

The NASA logo is a blue circle containing the word "NASA" in white, bold, sans-serif capital letters. A white orbital line with an arrow at its end circles the text. A red diagonal line, representing a spacecraft's path, cuts across the logo from the bottom left to the top right. The background of the logo is filled with small white stars.

---

# Applying Formal Methods to Safety-Critical Systems

---

J. Tanner Slagel

NASA Langley Research Center  
Hampton, VA, USA

[j.tanner.slagel@nasa.gov](mailto:j.tanner.slagel@nasa.gov)

April 2022 @ MAA MD-DC-VA Section Meeting  
Montgomery College, Germantown, Maryland

# Cows

---

**Proposition:** All cows in a field are the same color

**Proposition:** All cows in a field are the same color

## Principle of weak induction

For a proposition on positive natural numbers  $P$ , if

1.  $P(1)$  and
2.  $P(N)$  implies  $P(N + 1)$  for each positive natural number  $N$ ,  
then

$P(N)$  is true for all positive natural numbers  $N$

# Cows

---

**Proposition:** All cows in a field are the same color

## Principle of weak induction

For a proposition on positive natural numbers  $P$ , if

1.  $P(1)$  and
2.  $P(N)$  implies  $P(N + 1)$  for each positive natural number  $N$ ,  
then

$P(N)$  is true for all positive natural numbers  $N$

**Proof:** Let  $P(N) = 'N \text{ cows in a field are the same color}'$

Apply weak induction:

1. All cows in a field of one cow has the same color

# Cows

---

**Proposition:** All cows in a field are the same color

## Principle of weak induction

For a proposition on positive natural numbers  $P$ , if

1.  $P(1)$  and
2.  $P(N)$  implies  $P(N + 1)$  for each positive natural number  $N$ ,  
then

$P(N)$  is true for all positive natural numbers  $N$

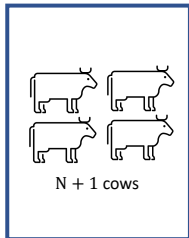
**Proof:** Let  $P(N) = 'N \text{ cows in a field are the same color}'$

Apply weak induction:

1. All cows in a field of one cow has the same color
2. Assume  $N$  cows in a field are the same color, and suppose there is a field with  $N + 1$  COWS...

