Automatic Large-Scale Software Verification by Abstract Interpretation

Patrick Cousot

cousot@di.ens.fr
di.ens.fr/~cousot

pcousot@cims.nyu.edu
cims.nyu.edu/~pcousot

Tsinghua Software Day
Beijing, China — March 15, 2012

Abstract

Abstract interpretation is a theory of abstraction and constructive approximation of the mathematical structures used in the formal description of programming languages and the inference or verification of undecidable program properties.

Developed in the late seventies with Radhia Cousot, it has since then been considerably applied to many aspects of programming, from syntax, to semantics, and proof methods where abstractions are sound and complete but incomputable to fully automatic, sound but incomplete approximate abstractions to solve undecidable problems such as static analysis of infinite state software systems, contract inference, type inference, termination inference, model-checking, abstraction refinement, program transformation (including watermarking), combination of decision procedures, security, malware detection, etc.

This last decade, abstract interpretation has been very successful in program verification for mission- and safety-critical systems. An example is Astrée (www.astree.ens.fr) which is a static analyzer to verify the absence of runtime errors in structured, very large C programs with complex memory usages, and involving complex boolean as well as floating-point computations (which are handled precisely and safely by taking all possible rounding errors into account), but without recursion or dynamic memory allocation. Astrée targets embedded applications as found in earth transportation, nuclear energy, medical instrumentation, aeronautics and space flight, in particular synchronous control/command such as electric flight control or more recently asynchronous systems as found in the automotive industry.

Astrée is industrialized by AbsInt (www.absint.com/astree).

Content

- Motivation
- An informal introduction to abstract interpretation
- A touch of theory of abstract interpretation
- A short overview of a few applications and on-going work on software verification

---

For a rather complete basic introduction to abstract interpretation and applications to cyber-physical systems, see:

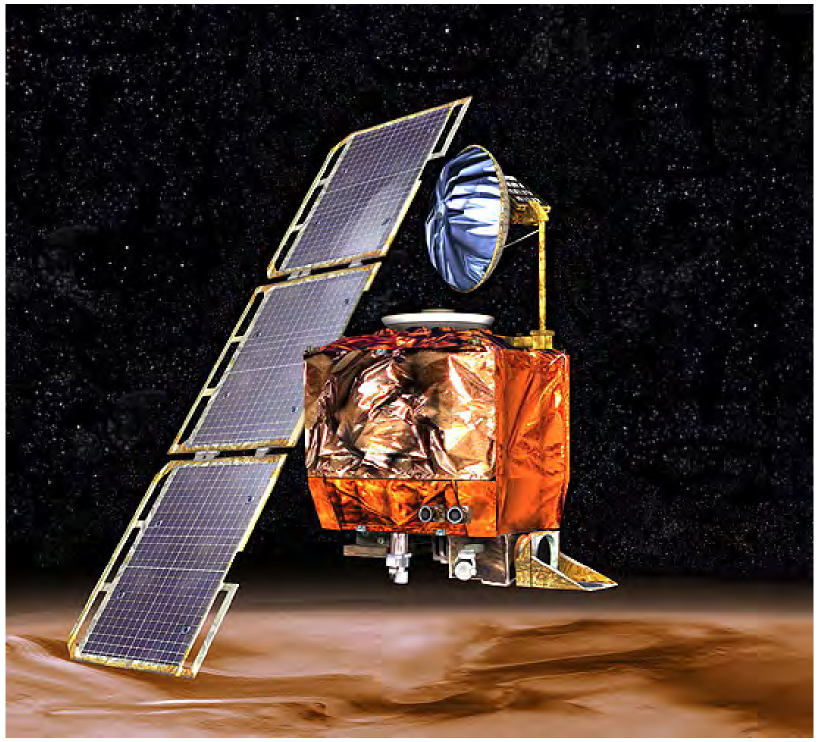
Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, & Xavier Rival. Static Analysis and Verification of Aerospace Software by Abstract Interpretation. In *AIAA Infotech@@Aerospace 2010*, Atlanta, Georgia. American Institute of Aeronautics and Astronautics, 20—22 April 2010. © AIAA.

Motivation

All computer scientists have experienced bugs

Ariane 5.01 failure
(overflow)

Patriot failure
(float rounding)



Mars orbiter loss
(unit error)

- Checking the presence of bugs is great
- Proving their absence is even better!

Abstract interpretation

Patrick Cousot & Radhia Cousot. Static Determination of Dynamic Properties of Programs. In B. Robinet, editor, *Proceedings of the second international symposium on Programming,* Paris, France, pages 106—130, April 13-15 1976, Dunod, Paris.

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Patrick Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes. *Thèse És Sciences Mathématiques*, Université Joseph Fourier, Grenoble, France, 21 March 1978

Patrick Cousot. Semantic foundations of program analysis. In S.S. Muchnick & N.D. Jones, editors, *Program Flow Analysis: Theory and Applications,* Ch. 10, pages 303—342, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

Abstract interpretation

- *Started in the 70's* and widely applied since then
- Based on the idea that undecidability and complexity of automated program analysis can be fought by *sound approximations* or *complete abstractions*
- Wide-spectrum theory so applications range from *static analysis* to *verification* to biology
- ***Does scale up!***

Fighting undecidability and complexity in program verification

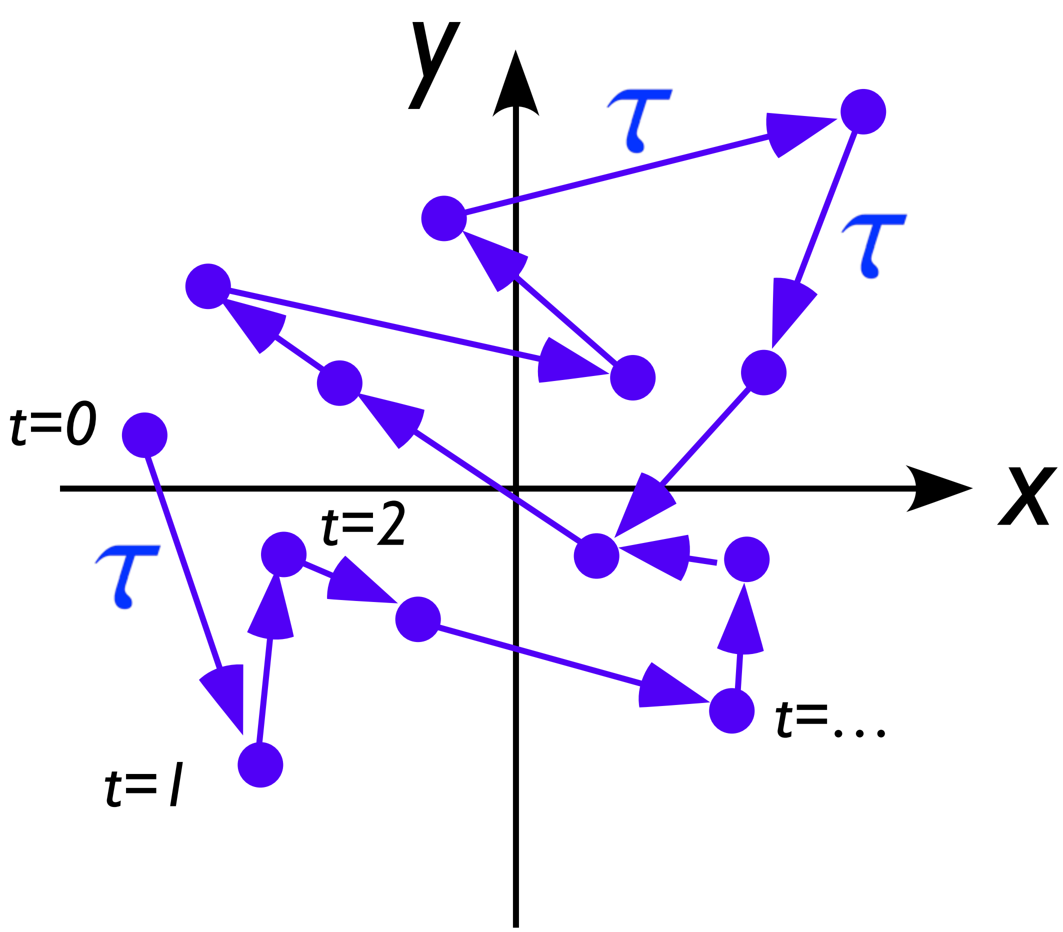
- Any *automatic* program verification method will definitely fail on infinitely many programs (Gödel)
- Solutions:
  - Ask for human help (theorem-prover/proof assistant based *deductive methods*)
  - Consider (small enough) finite systems (*model-checking*)
  - Do sound approximations or complete abstractions (*abstract interpretation*)

An informal introduction to abstract interpretation

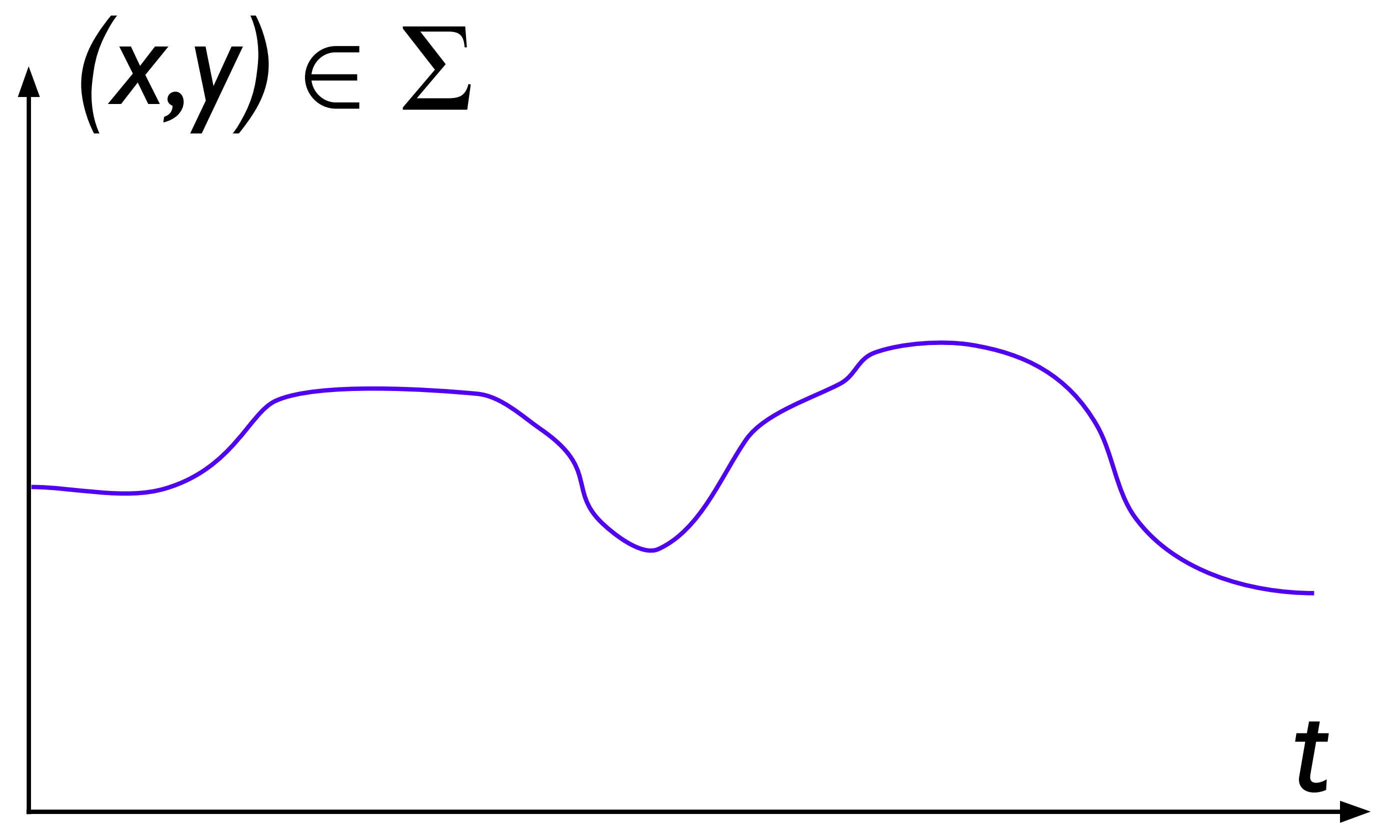
P. Cousot & R. Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In *Logics and Languages for Reliability and Security*, J. Esparza, O. Grumberg, & M. Broy (Eds), NATO Science Series III: Computer and Systems Sciences, © IOS Press, 2010, Pages 1—29.

