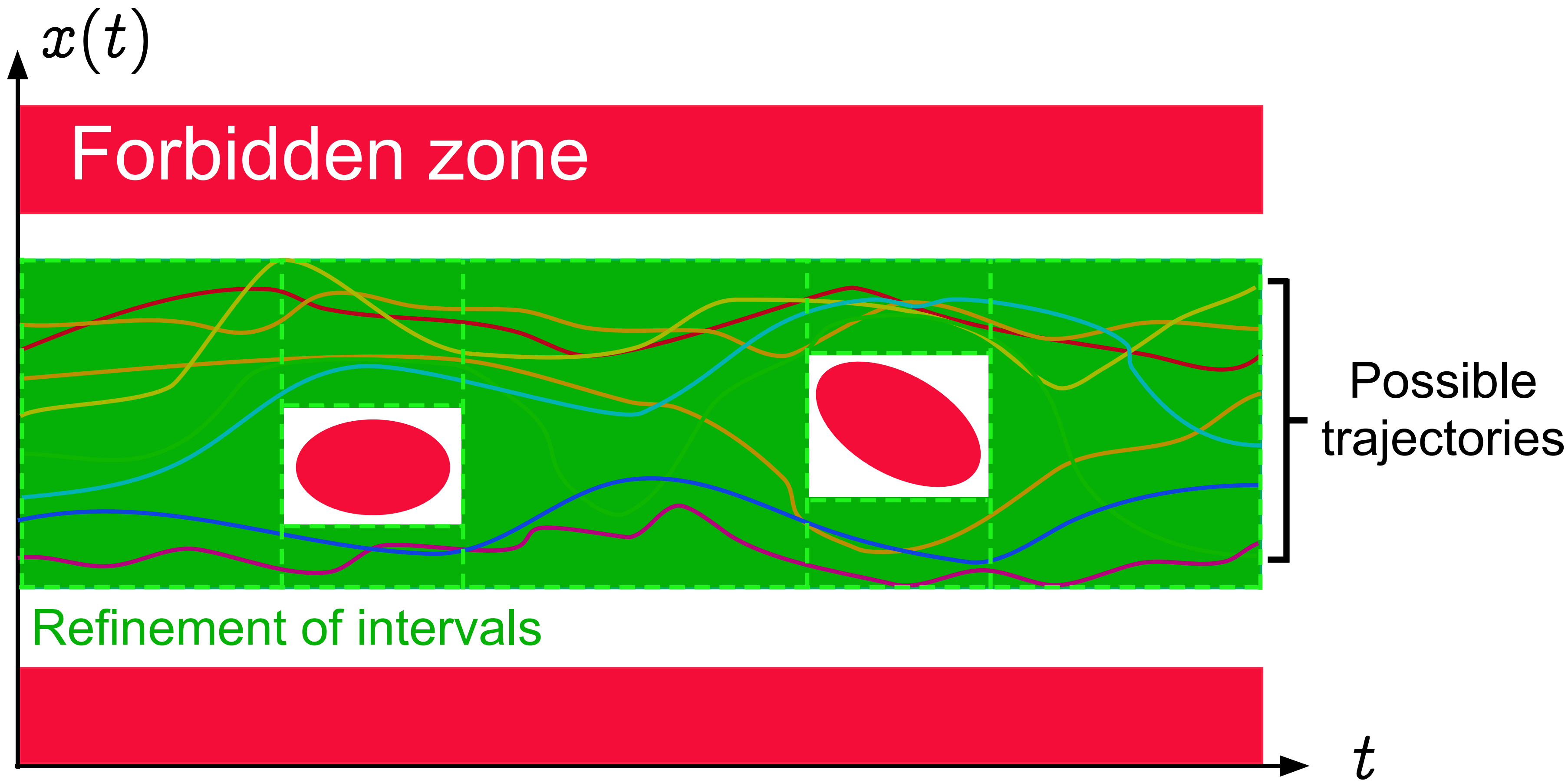
Imprecision $\Rightarrow$ False Alarms



Interval Abstraction ⇒ False Alarms



Refinement by Partitionning



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

Reference

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

[2] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. PLDI'03, San Diego, June 7–14, ACM Press, 2003.

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

www.astree.ens.fr

Implicit Specification: Absence of Runtime

- No violation of the norm of C (e.g. array index out of bounds)
- No implementation-specific undefined behaviors (e.g. maximum short integer is 32767)
- No violation of the programming guidelines (e.g. static variables cannot be assumed to be initialized to 0)
- No violation of the programmer assertions (must all be statically verified).

Example application

– Primary flight control software of the Airbus A340/A380 fly-by-wire system

– C program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)

– A340 family: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays

– A380: $\times$ 3 $\Rightarrow$ No false alarm!

Examples

Floating-Point Computations

– Code Sample:

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

```
/* 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
```

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

Filter Example [3]

```
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; }
}
```

Reference

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

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
```

Reference

[4] J. Feret. The Arithmetic-Geometric Progression Abstract Domain. In *VMCAI'05*, Paris, January 17—19, 2005, LNCS, Springer.

Conclusion

The Future & Grand Challenges

Forthcoming (1 year):

– More gereral memory model (`union`)

Future (5 years):

– Asynchronous concurrency (for less critical software)

– Functional properties (reactivity)

– Industrialization

Grand challenge:

– Verification from specifications to machine code (verifying compiler)

– Verification of systems (quasi-synchrony, distribution)

THE END, THANK YOU

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