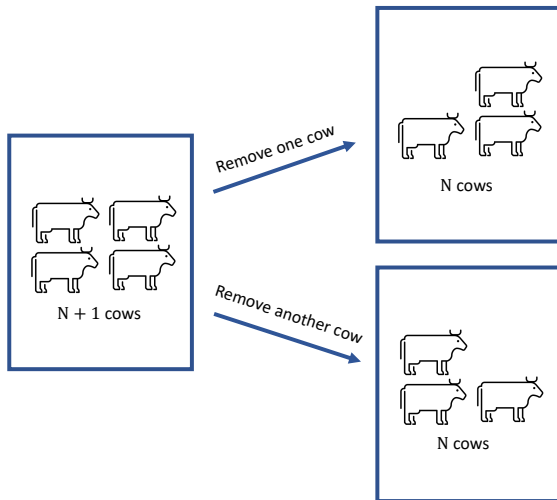
# Inductive step

---

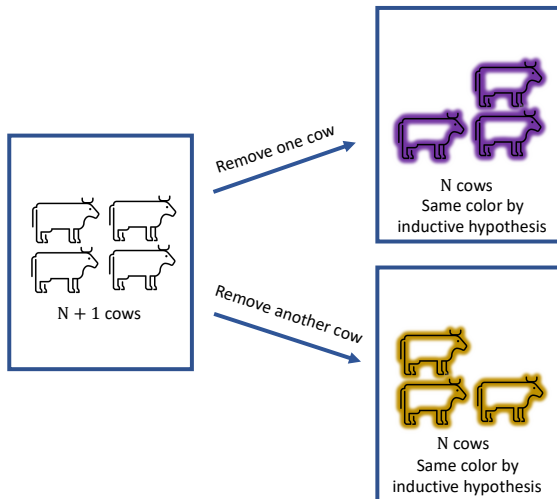


# Inductive step

---

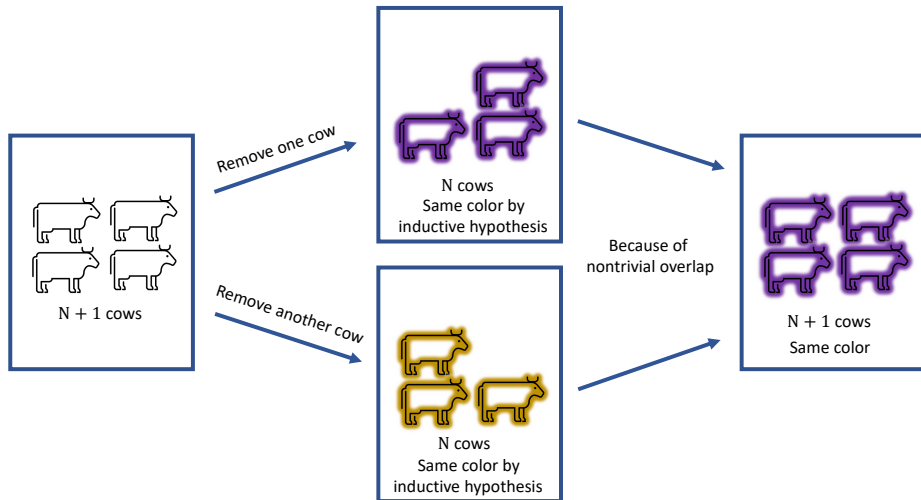


# Inductive step

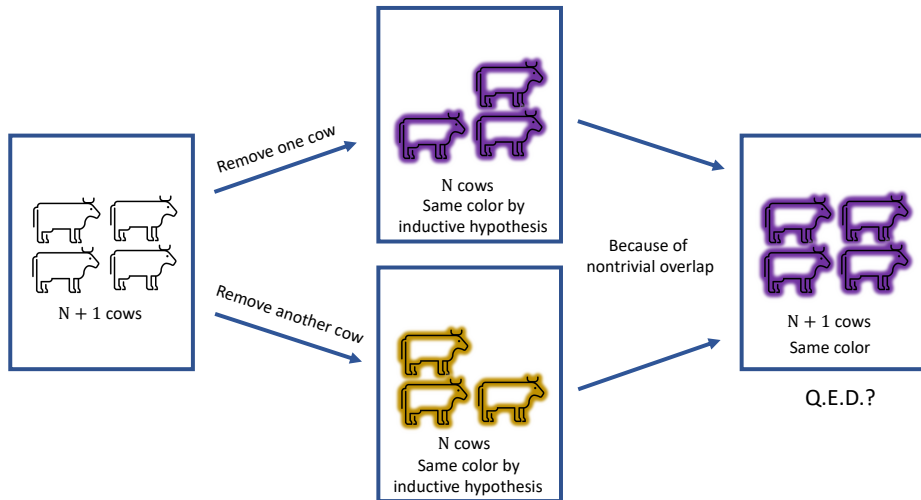




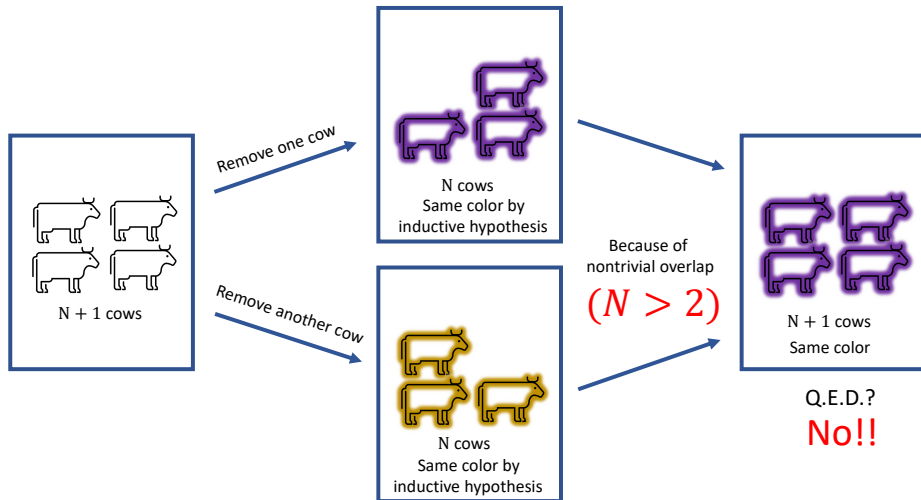
# Inductive step



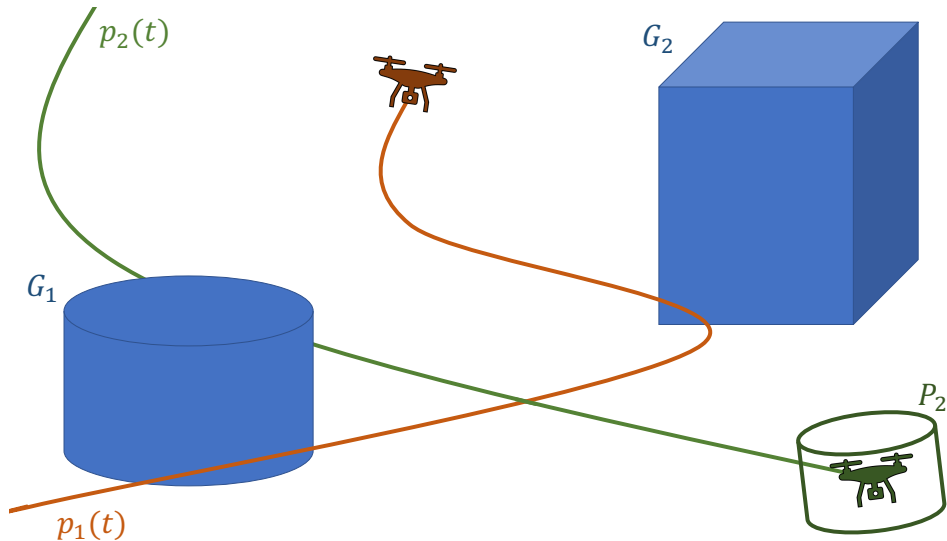
# Inductive step?



# Inductive step? Not a correct one



# Safe polynomial airspace



# Polynomial airspace

---

- Aircraft path  $p : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^3$

$$p(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}^\top$$

- $x, y, z : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$  polynomials in  $t$
- **Obstacle** (geofence, well-clear volume) defined by the conjunction (ands) of polynomial inequalities

$$G = \left\{ \begin{bmatrix} x & y & z & t \end{bmatrix}^\top \mid g_1(x, y, z, t) \leq 0 \wedge \cdots \wedge g_n(x, y, z, t) \leq 0 \right\}$$

- $g_i$  is polynomial of  $x, y, z$ , and  $t$
- **Violation** at  $t \in \mathbb{R}_{\geq 0}$

$$p(t) \in G \iff \begin{cases} g_1(x(t), y(t), z(t), t) & \leq 0 \\ & \vdots \\ g_n(x(t), y(t), z(t), t) & \leq 0 \end{cases}$$

# Polynomial airspace

- **Violation** at  $t \in \mathbb{R}_{\leq 0}$

$$p(t) \in G \iff \begin{cases} g_1(x(t), y(t), z(t), t) & \leq 0 \\ & \vdots \\ g_n(x(t), y(t), z(t), t) & \leq 0 \end{cases}$$

## Goal

- **Detect** when a **violation** occurs
- **Avoid** a violation by producing a **resolution**  $\hat{p}(t)$

# Polynomial airspace

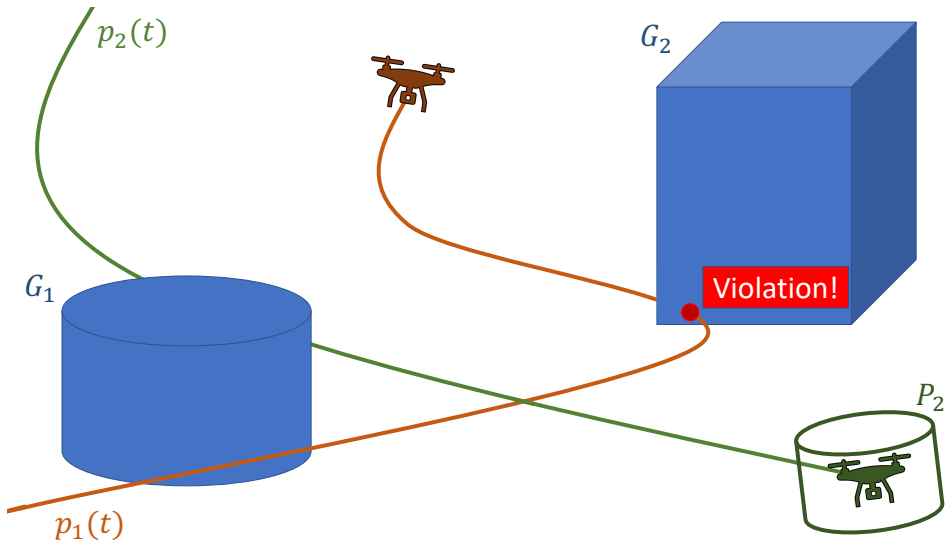
- **Violation** at  $t \in \mathbb{R}_{\leq 0}$

$$p(t) \in G \iff \begin{cases} g_1(x(t), y(t), z(t), t) & \leq 0 \\ & \vdots \\ g_n(x(t), y(t), z(t), t) & \leq 0 \end{cases}$$

## Goal

- **Detect** when a **violation** occurs
- **Avoid** a violation by producing a **resolution**  $\hat{p}(t)$

# Unsafe polynomial airspace





# Detecting a violation

- **Violation** at  $t \in \mathbb{R}_{\leq 0}$

$$p(t) \in G \iff \begin{cases} g_1(x(t), y(t), z(t), t) & \leq 0 \\ & \vdots \\ g_n(x(t), y(t), z(t), t) & \leq 0 \end{cases}$$

- $g_i(x(t), y(t), z(t), t)$  is a single-variable polynomial in  $t$  for each  $i \leq n$
- No violation at the roots of  $g_i$  for all  $i \leq n \implies$  no violation anywhere

## Claim

Given a single evaluation of each polynomial and the roots with multiplicity information

- The existence of a violation can be determined
- The first instance  $t^*$  of a violation can be determined

# Detecting a violation, example

---

**Example:** Path

$$p(t) = \begin{bmatrix} t & 2t & 3t \end{bmatrix}^\top$$

Geofence  $G$  defined by

$$\begin{cases} z & \geq 3 \\ z & \leq 12 \\ (x - 2)^2 + (3y - 9)^2 & \leq 25 \end{cases}$$

# Detecting a violation, example

---

**Example:** Path

$$p(t) = \begin{bmatrix} t & 2t & 3t \end{bmatrix}^\top$$

Geofence  $G$  defined by

$$\begin{cases} z & \geq 3 \\ z & \leq 12 \\ (x-2)^2 + (3y-9)^2 & \leq 25 \end{cases} \implies \begin{cases} 3-z & \leq 0 \\ z-12 & \leq 0 \\ (x-2)^2 + (3y-9)^2 - 25 & \leq 0 \end{cases}$$

# Detecting a violation, example

**Example:** Path

$$p(t) = \begin{bmatrix} t & 2t & 3t \end{bmatrix}^\top$$

Geofence  $G$  defined by

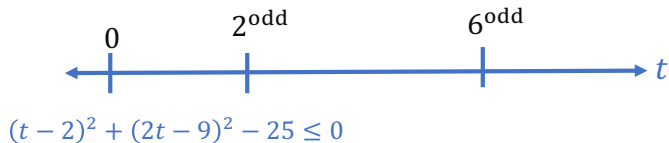
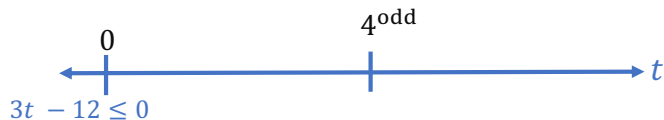
$$\begin{cases} z & \geq 3 \\ z & \leq 12 \\ (x-2)^2 + (3y-9)^2 & \leq 25 \end{cases} \implies \begin{cases} 3-z & \leq 0 \\ z-12 & \leq 0 \\ (x-2)^2 + (3y-9)^2 - 25 & \leq 0 \end{cases}$$

Checking for violations:

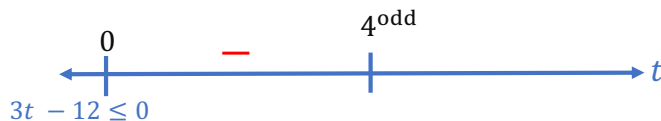
$$\begin{cases} 3-3t & \leq 0 \\ 3t-12 & \leq 0 \\ (t-2)^2 + (3(2t)-9)^2 - 25 & \leq 0 \end{cases} \xrightarrow[\text{with multiplicities}]{\text{finding roots}} \begin{cases} t = 3^{\text{odd}} \\ t = 4^{\text{odd}} \\ t = 2^{\text{odd}}, t = 6^{\text{odd}} \end{cases}$$

## Detecting a violation, example (continued)

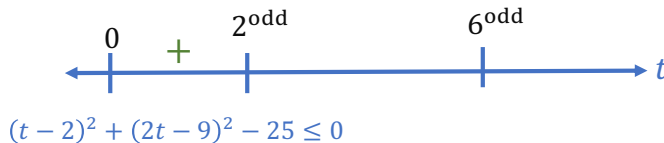
---



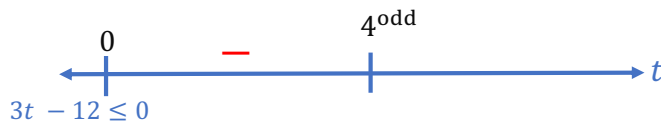
## Detecting a violation, example (continued)