1) Define the programming language semantics

*Formalize the concrete **execution** of programs (e.g. transition system)*



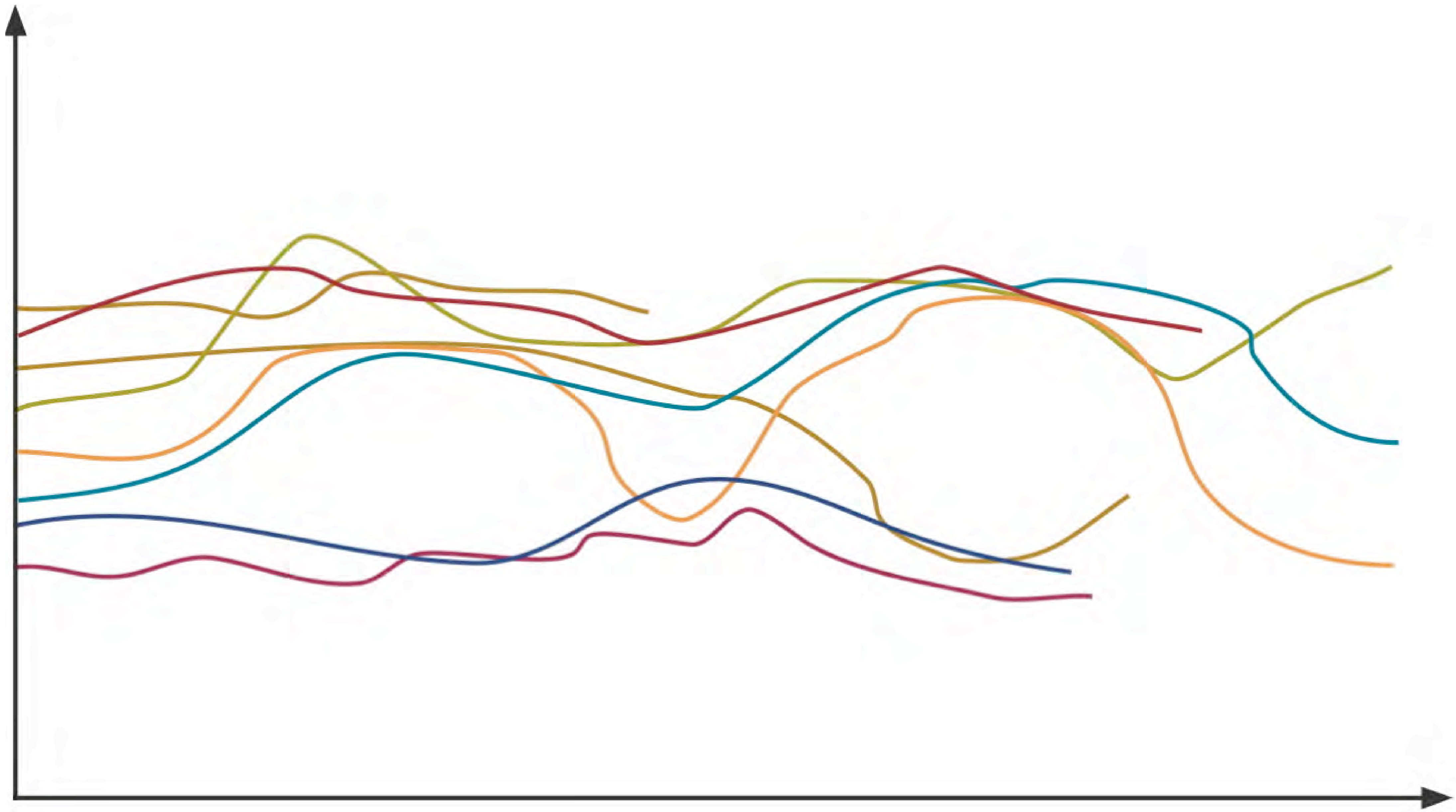
Trajectory
in state space Σ



Space/time trajectory

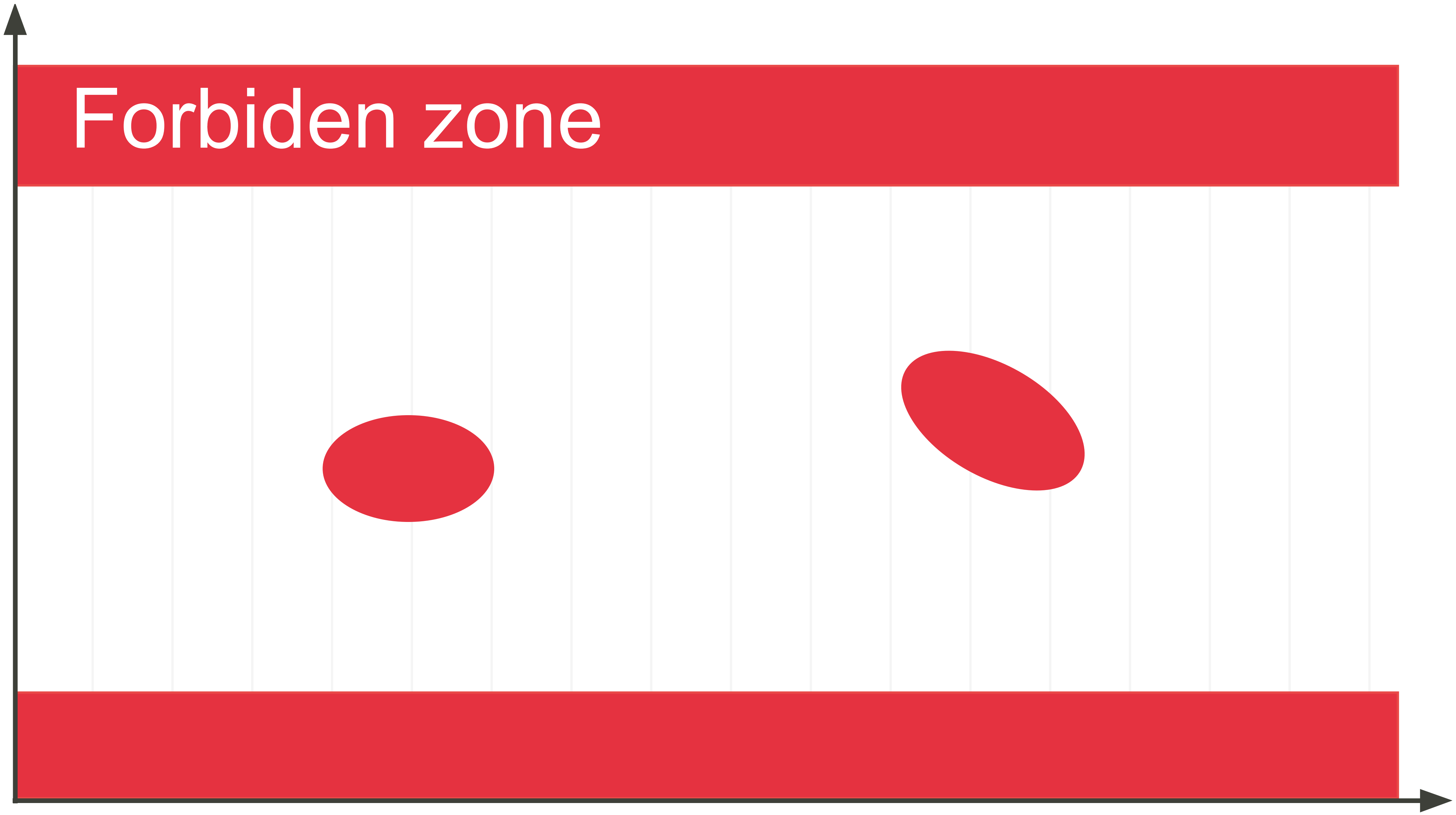
II) Define the program properties of interest

*Formalize what you are interested to **know** about program behaviors*



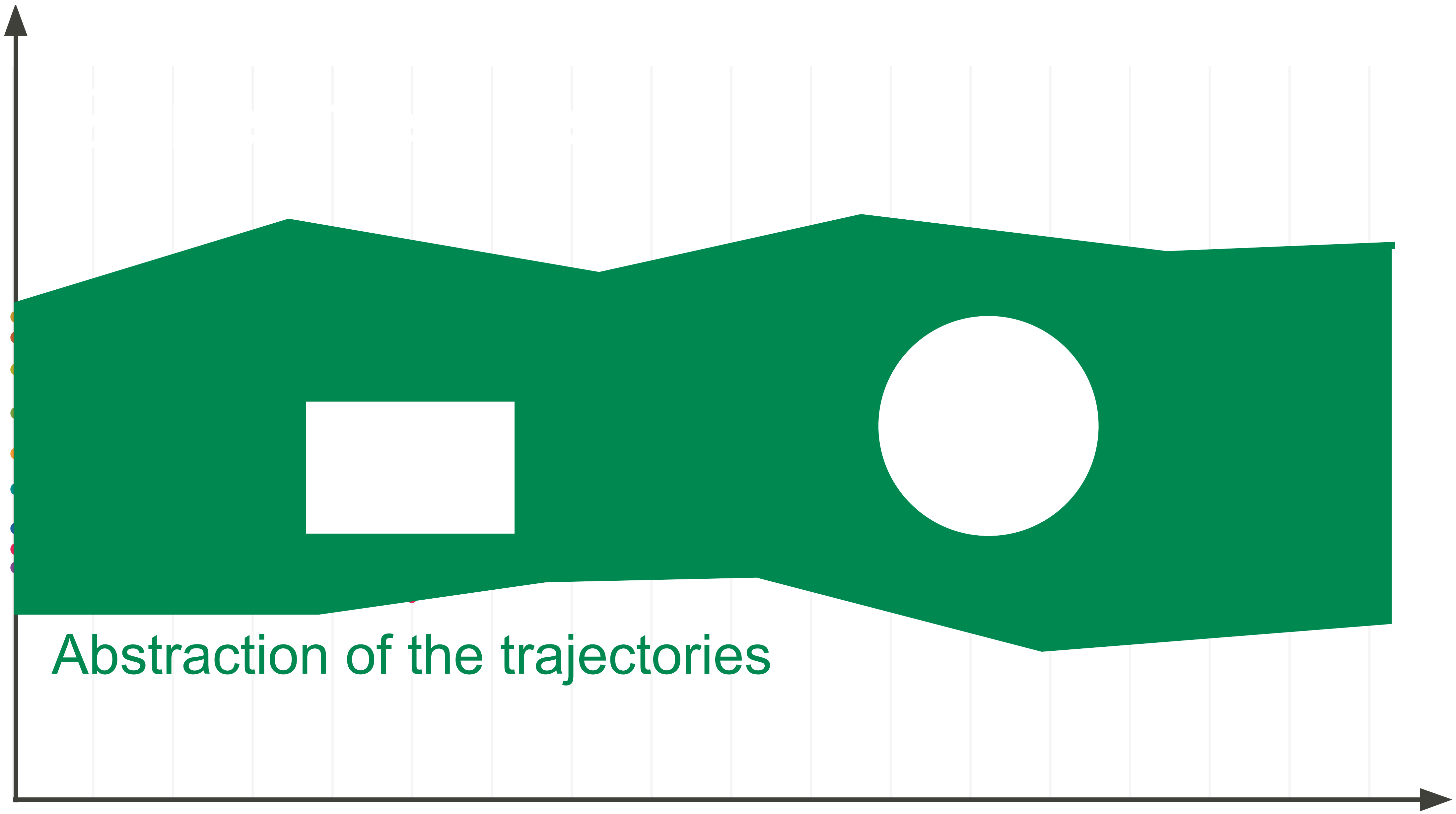
III) Define which specification must be checked

Formalize what you are interested to ***prove*** *about program behaviors*



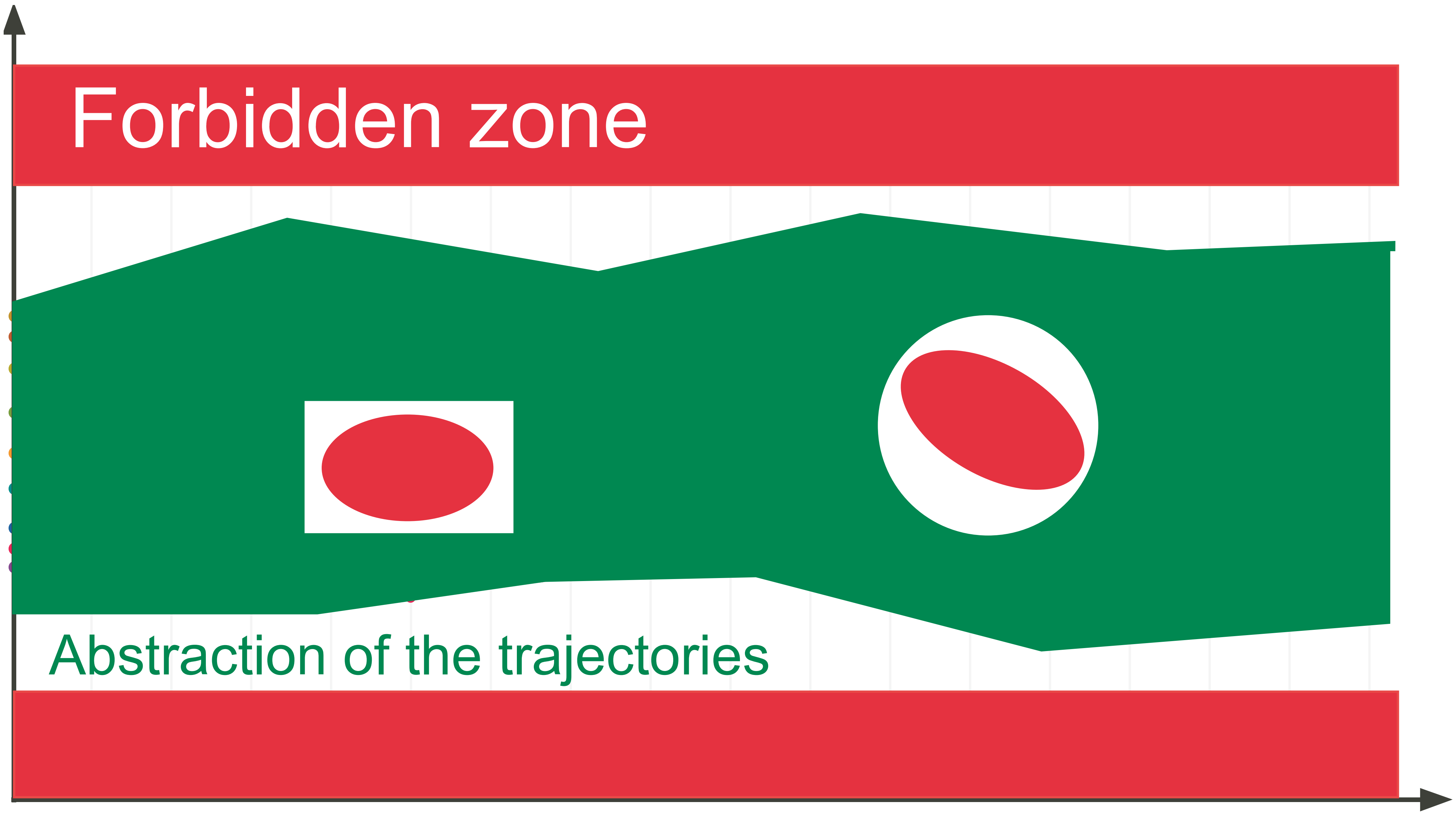
IV) Choose the appropriate abstraction

Abstract *away all information on program behaviors irrelevant to the proof*



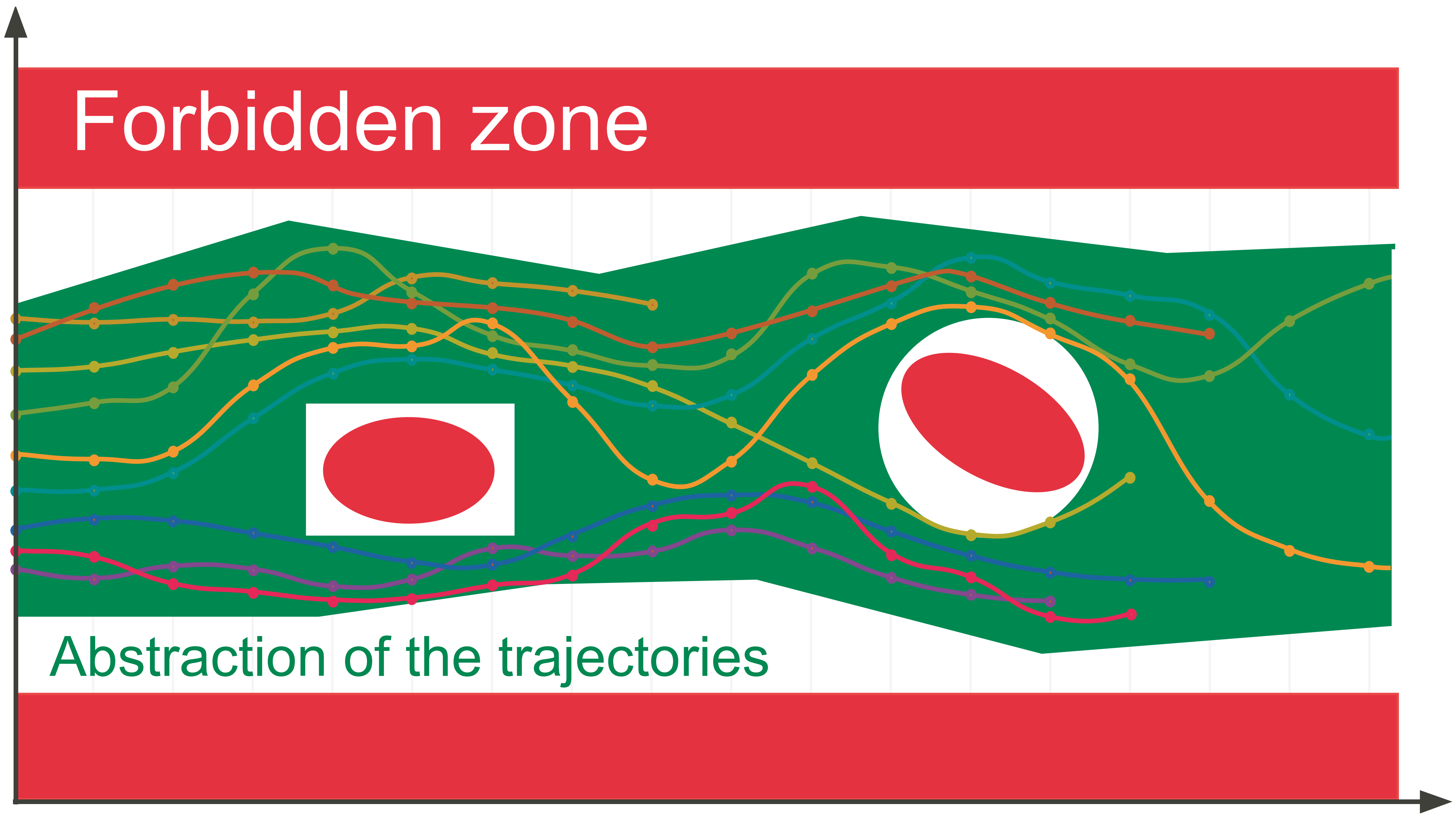
V) Mechanically verify in the abstract

The proof is fully ***automatic***



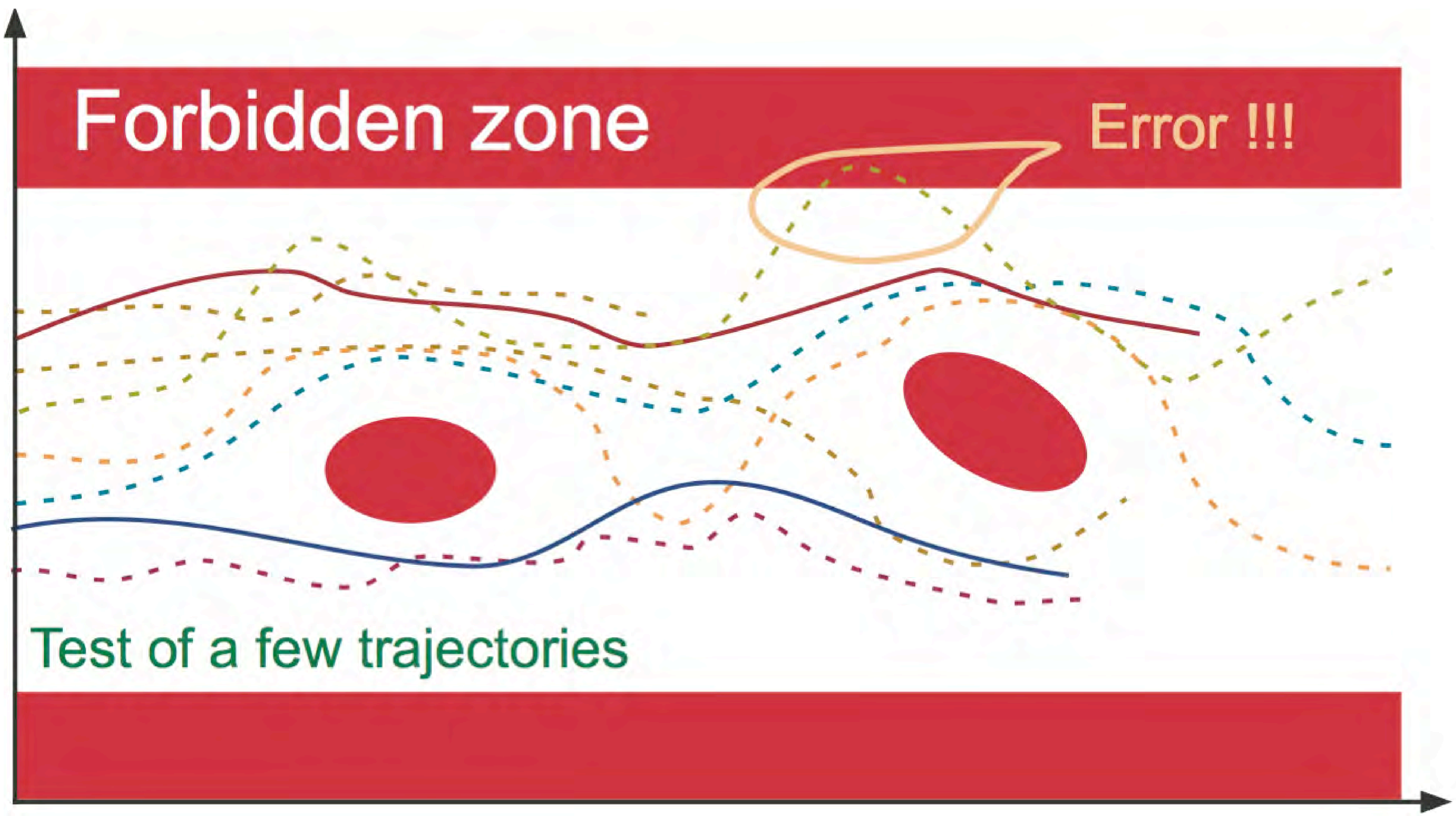
Soundness of the abstract verification

Never forget any possible case so the ***abstract proof is correct in the concrete***