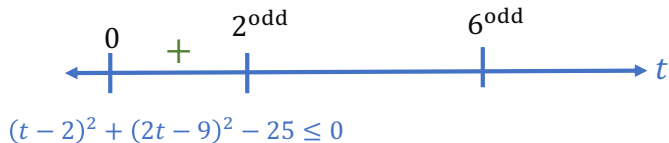
$$v = \begin{bmatrix} + \\ - \\ + \end{bmatrix}$$



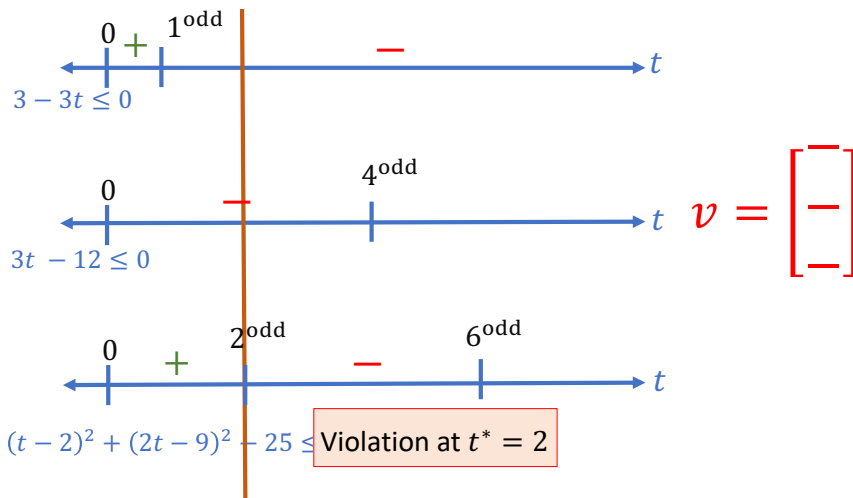
## Detecting a violation, example (continued)



$$v = \begin{bmatrix} - \\ - \\ + \end{bmatrix}$$

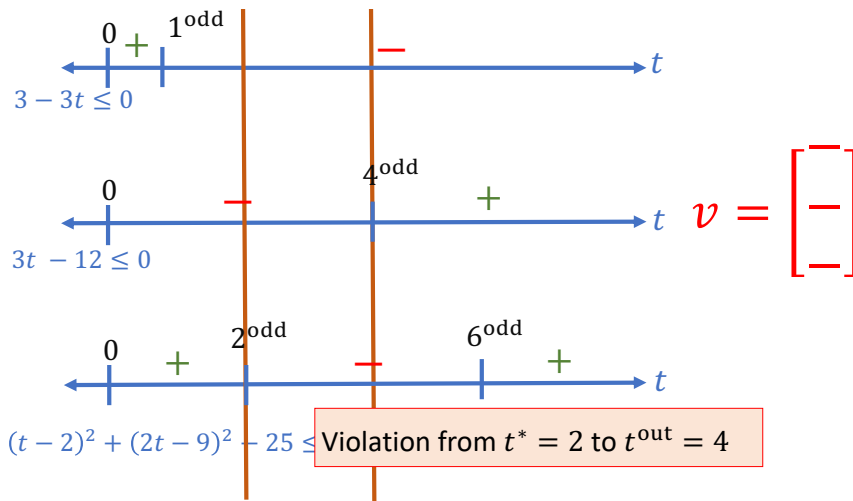


## Detecting a violation, example (continued)

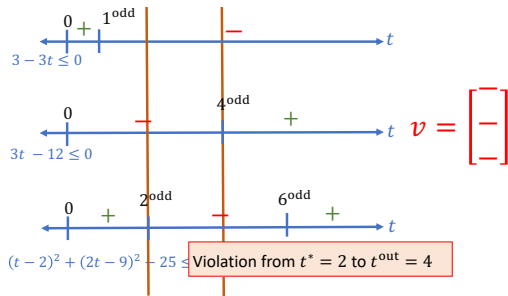
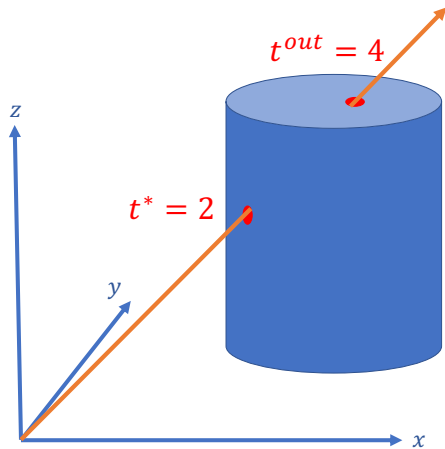




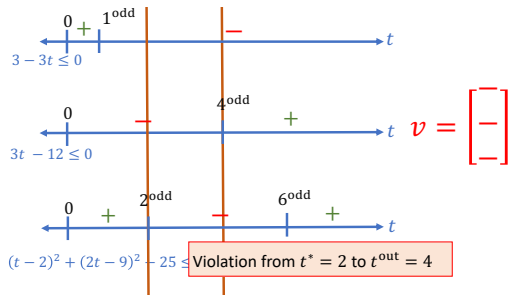
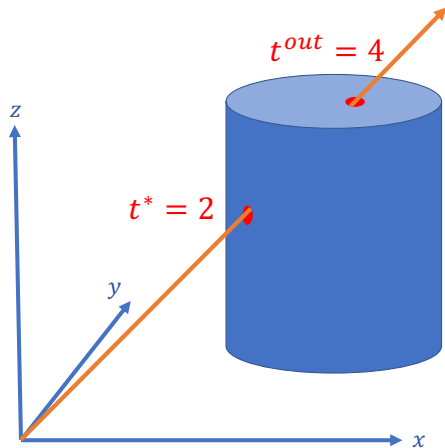
## Detecting a violation, example (continued)



# Detecting a violation, example (continued)



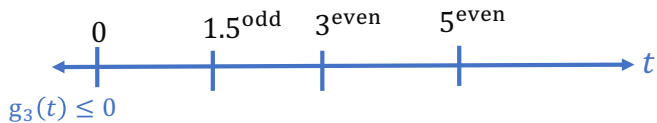
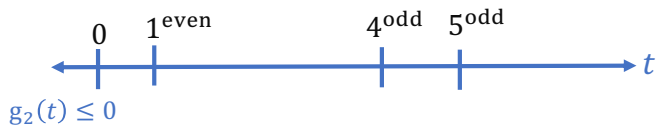
# Detecting a violation, example (continued)



How can we know if this always works?

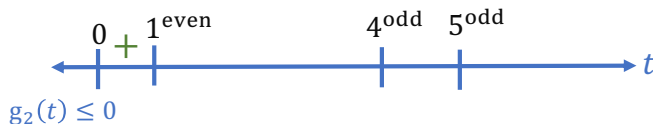
## Another example

---

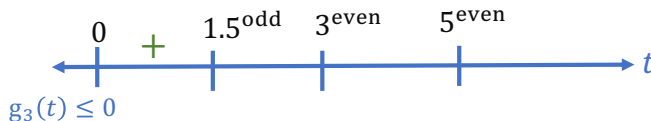


## Another example

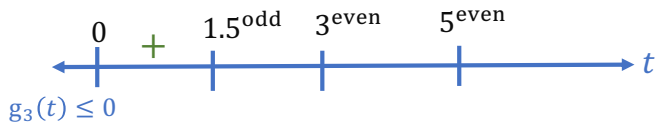
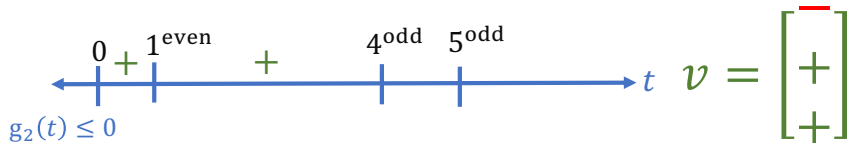
---



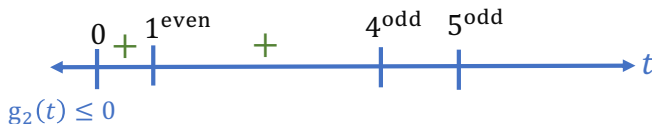
$$v = \begin{bmatrix} + \\ + \\ + \end{bmatrix}$$