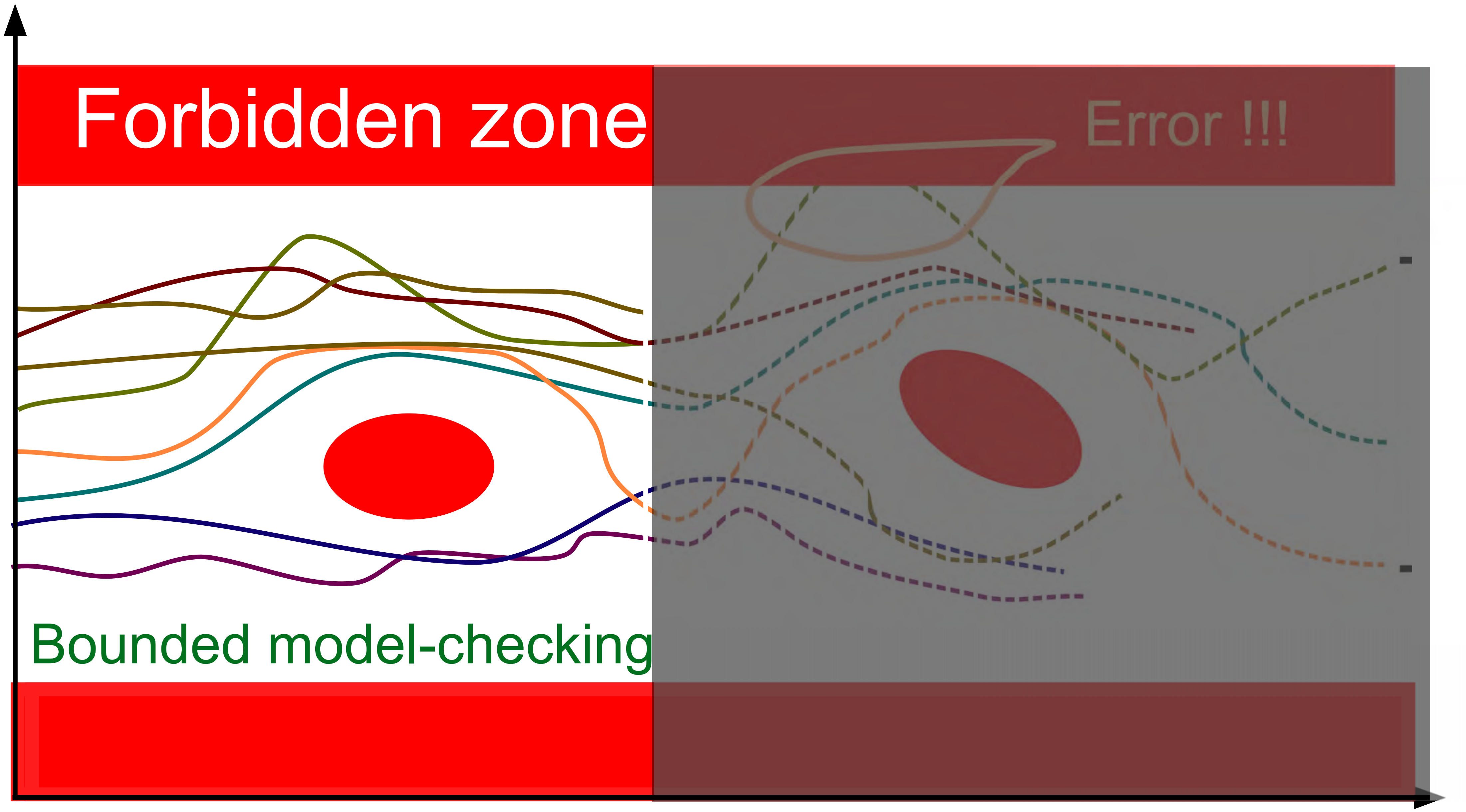
Unsound validation: testing

Try a few cases



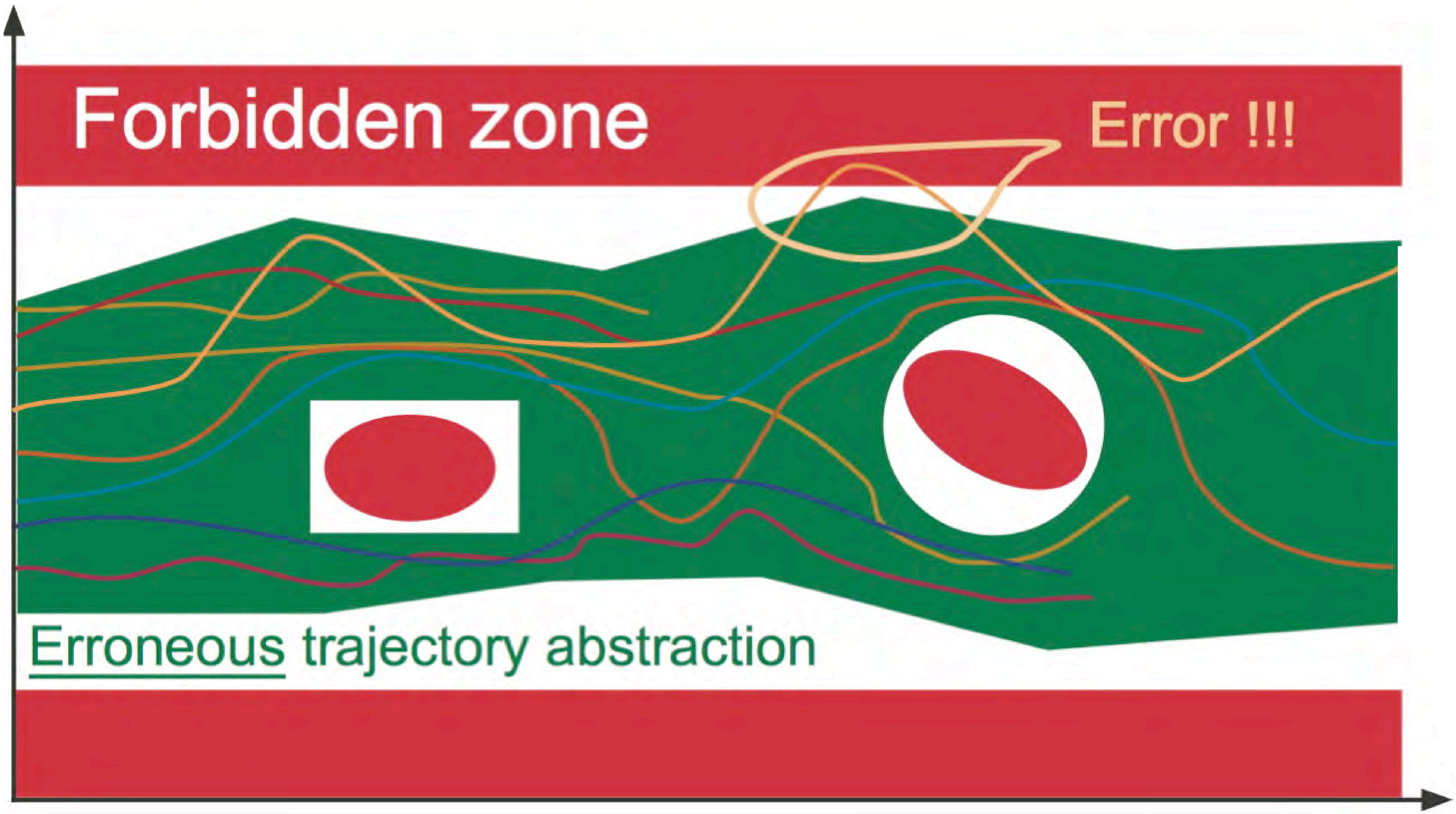
Unsound validation: bounded model-checking

Simulate the beginning of all executions



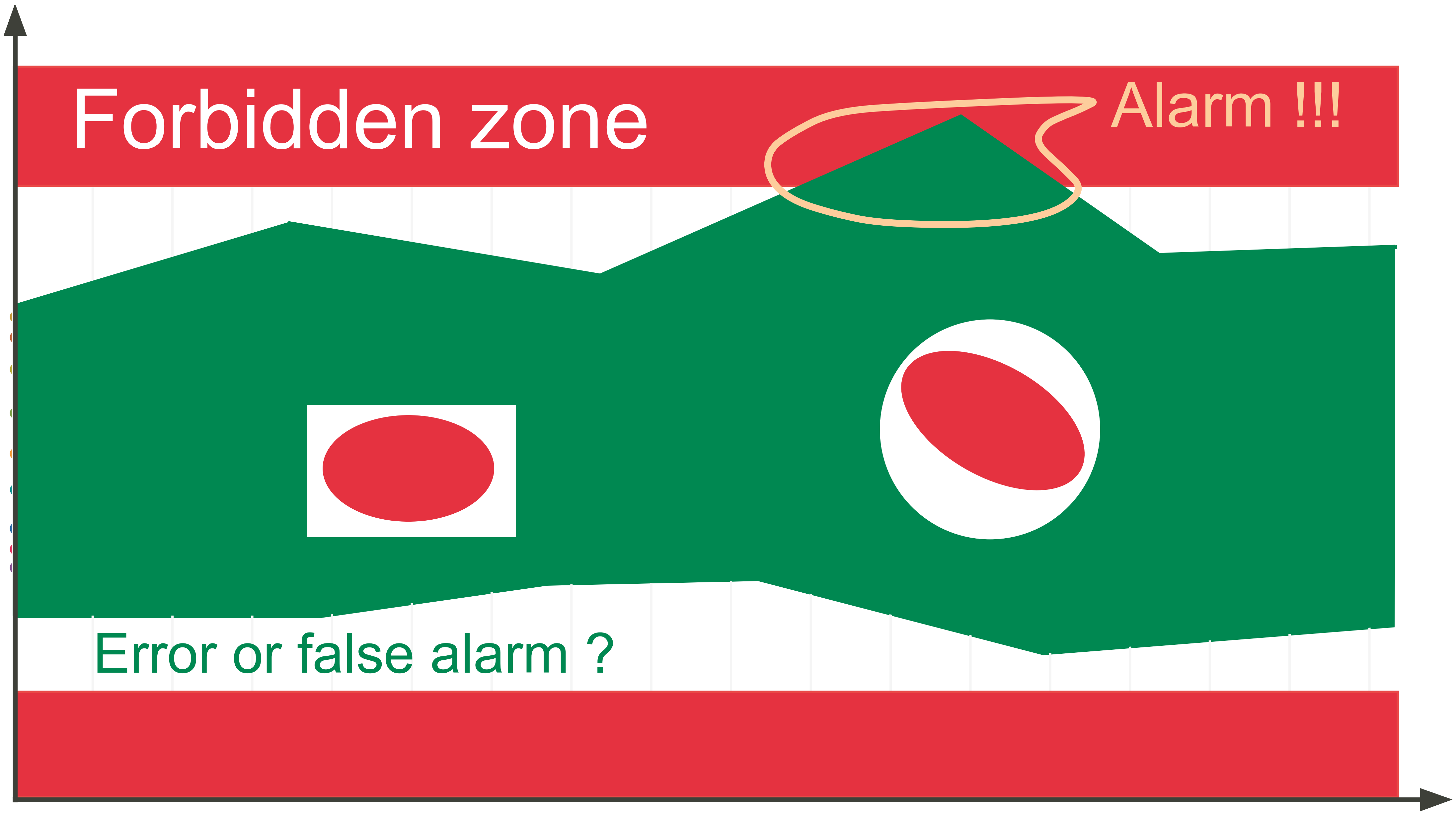
Unsound validation: static analysis

*Many static analysis tools are **unsound** (e.g. Coverity, etc.) so inconclusive*



Incompleteness

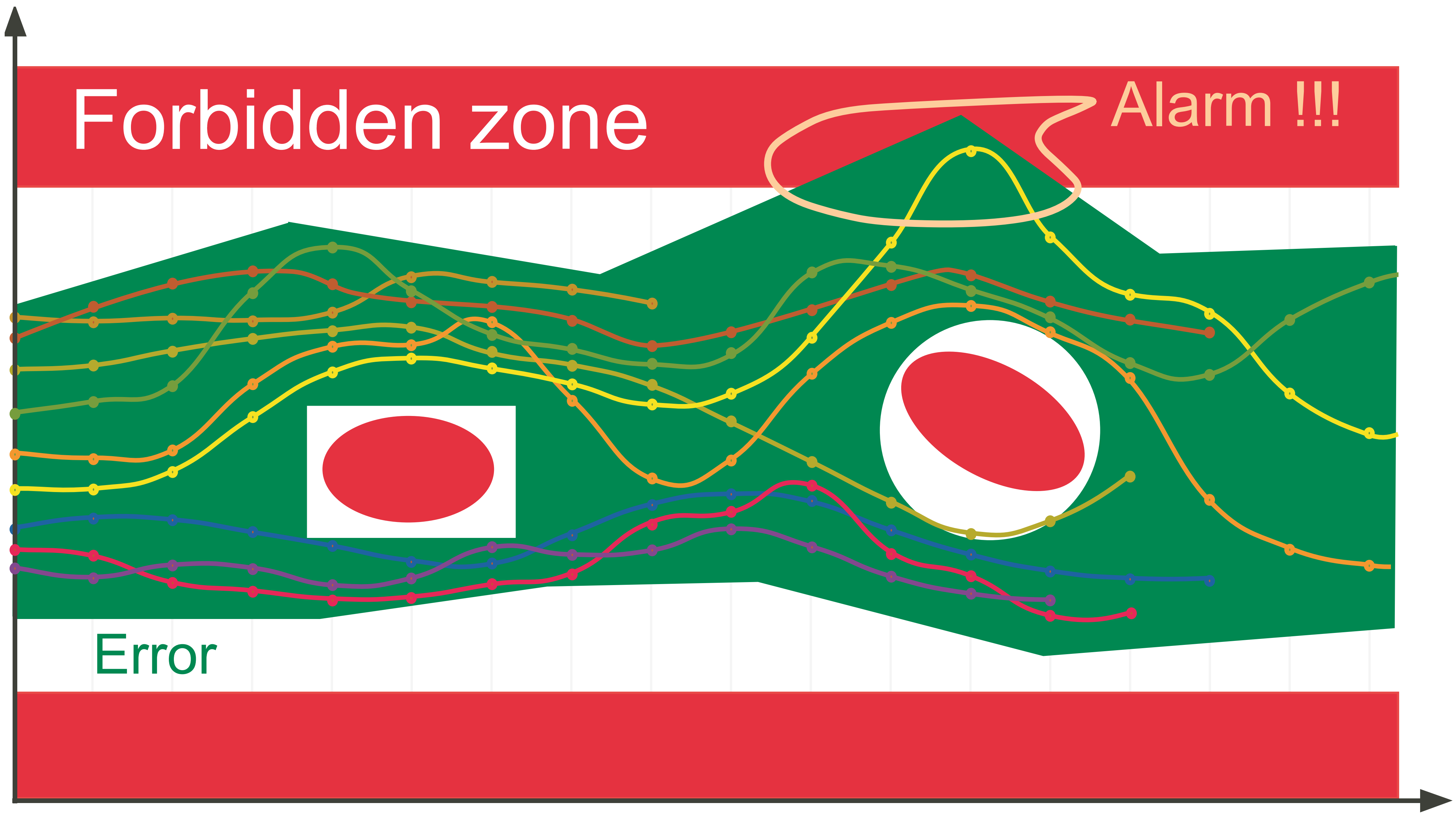
When abstract proofs may fail while concrete proofs would succeed



By soundness an alarm must be raised for this overapproximation!

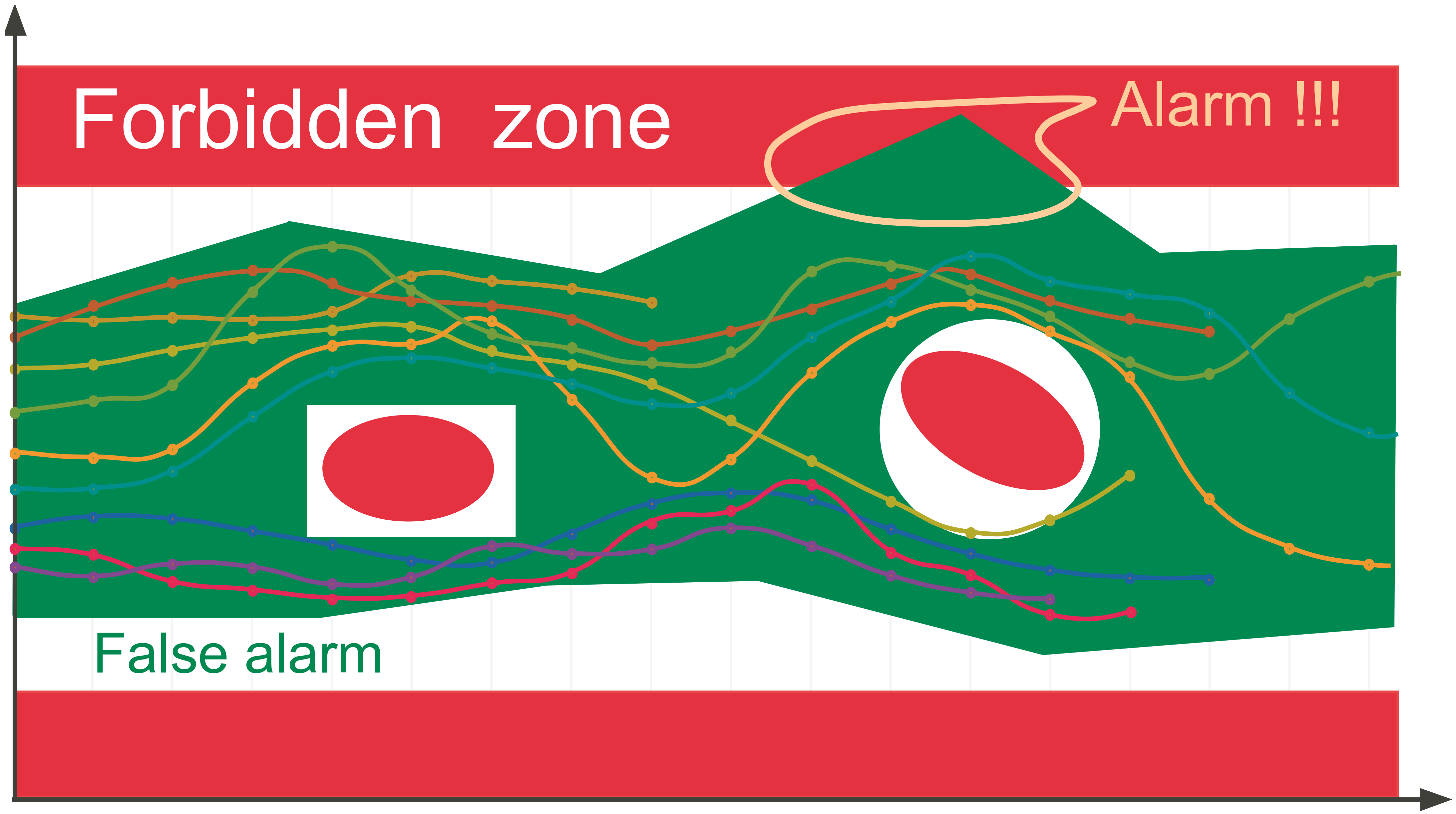
True error

The abstract alarm may correspond to a concrete error



False alarm

The abstract alarm may correspond to no concrete error (false negative)



What to do about false alarms?

- Automatic refinement: inefficient and may not terminate (Gödel)
- Domain-specific abstraction:
  - Adapt the abstraction to the *programming paradigms* typically used in given *domain-specific applications*
  - e.g. *synchronous control/command*: no recursion, no dynamic memory allocation, maximum execution time, etc.

A Touch of Abstract Interpretation Theory

Fixpoint

- Set $\mathcal{P}$
- Transformer $F \in \mathcal{P} \to \mathcal{P}$
- Fixpoint

$x \in \mathcal{P}$ is a fixpoint of F
$\iff F(x) = x$

- Poset $\langle \mathcal{P}, \leqslant \rangle$
- Least fixpoint

$x \in \mathcal{P}$ is the least fixpoint of F (written $x = \mathsf{lfp}^{\leqslant} F$)
$\iff F(x) = x \wedge \forall y \in \mathcal{P} : (F(y) = y) \Rightarrow (x \leqslant y)$

Program properties as fixpoints

- Program semantics and program properties can be formalized as least/greatest fixpoints of increasing transformers on complete lattices (1)
  - *Complete lattice / cpo of properties*

$$\langle \mathcal{P},\ \leqslant,\ 0,\ 1,\ \vee,\ \wedge \rangle$$

  - *Properties* of program P

$$S\,[\![\mathrm{P}]\!] = \mathsf{lfp}^{\leqslant} F\,[\![\mathrm{P}]\!]$$

  - *Transformer* of program P

$$F\,[\![\mathrm{P}]\!] \in \mathcal{P} \to \mathcal{P}, \text{ increasing (or continuous)}$$

---

(1) Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Example: reachable states

- *Transition system* (set of states Σ, initial states $\mathcal{I} \subseteq \Sigma$, transition relation τ)

$$\langle \Sigma,\ \mathcal{I},\ \tau \rangle$$

- *Right-image* of a set of states by transitions

$$\mathrm{post}[\tau]X \triangleq \{s' \mid \exists s \in X : \tau(s, s')\}$$

- *Reachable states* from initial states $\mathcal{I}$

$$\mathrm{post}[\tau^\star]\mathcal{I} = \mathrm{lfp}^{\subseteq} \lambda X \bullet \mathcal{I} \cup \mathrm{post}[\tau]X$$

(I) Patrick Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes. *Thèse És Sciences Mathématiques*, Université Joseph Fourier, Grenoble, France, 21 March 1978

Patrick Cousot. Semantic foundations of program analysis. In S.S. Muchnick & N.D. Jones, editors, *Program Flow Analysis: Theory and Applications,* Ch. 10, pages 303—342, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

Proof methods

- *Proof methods* directly follow from the fixpoint definition

$$S\llbracket \mathrm{P} \rrbracket \leqslant P$$

$$\Leftrightarrow \mathsf{lfp}^{\leqslant} F\llbracket \mathrm{P} \rrbracket \leqslant P$$

$$\Leftrightarrow \exists I : F\llbracket \mathrm{P} \rrbracket(I) \leqslant I \wedge I \leqslant P$$

(proof by Tarski's fixpoint theorem for increasing transformers on complete lattice or Pataria for cpos)

$$\mathsf{lfp}^{\leqslant} F = \bigwedge\{x \mid F(x) \leqslant x\}$$

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Example: Turing/Floyd Invariance Proof

- Bad states

$$\mathcal{B} \subseteq \Sigma$$

- Prove that no bad state is reachable

$$\mathsf{post}[\tau^\star]\mathcal{I} \subseteq \neg\mathcal{B}$$

- Turing/Floyd proof method

$$\exists I \in \wp(\Sigma) : \mathcal{I} \subseteq I \wedge \mathsf{post}[\tau]I \subseteq I \wedge I \subseteq \neg\mathcal{B}$$

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Abstraction

- Abstract the concrete properties into *abstract properties*

$$\langle \mathcal{A},\ \sqsubseteq,\ \bot,\ \top,\ \sqcup,\ \sqcap \rangle$$

- If any concrete property $P \in \mathcal{P}$ has a best abstraction $\alpha(P) \in \mathcal{A}$, then the correspondence is given by a *Galois connection*

$$\langle \mathcal{P},\ \leqslant \rangle \xleftarrow[\alpha]{\gamma} \langle \mathcal{A},\ \sqsubseteq \rangle$$

i.e.

$$\forall P \in \mathcal{P} : \forall Q \in \mathcal{A} : \alpha(P) \sqsubseteq Q \Leftrightarrow P \leqslant \gamma(Q)$$

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Example: elementwise abstraction

- Morphism

$$h \in \mathcal{P} \mapsto \mathcal{A}$$

- Abstraction

$$\alpha(X) \triangleq \{h(x) \mid x \in X\}$$

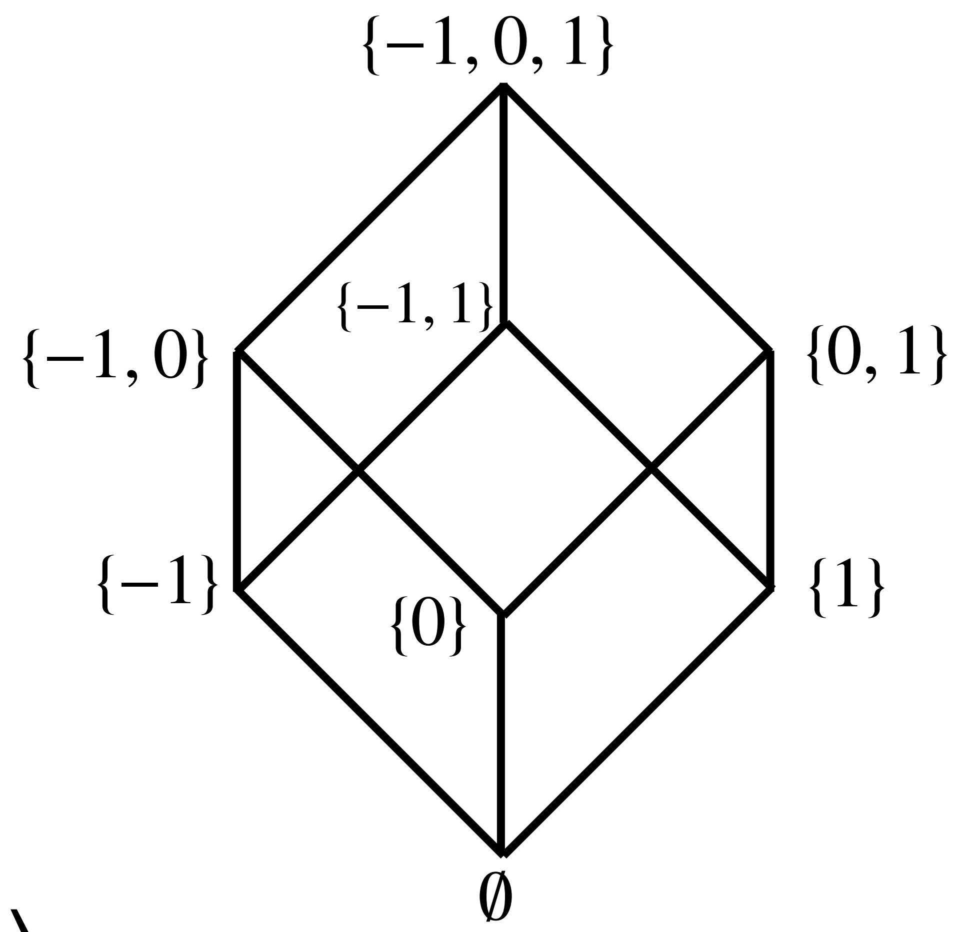
- Galois connection

$$\langle \wp(\mathcal{P}), \subseteq \rangle \xleftarrow[\alpha]{\gamma} \hskip-1em \xrightarrow{} \langle \wp(\mathcal{A}), \subseteq \rangle$$

- Example: rule of signs

$$h : \mathbb{Z} \to \{-1, 0, 1\}$$

$$h(z) \triangleq z/|z|$$



Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Abstract transformer

- An abstract transformer $\overline{F} \in \mathcal{A} \to \mathcal{A}$ is
  - *Sound* iff

$$\forall P \in \mathcal{P} : \alpha \circ F(P) \sqsubseteq \overline{F} \circ \alpha(P)$$

  - *Complete* iff

$$\forall P \in \mathcal{P} : \alpha \circ F(P) = \overline{F} \circ \alpha(P)$$

- Example (rule of sign)
  - Addition: sound, incomplete
  - Multiplication: sound, complete

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Example: rule of signs

$$
\begin{tikzcd}
\{-1, -2, -7\} \arrow[dd, "\alpha"'] & * & \{0, -2, -5\} \arrow[dd, "\alpha"'] & = & \{0, 2, 4, 14, 5, 10, 35\} \arrow[d, phantom, "\subseteq" description] \\
 & & & & \{\, n \mid n \geqslant 0 \,\} \\
\{-1\} & \circledast & \{-1,0\} & = & \{1,0\} \arrow[u, "\gamma"']
\end{tikzcd}
$$

Negative | Negative or zero | Positive or zero

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Fixpoint abstraction

- For an increasing and sound abstract transformer, we have a *fixpoint approximation*

$$\alpha(\mathsf{lfp}^{\leqslant} F) \sqsubseteq \mathsf{lfp}^{\sqsubseteq} \overline{F}$$

- For an increasing, sound, and complete abstract transformer, we have an *exact fixpoint abstraction*

$$\alpha(\mathsf{lfp}^{\leqslant} F) = \mathsf{lfp}^{\sqsubseteq} \overline{F}$$

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Iterative fixpoint computation

- Fixpoint of increasing transformers on cpos can be computed iteratively as limits of (transfinite) iterates

$$F^0 \triangleq \bot$$

$$F^{\beta+1} \triangleq F(F^\beta), \quad \beta + 1 \text{ successor ordinal}$$

$$F^\lambda \triangleq \bigsqcup_{\beta<\lambda} F^\beta, \quad \lambda \text{ limit ordinal}$$

Ultimately stationary at rank ϵ

Converges to $F^\epsilon = \mathsf{lfp}^{\sqsubseteq} F$

- $\epsilon = \omega$ when F is continuous
- Finite iterates when F operates on a cpo satisfying the ascending chain condition

---

Patrick Cousot & Radhia Cousot. Constructive versions of Tarski's fixed point theorems. In *Pacific Journal of Mathematics*, Vol. 82, No. 1, 1979, pp. 43—57.