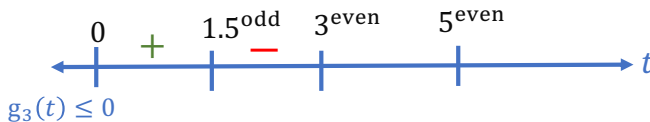
## Another example



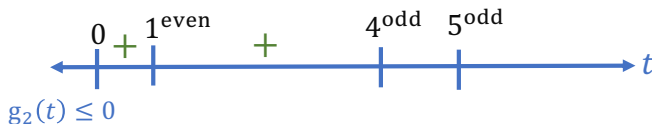
## Another example



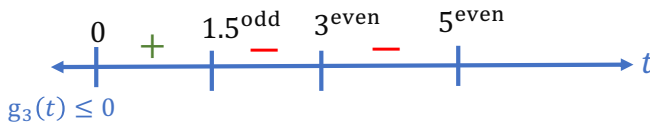
$$v = \begin{bmatrix} - \\ + \\ - \end{bmatrix}$$



## Another example

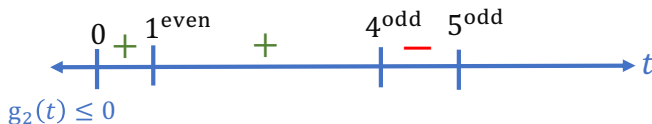


$$v = \begin{bmatrix} - \\ + \\ - \end{bmatrix}$$

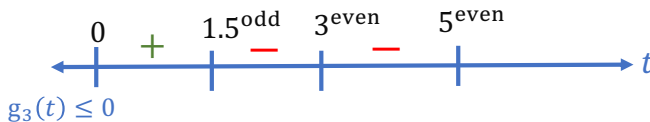




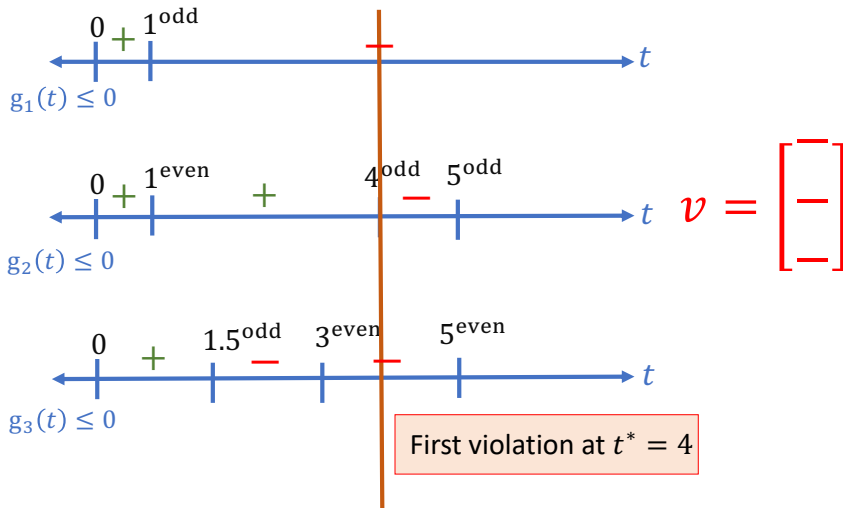
## Another example



$$v = \begin{bmatrix} - \\ - \\ - \end{bmatrix}$$



## Another example



# Safety- and mission-critical operations

---

## Software

- Aircraft, satellite, space shuttle
- Cybersecurity
- Medical equipment
- Banking, blockchain

## Hardware

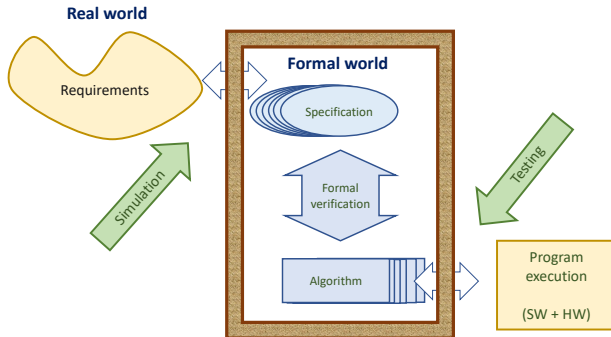
- Computer processors
- Fingerprint readers
- Avionics

## Systems

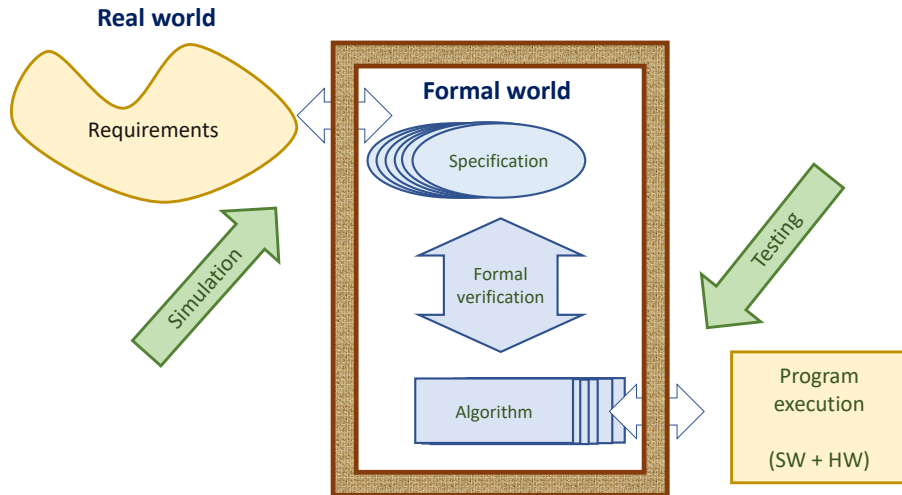
- National airspace / urban airspace
- Autonomous vehicles
- Power plants
- Mission defense systems

# Formal methods

- ‘Applying mathematically rigorous techniques for specification and verification of software and/or hardware’
  - Formal specifications are well-formed statements in a mathematical logic
  - Formal verification uses a set of inference rules to prove properties of formal specifications
- Experimentation is not enough in safety-critical applications



# Formal methods

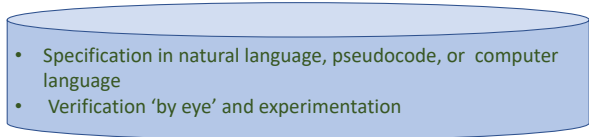


# Levels of formal methods

---

Example:

Level 0

- 
- Specification in natural language, pseudocode, or computer language
  - Verification 'by eye' and experimentation

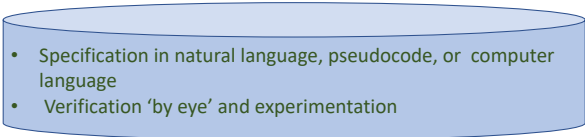
# Levels of formal methods

---

Example:

"If one aircraft is always slower than another aircraft, they will never get too close."

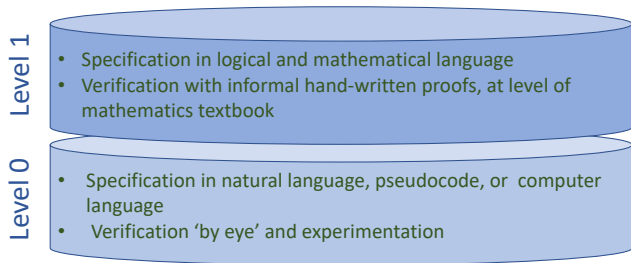
Level 0

- 
- Specification in natural language, pseudocode, or computer language
  - Verification 'by eye' and experimentation

# Levels of formal methods

---

Example:





# Levels of formal methods

Level 1

- Specification in logical and mathematical language
- Verification with informal hand-written proofs, at level of mathematics textbook

Level 0

- Specification in natural language, pseudocode, or computer language
- Verification 'by eye' and experimentation

## Example:

“Let the path of two aircraft be given by

$$p_1, p_2 : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$$

such that

$$p_1'(t) \leq p_2'(t),$$

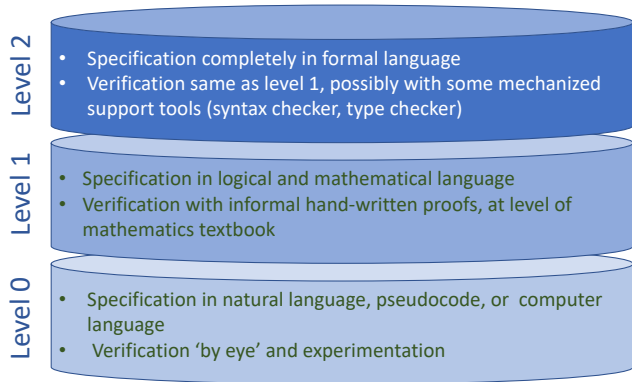
for all  $t \in \mathbb{R}_{\geq 0}$ . If  $p_1(0) \leq p_2(0)$  and  $|p_1(0) - p_2(0)| \geq D_{\text{safe}}$  then for all  $t \in \mathbb{R}_{\geq 0}$ ,

$$|p_1(0) - p_2(0)| \geq D_{\text{safe}}.”$$

# Levels of formal methods

---

Example:



# Levels of formal methods

Level 2  
Level 1  
Level 0

- Specification completely in formal language
- Verification same as level 1, possibly with some mechanized support tools (syntax checker, type checker)

- Specification in logical and mathematical language
- Verification with informal hand-written proofs, at level of mathematics textbook

- Specification in natural language, pseudocode, or computer language
- Verification 'by eye' and experimentation

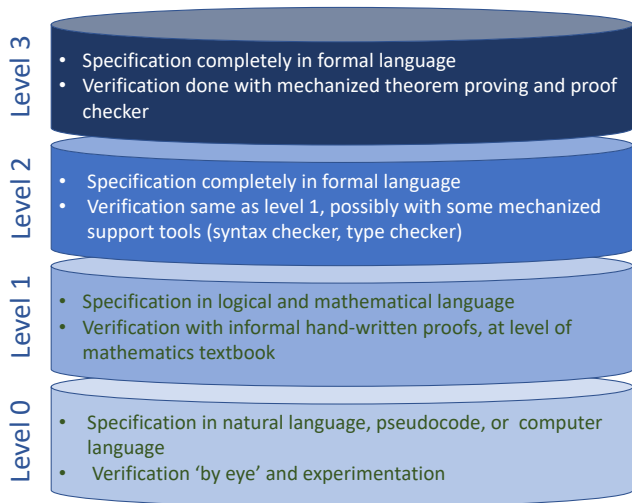
Example:

```
follow_safe: LEMMA
  FORALL(p1,p2:position, Dsafe:nnreal,
    | t1: nnreal):
    abs(p1(0)-p2(0)) < Dsafe
  AND p2(0) < p1(0)
  AND FORALL(t:nnreal):
    deriv(p1)(t) < deriv(p2)(t)
  IMPLIES
    FORALL(t:nnreal):
      abs(p1(t)-p2(t)) < Dsafe
```

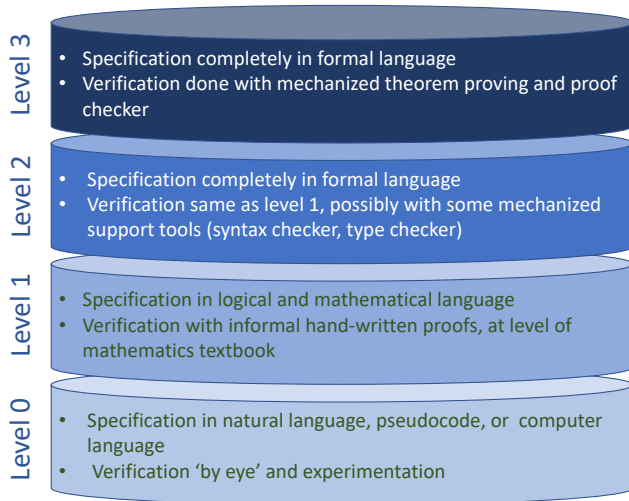
# Levels of formal methods

---

Example:



# Levels of formal methods



## Example:

```
follow_safe: LEMMA
  FORALL(p1,p2:position, Dsafe:nnreal,
    t1: nnreal):
    abs(p1(0)-p2(0)) < Dsafe
  AND p2(0) < p1(0)
  AND FORALL(t:nnreal):
    deriv(p1)(t) < deriv(p2)(t)
  IMPLIES
    FORALL(t:nnreal):
      abs(p1(t)-p2(t)) < Dsafe
```

```
follow_safe : PROOF
(then (skip)
  (spread (lemma "monotonic_antideriv_gt[nnreal]")
    ((spread (case "t1=0")
      ((then (assert)(typepred "t1")(hide -3)
        (expand "Safe?")(propax))
      (then (expand "Safe?")(skip)(expand "safe?")
        (expand "violation?")(inst -1 "0" "t" "p1" "p2")(ass
          (spread (split -1)
            ((then (move-terms -2 1 1)(expand "abs" 2 1)(asser
              (then (skip)(expand "slower?" -2)(inst -2 "x")
                (expand "vel" -2)(assert))))))
          (then (assert)(lemma "connected_nnreal")(propax))))))
  QED follow_safe
```

# PVS

## Specification (.pvs)

```
8      % Define half
9      half(a:real,b:real | b>a):
10         {r:real | abs(a-r) = abs(b-r)} =
11             (a+b)/2
12
13      % Theorem about half
14      prove | show-proofite
15      half_sq: THEOREM
16         FORALL(a:real,b:real | b>a):
17             EXISTS(n:posnat):
18                 a>n AND b>n
19                 IMPLIES half(a,b) < half(a^n,b^n)
```

## Interactive theorem prover

```
half_sq.1.1 :
{-1}  b < b ^ 2
[-2]  a < a ^ 2
[-3]  a > 2
[-4]  b > 2
|-----
[1]   half(a, b) < half(a ^ 2, b ^ 2)

>> (expand "half")

- Ctrl+SPACE shows the full list of commands.
- TAB autocompletes commands. Double click expands definitions.
```

[6]: PVS 7.1 official webpage: <https://pvs.csl.sri.com/>

[7]: VScode-PVS <https://github.com/nasa/vscode-pvs>

# PVS

## Proof (.prt)

```
7 half_sq : PROOF
8 (then (skeep)(inst 1 "2")(flatten)
9   (spread (case "a<a^2")
10    ((spread (case "b<b^2")
11     ((then (expand "half")(mult-by 1 "2")(assert))
12      (then (div-by 1 "b")(grind))))
13    (then (div-by 1 "a")(grind))))))
14 QED half_sq
```

## Interactive theorem prover

```
half_sq.1.1 :
{-1} b < b ^ 2
[-2] a < a ^ 2
[-3] a > 2
[-4] b > 2
[1] half(a, b) < half(a ^ 2, b ^ 2)

>> (expand "half")
```

- Ctrl+SPACE shows the full list of commands.
- TAB autocompletes commands. Double click expands definitions.

[6]: PVS 7.1 official webpage: <https://pvs.csl.sri.com/>

[7]: VScode-PVS <https://github.com/nasa/vscode-pvs>

# PVS

---

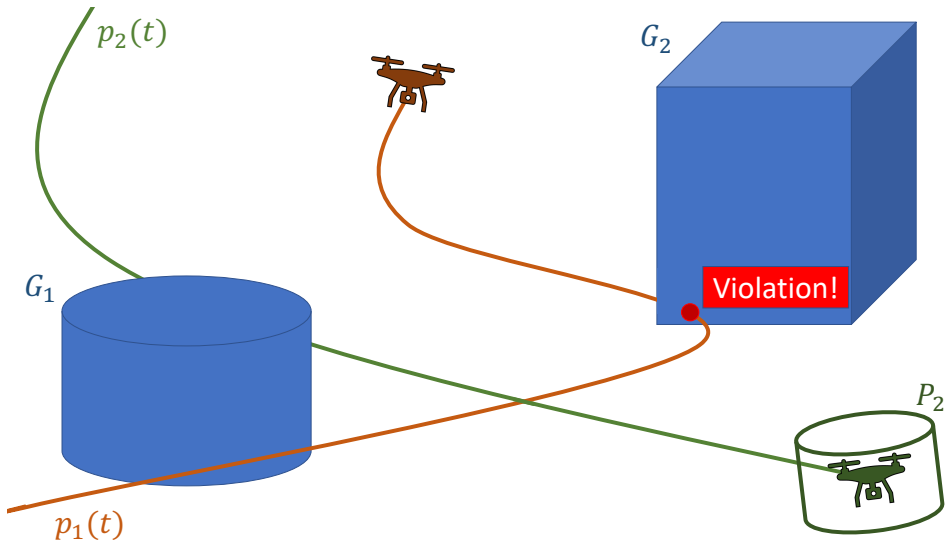
- 'Prototype Verification System' - developed by SRI International
- Interactive theorem prover
  - Higher order logic
  - Completely typed, dependent types
- Automation
  - Customizable tactics and strategies
  - Floating point analysis tool PRECiSA
  - Real-number proving
    - Interval arithmetic
    - Affine arithmetic
    - Numerical integration
    - Tarski/Storm
- Animation and rapid prototyping - PVSio
- VS-Code PVS



- [9] NASAlib is available at <https://github.com/nasa/pvslib>



## Back to PolySafe



# Formal analysis of PolySafe

---

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. for  $i, \dots, n$  do
2.   calculate roots with multiplicities of  $g_i$
3. end for
4. sort roots in ascending order
5. calculate  $v$ , vector of signs of polynomials at zero
6. for each root do
7.   update  $v$  at current root
8.   if all entries of  $v$  are '-' then
9.     return current root
10.   end if
11. end for
12. return FALSE