Widening

- Definition (widening $\triangledown \in \mathcal{A} \times \mathcal{A} \to \mathcal{A}$)
  - $\langle \mathcal{A}, \sqsubseteq \rangle$ poset
  - Over-approximation

$$\forall x, y \in \mathcal{A} : x \sqsubseteq x \triangledown y \wedge y \sqsubseteq x \triangledown y$$

  - Termination

Given any sequence $\langle x^n,\ n \in \mathbb{N} \rangle$, the widened sequence $\langle y^n,\ n \in \mathbb{N} \rangle$

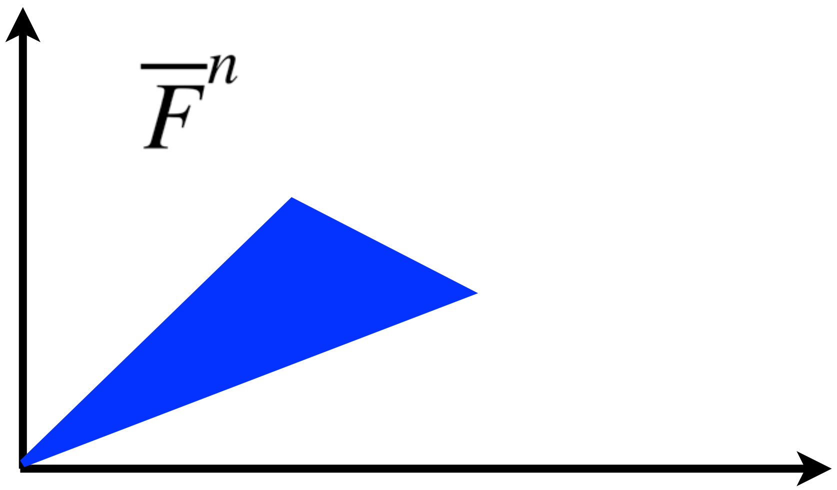
$$y^0 \triangleq x^0, \ldots, y^{n+1} \triangleq y^n \triangledown x^n, \ldots$$

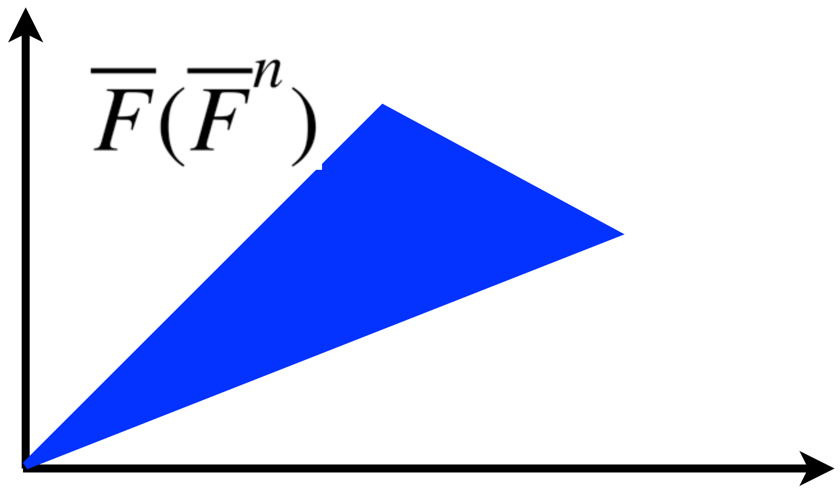
converges to a limit y^ℓ (such that $\forall m \geqslant \ell : y^m = y^\ell$)

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

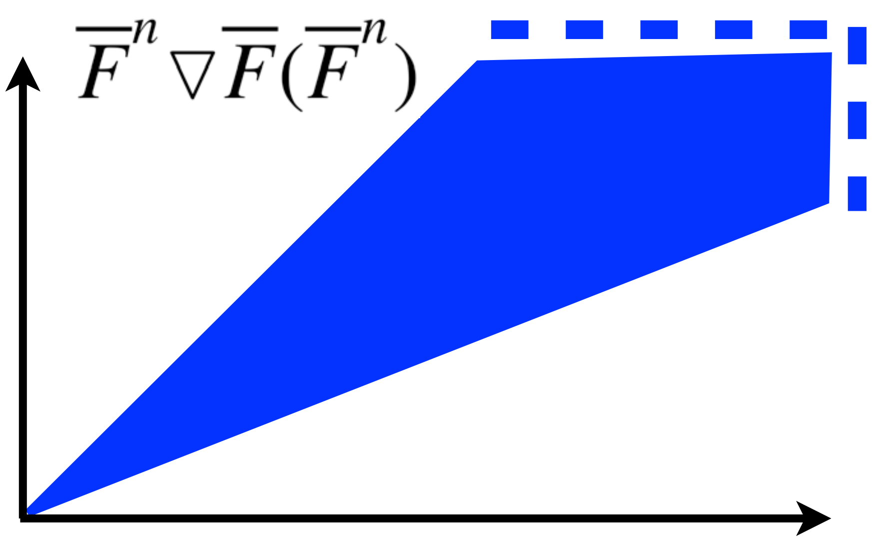
Example: (simple) widening for polyhedra

- Iterates





- Widening



Patrick Cousot, Nicolas Halbwachs: *Automatic Discovery of Linear Restraints Among Variables of a Program*. POPL 1978: 84-96

Iteration with widening

- *Iterates with widening* for transformer $\overline{F} \in \mathcal{A} \to \mathcal{A}$

$$
\begin{aligned}
\overline{F}^0 &\triangleq \bot \\
\overline{F}^{n+1} &\triangleq \overline{F}^n && \text{when } \overline{F}(\overline{F}^n) \sqsubseteq \overline{F}^n \\
\overline{F}^{n+1} &\triangleq \overline{F}^n \,\triangledown\, \overline{F}(\overline{F}^n) && \text{otherwise}
\end{aligned}
$$

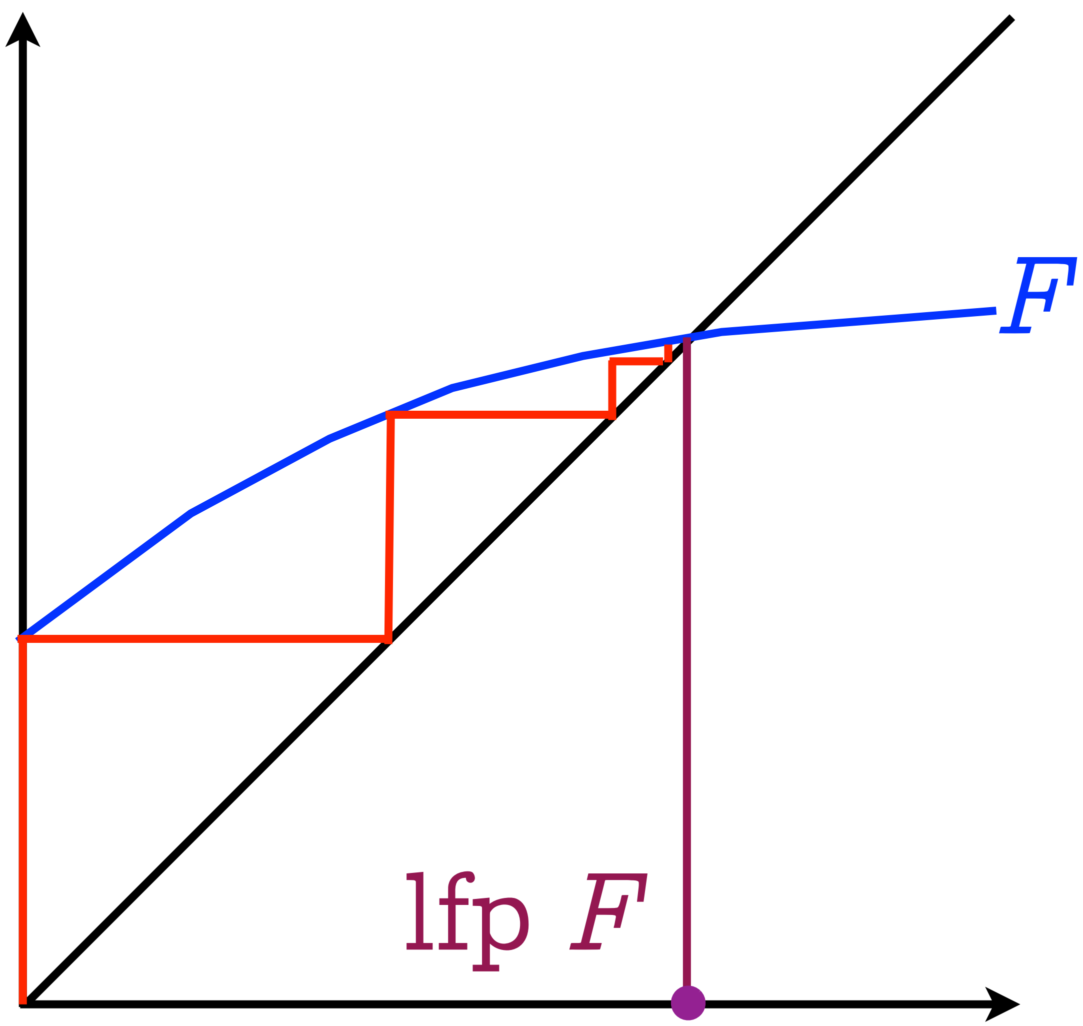
- The *widening speeds up convergence* (at the cost of imprecision)

Theorem (*Limit of iterates with widening*) The iterates of $\overline{F}$ with widening $\triangledown$ from $\bot$ on a poset $\langle \mathcal{A}, \sqsubseteq, \bot \rangle$ converge to a limit $\overline{F}^\ell$ such that $\overline{F}(\overline{F}^\ell) \sqsubseteq \overline{F}^\ell$ (and so $\mathsf{lfp}^{\sqsubseteq}\overline{F} \sqsubseteq \overline{F}^\ell$ when $\overline{F}$ is increasing).

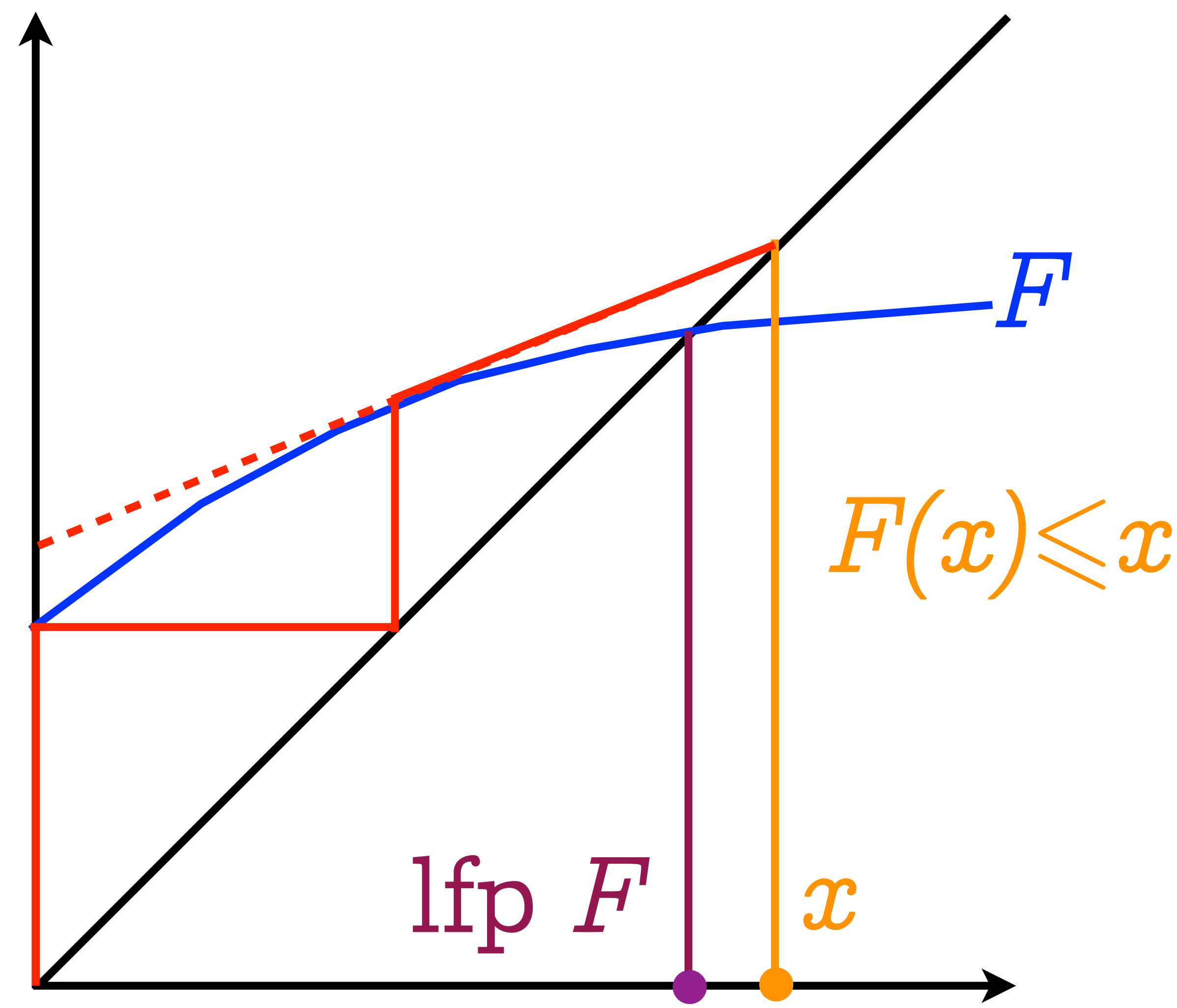
- Can be improved by a *narrowing*.