# Formal analysis of PolySafe

---

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. for  $i, \dots, n$  do
2.   calculate roots with multiplicities of  $g_i$
3. end for
4. sort roots in ascending order
5. calculate  $v$ , vector of signs of polynomials at zero
6. for each root do
7.   update  $v$  at current root
8.   if all entries of  $v$  are '-' then
9.     return current root
10.   end if
11. end for
12. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

# Formal analysis of PolySafe

---

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. for  $i, \dots, n$  do
2.   calculate roots with multiplicities of  $g_i$
3. end for
4. **sort roots in ascending order**
5. calculate  $v$ , vector of signs of polynomials at zero
6. for each root do
7.   update  $v$  at current root
8.   if all entries of  $v$  are '-' then
9.     return current root
10.   end if
11. end for
12. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

There could be

- Complex roots
- Infinitely many roots

# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. for  $i, \dots, n$  do
2.   calculate roots with multiplicities of  $g_i$
3. end for
4. **sort roots in ascending order**
5. calculate  $v$ , vector of signs of polynomials at zero
6. for each root do
7.   update  $v$  at current root
8.   if all entries of  $v$  are '-' then
9.     return current root
10. end if
11. end for
12. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

There could be

- Complex roots
- Infinitely many roots ( $g_i = 0$ )

Example:

$$p(t) = \begin{bmatrix} \frac{1}{2}t & t - 1 & 2t + 2 \end{bmatrix}^T$$

$$g_1(x, y, z, t) = -32x + -4y^2 + z^2$$

# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. for  $i, \dots, n$  do
2.   calculate roots with multiplicities of  $g_i$
3. end for
4. **sort roots in ascending order**
5. calculate  $v$ , vector of signs of polynomials at zero
6. for each root do
7.   update  $v$  at current root
8.   if all entries of  $v$  are '-' then
9.     return current root
10. end if
11. end for
12. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

There could be

- Complex roots
- Infinitely many roots ( $g_i = 0$ )

Example:

$$p(t) = \begin{bmatrix} \frac{1}{2}t & t - 1 & 2t + 2 \end{bmatrix}^\top$$

$$g_1(x, y, z, t) = -32x + -4y^2 + z^2$$

$$\forall t, \quad g_1(x(t), y(t), z(t), t) = 0$$

# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. **discard zero polynomials, update  $G$**
2. **for**  $i, \dots, n$  **do**
3.   calculate roots with multiplicities of  $g_i$
4.   **discard non-real roots, negative roots**
5. **end for**
6. **sort roots in ascending order**
7. calculate  $v$ , vector of signs of polynomials at zero
8. **for** each root **do**
9.   update  $v$  at current root
10.   **if** all entries of  $v$  are '-' **then**
11.     **return** current root
12.   **end if**
13. **end for**
14. **return** FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

There could be

- Complex roots
- Infinitely many roots ( $g_i = 0$ )

Example:

$$p(t) = \begin{bmatrix} \frac{1}{2}t & t - 1 & 2t + 2 \end{bmatrix}^T$$

$$g_1(x, y, z, t) = -32x + -4y^2 + z^2$$

$$\forall t, \quad g_1(x(t), y(t), z(t), t) = 0$$



# Formal analysis of PolySafe

---

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. for  $i, \dots, n$  do
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. end for
6. sort roots in ascending order
7. calculate  $v$ , vector of signs of polynomials at zero
8. for each root do
9.   update  $v$  at current root
10.   if all entries of  $v$  are '-' then
11.     return current root
12.   end if
13. end for
14. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

# Formal analysis of PolySafe

---

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. for  $i, \dots, n$  do
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. end for
6. sort roots in ascending order
7. **calculate  $v$ , vector of signs of polynomials at zero**
8. for each root do
9.   update  $v$  at current root
10.   if all entries of  $v$  are '-' then
11.     return current root
12.   end if
13. end for
14. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

If 0 is the root of  $g_i$ , then computing  $v$  fails

# Formal analysis of PolySafe

---

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. **for**  $i, \dots, n$  **do**
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. **end for**
6. sort roots in ascending order
7. **choose**  $c$  **less than all roots**
8. **calculate**  $v$ , **vector of signs of polynomials at**  $c$
9. **for** each root **do**
10.   update  $v$  at current root
11.   **if** all entries of  $v$  are '-' **then**
12.     **return** current root
13.   **end if**
14. **end for**
15. **return** FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

**If 0 is the root of  $g_i$ , then computing  $v$  fails**

# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. **for**  $i, \dots, n$  **do**
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. **end for**
6. sort roots in ascending order
7. choose  $c$  less than all roots
8. calculate  $v$ , vector of signs of polynomials at  $c$
9. **for** each root **do**
10.   **update**  $v$  **at current root**
11.   **if all entries of**  $v$  **are** '−' **then**
12.     return current root
13.   **end if**
14. **end for**
15. return FALSE

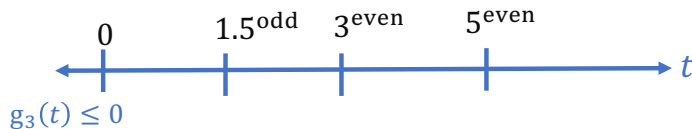
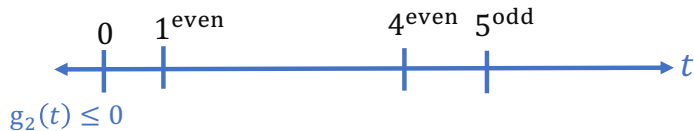
**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

## Catastrophic failure

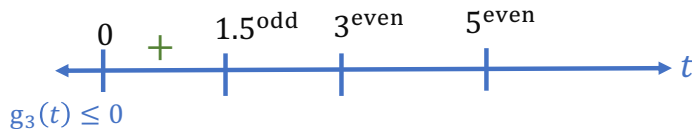
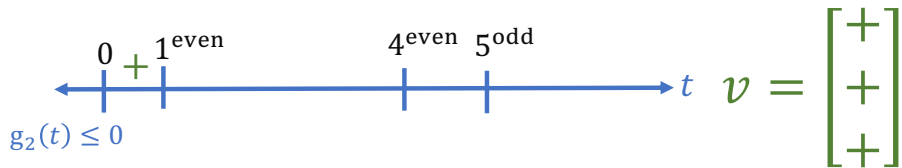
The update strategy of  $v$  is flawed, and can result in a violation not being detected

# Update fail

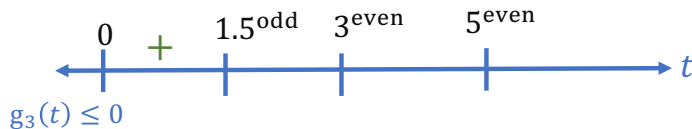
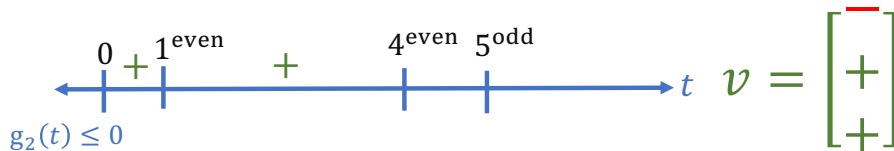
---



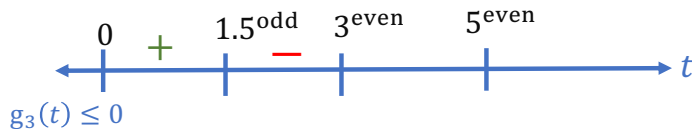
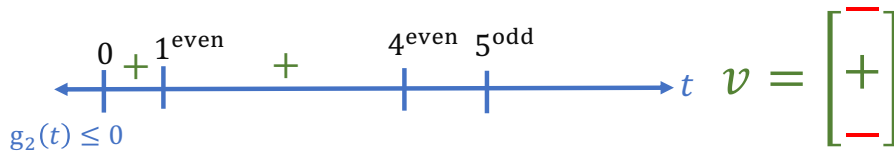
# Update fail



# Update fail

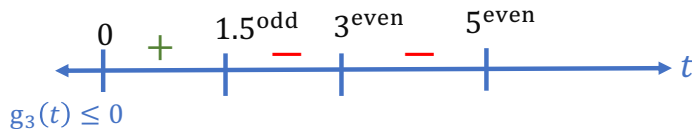
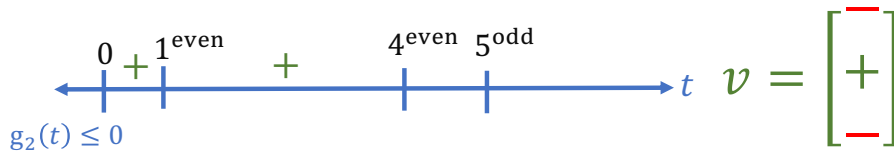


# Update fail

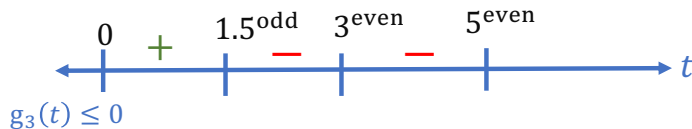
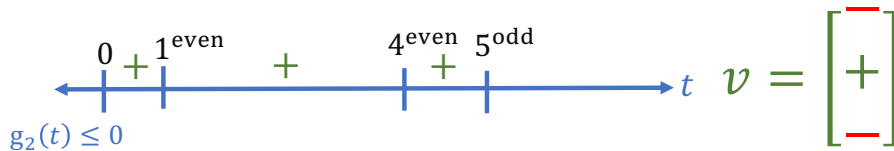




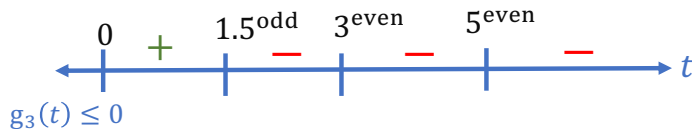
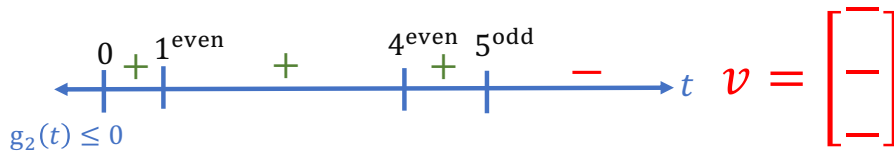
# Update fail



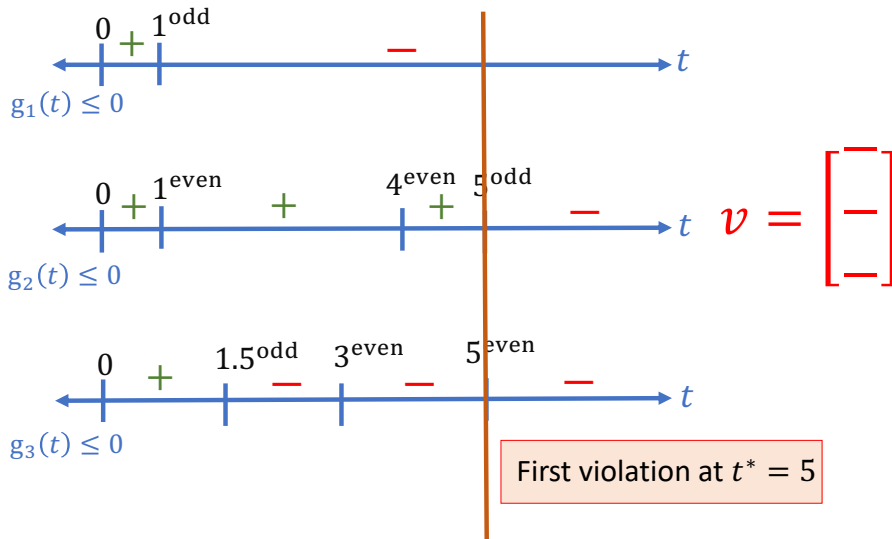
# Update fail



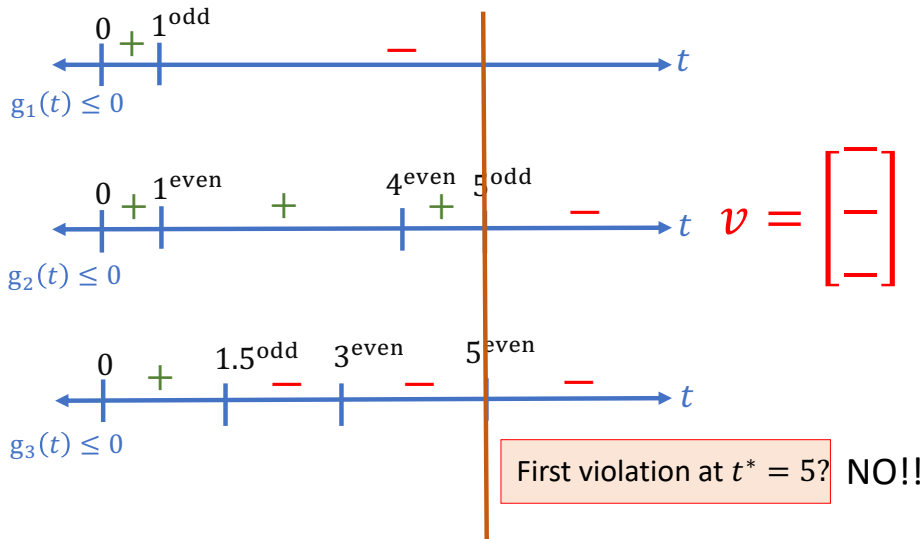
# Update fail



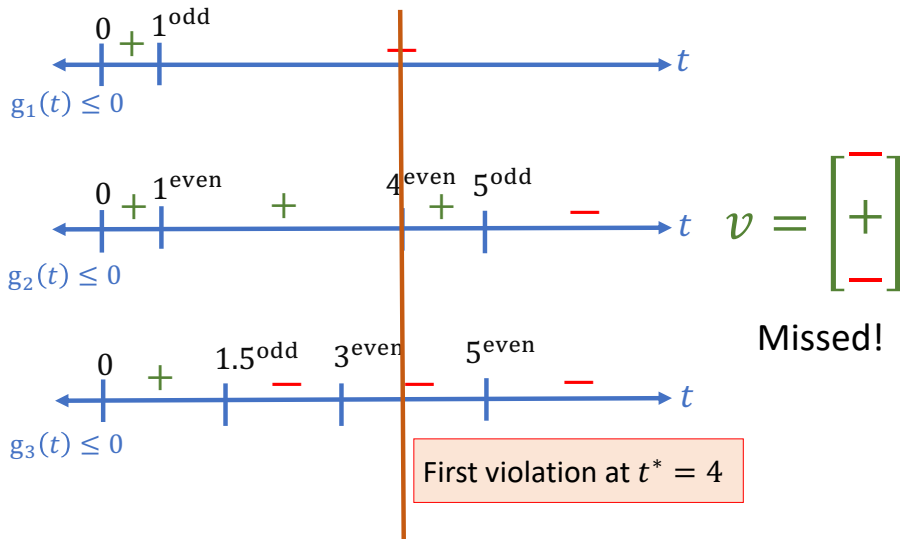
# Update fail



# Update fail



# Update fail



# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. for  $i, \dots, n$  do
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. end for
6. sort roots in ascending order
7. choose  $c$  less than all roots
8. calculate  $v$ , vector of signs of polynomials at  $c$
9. for each root do
10.   **update  $v$  at current root**
11.   **if all entries of  $v$  are '-' then**
12.     return current root
13.   end if
14. end for
15. return FALSE

**Warning!** This algorithm has flaws that could result in a **catastrophic failure**

## Catastrophic failure

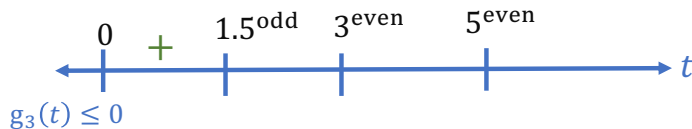
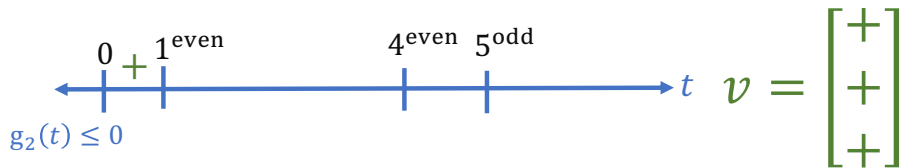
The update strategy of  $v$  is flawed, and can result in a violation not being detected

When a polynomial is positive around a root with even multiplicity, a violation will not be detected

## Fix

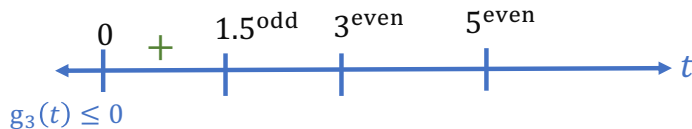
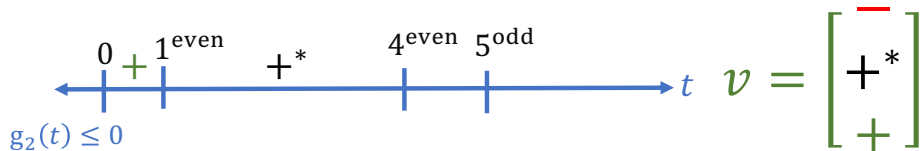
Introduce  $+$  to fix the update process