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Intuition for iteration with widening



Iteration



Iteration with widening (using the derivative as in Newton-Raphson method)

Reduced product

- The reduced product combines abstractions by performing their conjunction in the abstract

$$\langle \mathcal{P},\ \leqslant \rangle \xrightleftharpoons[\alpha_1]{\gamma_1} \langle \mathcal{A}_1,\ \sqsubseteq_1 \rangle$$

$$\langle \mathcal{P},\ \leqslant \rangle \xrightleftharpoons[\alpha_2]{\gamma_2} \langle \mathcal{A}_2,\ \sqsubseteq_2 \rangle$$

$$\mathcal{A}_1 \otimes \mathcal{A}_2 \triangleq$$
$$\{\langle \alpha_1(\gamma_1(P_1) \wedge \gamma_2(P_2)),\ \alpha_2(\gamma_1(P_1) \wedge \gamma_2(P_2)) \rangle \mid P_1 \in \mathcal{A}_1 \wedge P_2 \in \mathcal{A}_2\}$$

$$\langle \mathcal{P},\ \leqslant \rangle \xrightleftharpoons[\alpha_1 \times \alpha_2]{\gamma_1 \times \gamma_2} \langle \mathcal{A}_1 \otimes \mathcal{A}_2,\ \sqsubseteq_1 \times \sqsubseteq_2 \rangle$$

- Example: (positive or zero) ⊗ odd = <positive,odd>

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282
Patrick Cousot, Radhia Cousot, Laurent Mauborgne: The Reduced Product of Abstract Domains and the Combination of Decision Procedures. FOSSACS 2011: 456-472

Recent advances

- The same principles apply to *termination*

Patrick Cousot, Radhia Cousot: *An abstract interpretation framework for termination*. POPL 2012: 245-258

- and to *probabilistic programs*

Patrick Cousot and Michaël Monerau. Probabilistic Abstract Interpretation. In H. Seidel (Ed), *22nd European Symposium on Programming (ESOP 2012)*, Tallinn, Estonia, 24 March—1 April 2012. Lecture Notes in Computer Science, vol. 7211, pp. 166—190, © Springer, 2012.

ASTRÉE

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, Xavier Rival: Why does Astrée scale up? Formal Methods in System Design 35(3): 229-264 (2009)

Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné, Laurent Mauborgne, David Monniaux, Xavier Rival: Varieties of Static Analyzers: A Comparison with ASTREE. TASE 2007: 3-20

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: Combination of Abstractions in the ASTRÉE Static Analyzer. ASIAN 2006: 272-300

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: The ASTREÉ Analyzer. ESOP 2005: 21-30

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: A static analyzer for large safety-critical software. PLDI 2003: 196-207

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival: Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software. The Essence of Computation 2002: 85-108

Target language and applications

- C programming language
  - Without recursion, `longjump`, dynamic memory allocation, conflicting side effects, backward jumps, system calls (stubs)
  - With all its horrors (`union`, pointer arithmetics, etc)
  - Reasonably extending the standard (e.g. size & endianess of integers, IEEE 754-1985 floats, etc)
- Originally for synchronous control/command
  - e.g. generated from Scade

The semantics of C implementations is very hard to define

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]);
}
```

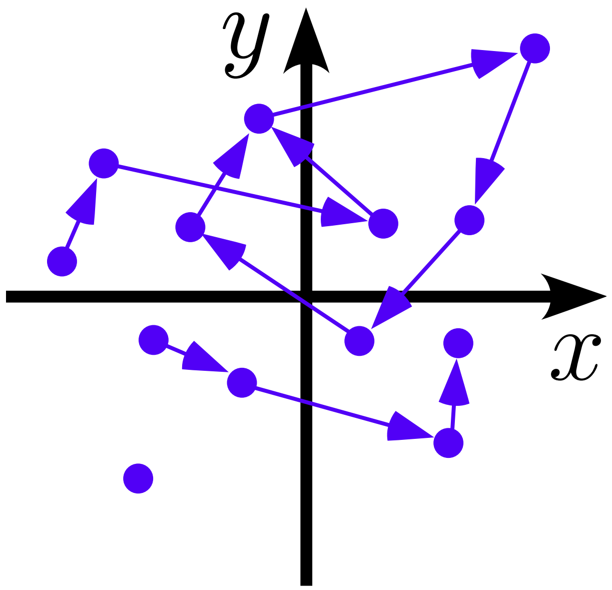
Yields different results on different machines:

`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

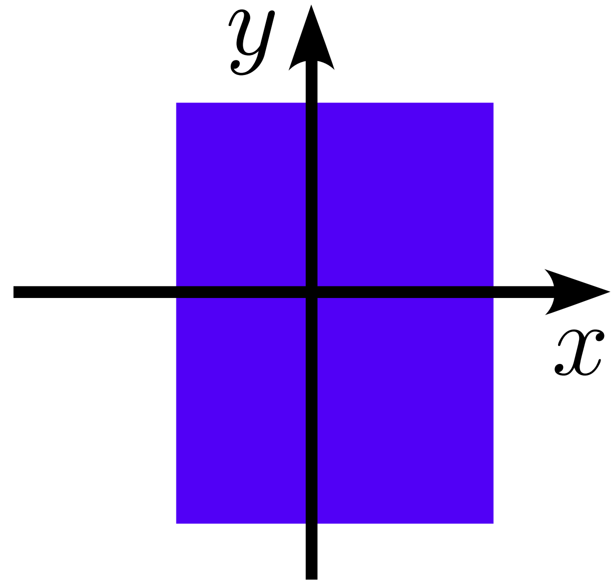
Implicit specification

- Absence of runtime errors: overflows, division by zero, buffer overflow, null & dangling pointers, alignment errors, …
- Semantics of runtime errors:
  - Terminating execution: stop (e.g. floating-point exceptions when traps are activated)
  - Predictable outcome: go on with worst case (e.g. signed integer overflows result in some integer, some options: e.g. modulo arithmetics)
  - Unpredictable outcome: stop (e.g. memory corruption)

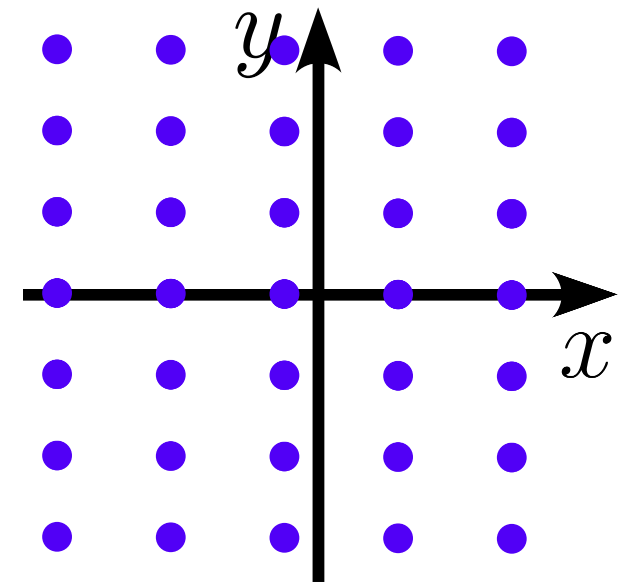
Abstractions



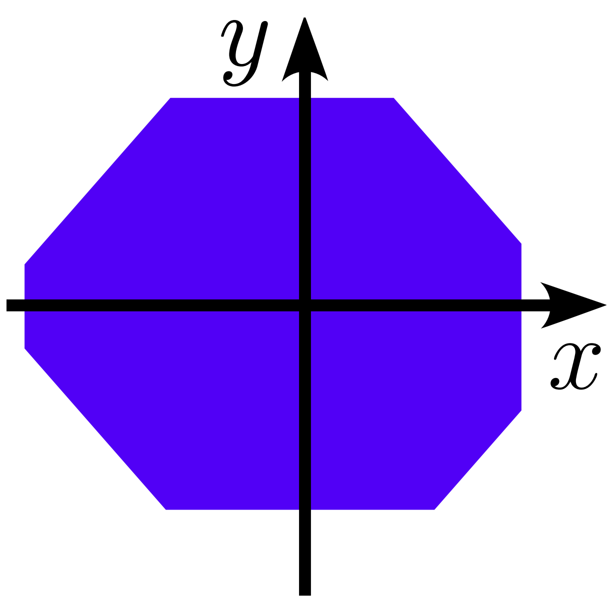
Collecting semantics:
partial traces



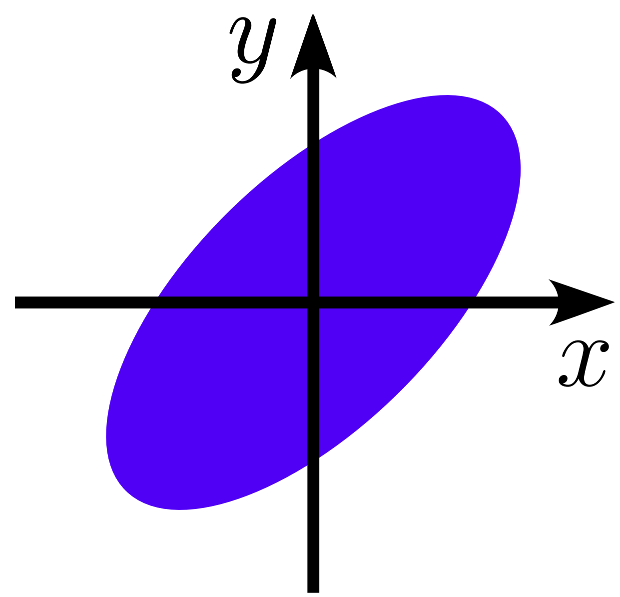
Intervals:
$\mathtt{x} \in [a, b]$



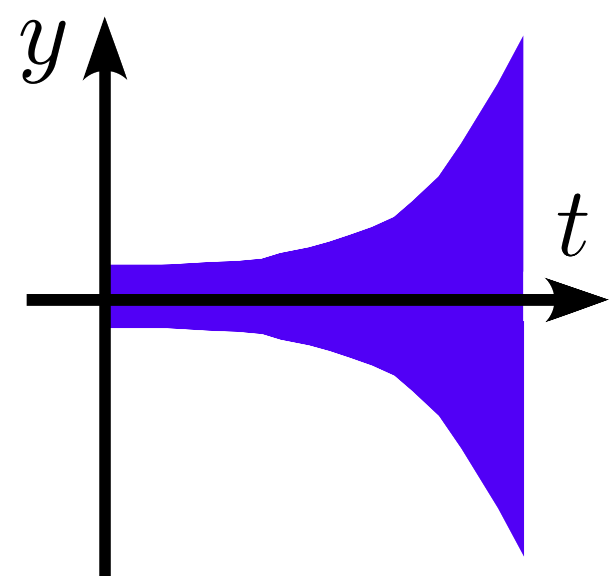
Simple congruences:
$\mathtt{x} \equiv a[b]$



Octagons:
$\pm\mathtt{x} \pm \mathtt{y} \leqslant a$



Ellipses:
$\mathtt{x}^2 + b\mathtt{y}^2 - a\mathtt{x}\mathtt{y} \leqslant d$



Exponentials:
$-a^{bt} \leqslant \mathtt{y}(t) \leqslant a^{bt}$

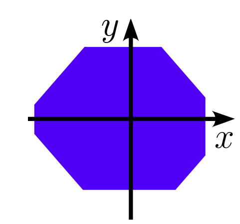
Example of general purpose abstraction: octagons

- Invariants of the form $\pm\, \mathtt{x} \pm \mathtt{y} \leq \mathtt{c}$, with $\mathcal{O}(\mathbf{N}^2)$ memory and $\mathcal{O}(\mathbf{N}^3)$ time cost.
- Example:

```
while (1) {
  R = A-Z;
  L = A;
  if (R>V)
    { ★ L = Z+V; }
  ★
}
```

- At ★, the interval domain gives $\mathtt{L} \leq \max(\max \mathtt{A}, (\max \mathtt{Z}) + (\max \mathtt{V}))$.
- In fact, we have $\mathtt{L} \leq \mathtt{A}$.
- To discover this, we must know at ★ that $\mathtt{R} = \mathtt{A}\text{-}\mathtt{Z}$ and $\mathtt{R} > \mathtt{V}$.

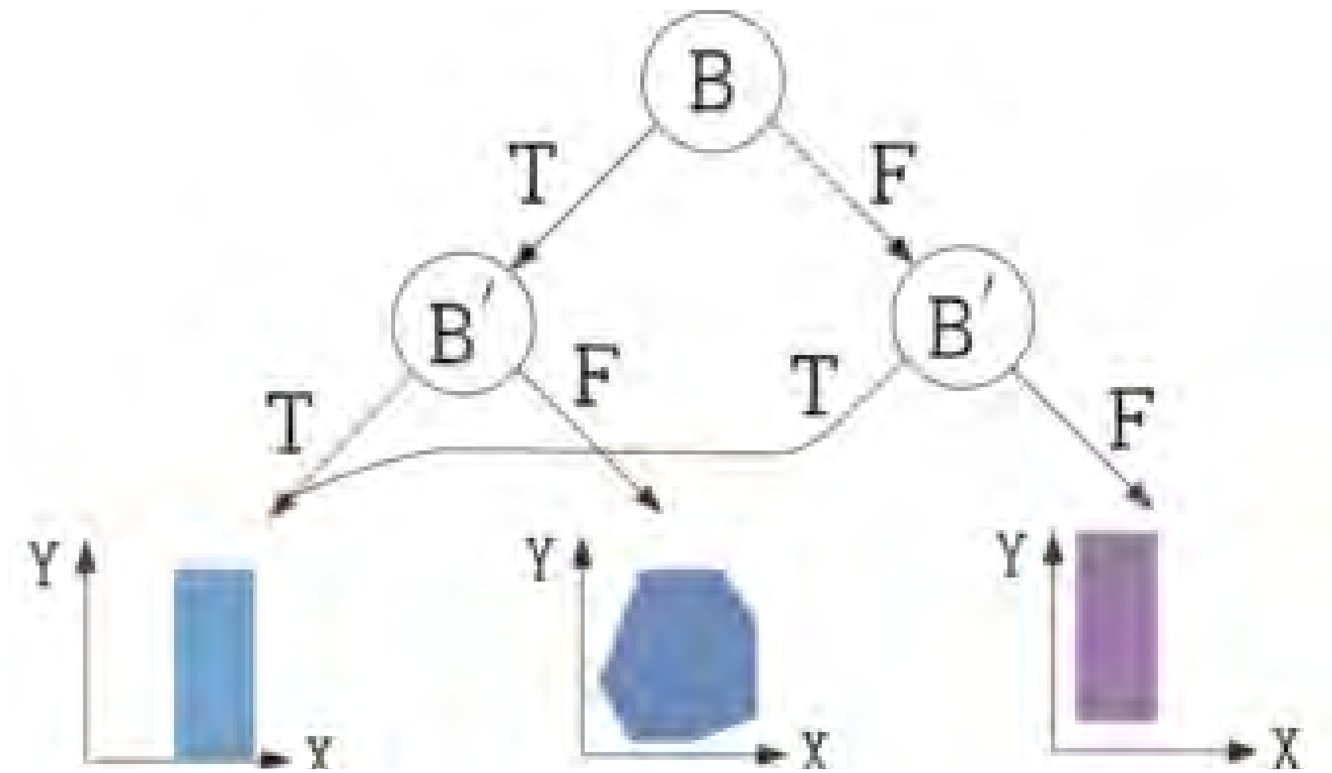
- Here, $\mathtt{R} = \mathtt{A}\text{-}\mathtt{Z}$ cannot be discovered, but we get $\mathtt{L}\text{-}\mathtt{Z} \leq \max \mathtt{R}$ which is sufficient.
- We use many octagons on **small packs** of variables instead of a large one using all variables to cut costs.



Antoine Miné: *The octagon abstract domain*. Higher-Order and Symbolic Computation 19(1): 31-100 (2006)

Example of general purpose abstraction: decision trees

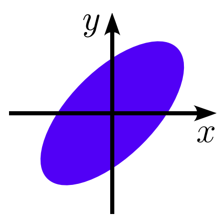
```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leaves

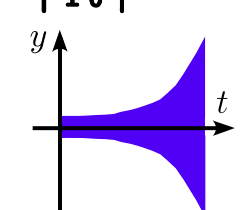
Example of domain-specific abstraction: ellipses

```
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; }
}
```



Example of domain-specific abstraction: exponentials

```
% 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; }                ← potential overflow!
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
% 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.
```



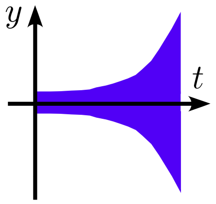
Example of domain-specific abstraction: exponentials

```
% 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
```



Jérôme Feret: *The Arithmetic-Geometric Progression Abstract Domain*. VMCAI 2005: 42-58

An erroneous common belief on static analyzers

"The properties that can be proved by static analyzers are often simple" [2]

Like in mathematics:

- May be simple to state (no overflow)
- But harder to discover (`S[0], S[1] in [-1327.02698354, 1327.02698354]`)
- And difficult to prove (since it requires finding a non trivial non-linear invariant for second order filters with complex roots [Fer04], which can hardly be found by exhaustive enumeration)

Reference

[2] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, No. 7, July 2008.

[Fer04] Jérôme Feret: Static Analysis of Digital Filters. ESOP 2004: 33-48

Industrial applications

Daniel Kästner, Christian Ferdinand, Stephan Wilhelm, Stefana Nevona, Olha Honcharova, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, Xavier Rival, and Élodie-Jane Sims. Astrée: Nachweis der Abwesenheit von Laufzeitfehlern. In *Workshop ``Entwicklung zuverlässiger Software-Systeme''*, Regensburg, Germany, June 18th, 2009.

Olivier Bouissou, Éric Conquet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Khalil Ghorbal, Éric Goubault, David Lesens, Laurent Mauborgne, Antoine Miné, Sylvie Putot, Xavier Rival, & Michel Turin. Space Software Validation using Abstract Interpretation. In *Proc. of the Int. Space System Engineering Conf., Data Systems in Aerospace (DASIA 2009)*. Istambul, Turkey, May 2009, 7 pages. ESA.

Jean Souyris, David Delmas: Experimental Assessment of Astrée on Safety-Critical Avionics Software. SAFECOMP 2007: 479-490

David Delmas, Jean Souyris: Astrée: From Research to Industry. SAS 2007: 437-451

Jean Souyris: Industrial experience of abstract interpretation-based static analyzers. IFIP Congress Topical Sessions 2004: 393-400

Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, Christian Ferdinand: An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. DSN 2003: 625-632

Examples of applications

- Verification of the absence of runtime-errors in
  - Fly-by-wire flight control systems (*)

  - ATV docking system (*)

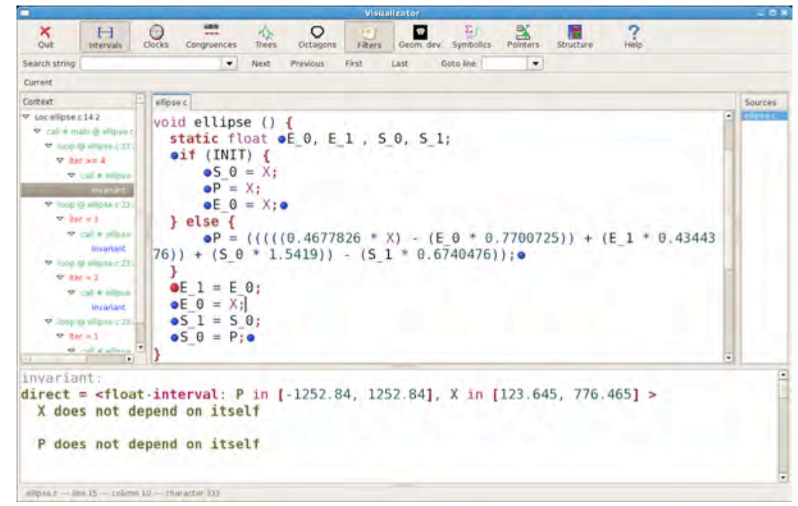
  - Flight warning system (on-going work)

(*) No false alarm a all!

Industrialization

- 8 years of research (CNRS/ENS/INRIA):

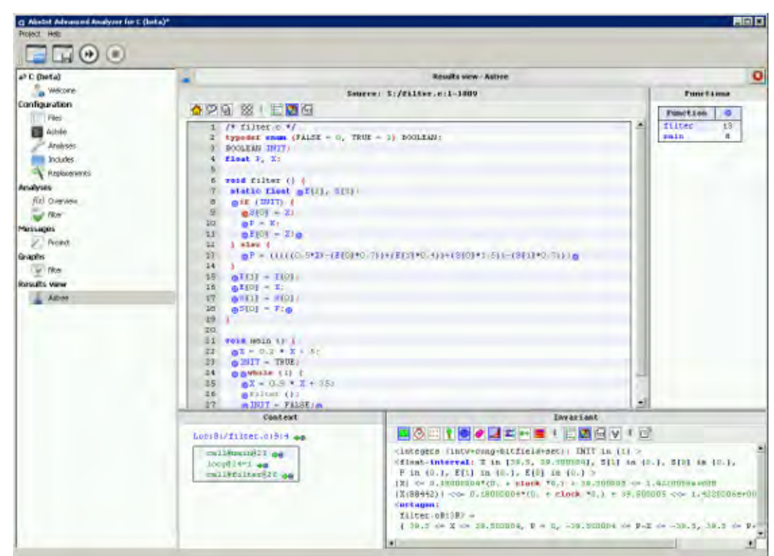
www.astree.ens.fr

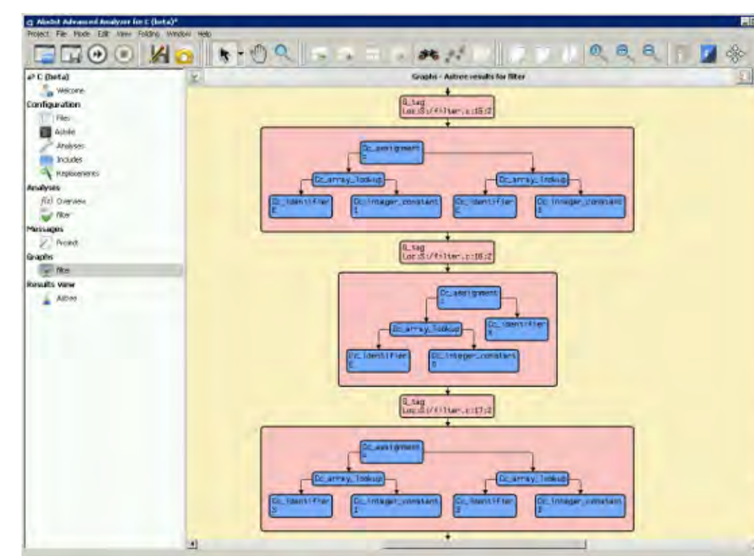


- Industrialization by AbsInt (since Jan. 2010):

www.absint.com/astree/

AbsInt
Angewandte Informatik





On-going work

ASTRÉEA: Verification of embedded real-time parallel C programs

---

Antoine Miné: Static *Analysis of Run-Time Errors in Embedded Critical Parallel C Programs*. ESOP 2011: 398-418

Parallel programs

- Bounded number of processes with shared memory, events, semaphores, message queues, blackboards,…
- Processes created at initialization only
- Real time operating system (ARINC 653) with fixed priorities (highest priority runs first)
- Scheduled on a single processor

Verified properties

- Absence of runtime errors
- Absence of unprotected data races

Semantics

- No memory consistency model for C
- Optimizing compilers consider sequential processes out of their execution context

init: flag1 = flag2 = 0	
process 1:	process 2:

`flag1 = 1;`
`if (!flag2)`
`{`
`  /* critical section */`
	`flag2 = 1;`
	`if (!flag1)`
	`{`
	`  /* critical section */`

write to `flag1/2` and read of `flag2/1` are independent so can be reordered → error!

- We assume:
  - sequential consistency in absence of data race
  - for data races, values are limited by possible interleavings between synchronization points

Abstractions

- Based on Astrée for the sequential processes
- Takes scheduling into account
- OS entry points (semaphores, logbooks, sampling and queuing ports, buffers, blackboards, …) are all stubbed (using Astrée stubbing directives)
- Interference between processes: flow-insensitive abstraction of the writes to shared memory and inter-process communications

Example of application: FWS

- Degraded mode (5 processes, 100 000 LOCS)
  - 1h40 on 64-bit 2.66 GHz Intel server
  - A few dozens of alarms
- Full mode (15 processes, 1 600 000 LOCS)
  - 24 h
  - a few hundreds of alarms !!! work going on !!! (e.g. analysis of complex data structures, logs, etc)

Conclusion

Cost-effective verification

- The rumor has it that:
  - Manuel validation (testing/debugging/bug finding) is costly, unsafe, not a verification!
  - Formal proofs by theorem provers are extremely laborious hence costly to create and maintain for program/specifications changing over time (15/20 years for planes)
  - Model-checkers are unsound or do not scale up for complex software (which is unbounded)

Cost-effective verification

- Why not try abstract interpretation?
  - Domain-specific static analysis scales and can deliver no or few false alarms on large industrial code!
  - Conceptual bugs are discovered through their consequences on runtime errors
  - Very cost effective
  - Compliant with DO178C formal methods!

The End, Thank You