## Update fix

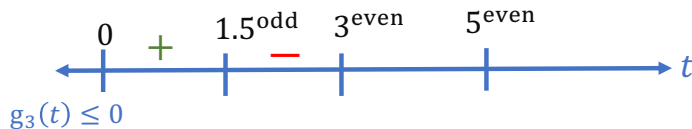
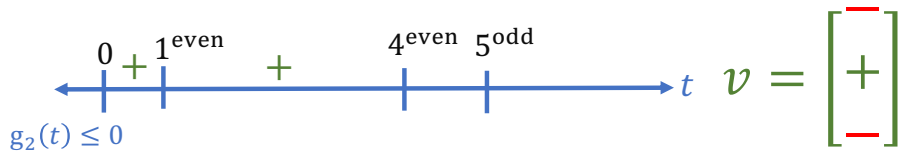




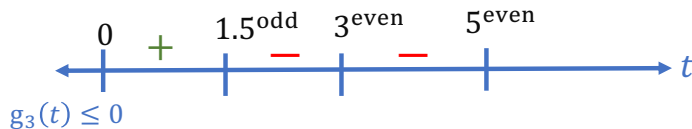
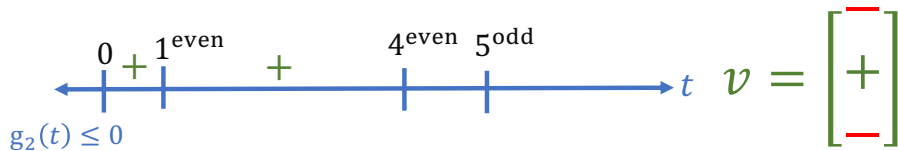
# Update fix



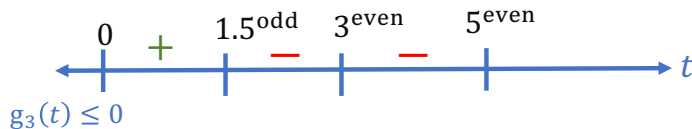
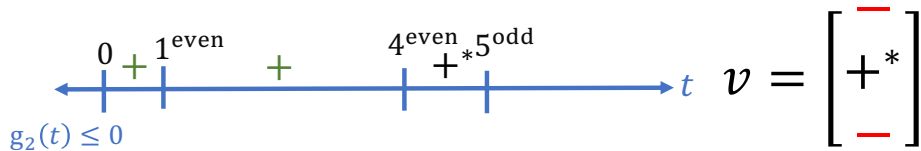
## Update fix



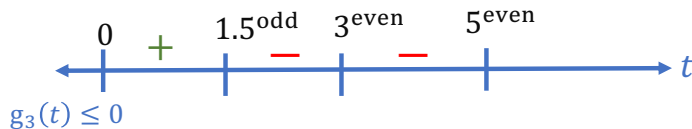
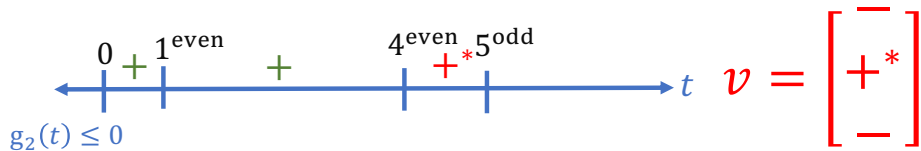
## Update fix



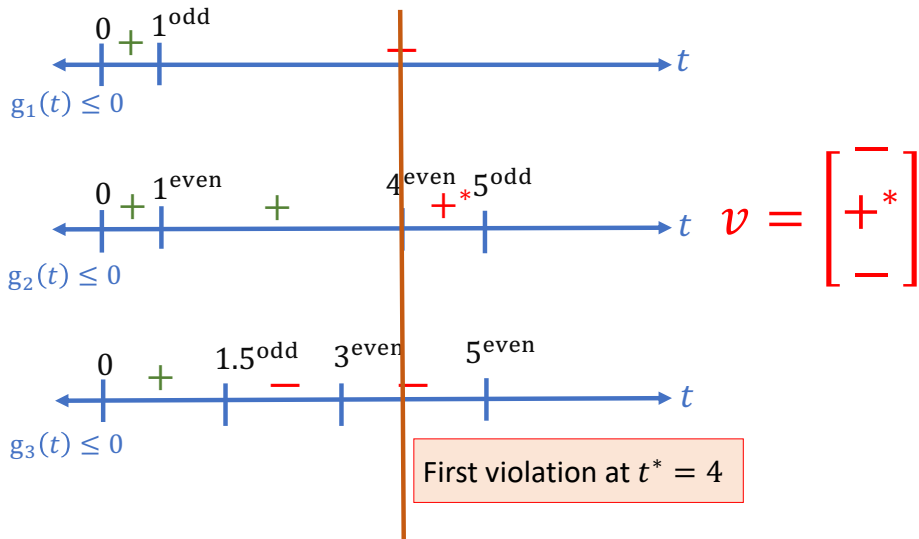
# Update fix



## Update fix



# Update fix



# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. **for**  $i, \dots, n$  **do**
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. **end for**
6. sort roots in ascending order
7. choose  $c$  less than all roots
8. calculate  $v$ , vector of signs of polynomials at  $c$
9. **for** each root **do**
10.   **update**  $v$  **at current root (with change)**
11.   **if all entries of**  $v$  **are either '-' or '+'** **then**
12.     **return** current root
13.   **end if**
14. **end for**
15. **return** FALSE

## Theorem: PolySafe works

- If the Polysafe algorithm returns False, then there is no violation
- If the Polysafe algorithm returns a number  $t^*$ , then the first instance of a violation occurs at time  $t^*$

# Formal analysis of PolySafe

Input:  $G = \{t \mid g_i(t) \leq 0, \forall i = 1, \dots, n\}$

Output:  $t^*$  time of first violation or FALSE

1. discard zero polynomials, update  $G$
2. **for**  $i, \dots, n$  **do**
3.   calculate roots with multiplicities of  $g_i$
4.   discard non-real roots, negative roots
5. **end for**
6. sort roots in ascending order
7. choose  $c$  less than all roots
8. calculate  $v$ , vector of signs of polynomials at  $c$
9. **for** each root **do**
10.   **update**  $v$  **at current root (with change)**
11.   **if all entries of  $v$  are either '-' or '+'**
12.     **then**
13.       return current root
14.   **end if**
15. **end for**
16. return FALSE

## Theorem: PolySafe works

- If the Polysafe algorithm returns False, then there is no violation
- If the Polysafe algorithm returns a number  $t^*$ , then the first instance of a violation occurs at time  $t^*$

```
PolySafe_works: THEOREM
FORALL(p:Polynomial,
      G:Ob_T,
      rl:(sorted_root_list?):
  (PolySafe_full(G)(p) < -1
   IMPLIES
    NOT Violation?(G)(p))
AND
  (PolySafe_full(G)(p) >= 0
   IMPLIES Violation?(G)(p))
```



# Formal analysis of PolySafe (conclusions)

---

```
PolySafe_works: THEOREM
FORALL(p:Polynomial,
  G:Ob_T,
  rl:(sorted_root_list?):
(PolySafe_full(G)(p) < -1
  IMPLIES
  NOT Violation?(G)(p))
AND
(PolySafe_full(G)(p) >= 0
  IMPLIES Violation?(G)(p))
```

- Summary
  - 438 proofs
  - Polynomials as lists (with standard form)
  - Polynomials around roots
  - First violation occurring at a root

- Outcomes
  - Identified missing requirements of PolySafe
  - Identified and fixed root behavior
  - Fully executable PolySafe in PVS environment
- Further considerations
  - Root computation and approximation
  - Resolving a conflict

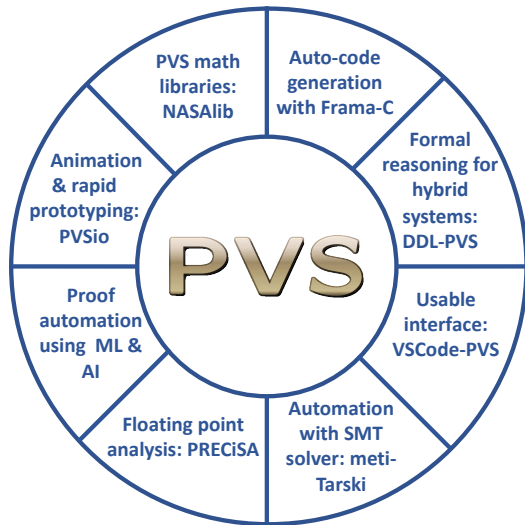
# Conclusions, challenges, future Work

## Conclusion:

- Cows
- Formal methods
- PVS, NASALib
- Formal reasoning in a polynomial airspace

## Challenges, future work :

- Expanding NASALib
- Formal verification for increasing autonomous complex systems
- Proof automation, automated reasoning
- Animation and rapid prototyping
- Code generation



Thanks for listening! Q & C?