Verifying Safety Properties of Hybrid Systems.

Sriram Sankaranarayanan
University of Colorado, Boulder, CO.

October 22, 2010.

Talk Outline

1. Formal Verification
2. Hybrid Systems
3. Invariant Synthesis
4. Algebraic Invariants
5. Results

Proving Programs Correct

Example: Compute $\lceil \sqrt{n} \rceil$, for $n \geq 0$.

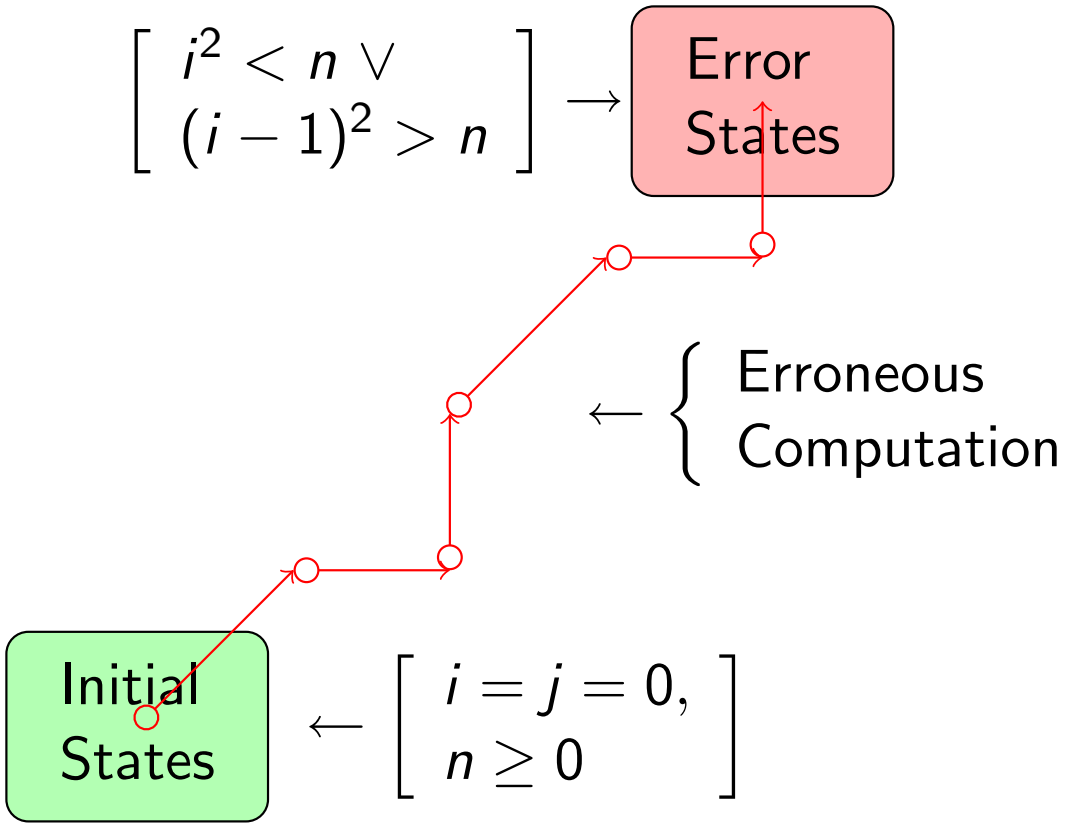
```
int computeSqrt ( int n )
      @Pre: n ≥ 0
1:    int i, j = (0, 0);
2:    while ( j ≤ n ) {
3:          i := i + 1;
4:          j := j + 2 * i − 1;
5:    }
      @Post: i^2 ≥ n ∧ (i − 1)^2 ≤ n
```

Verification Problem: Is this program "correct"?

What is Verification?

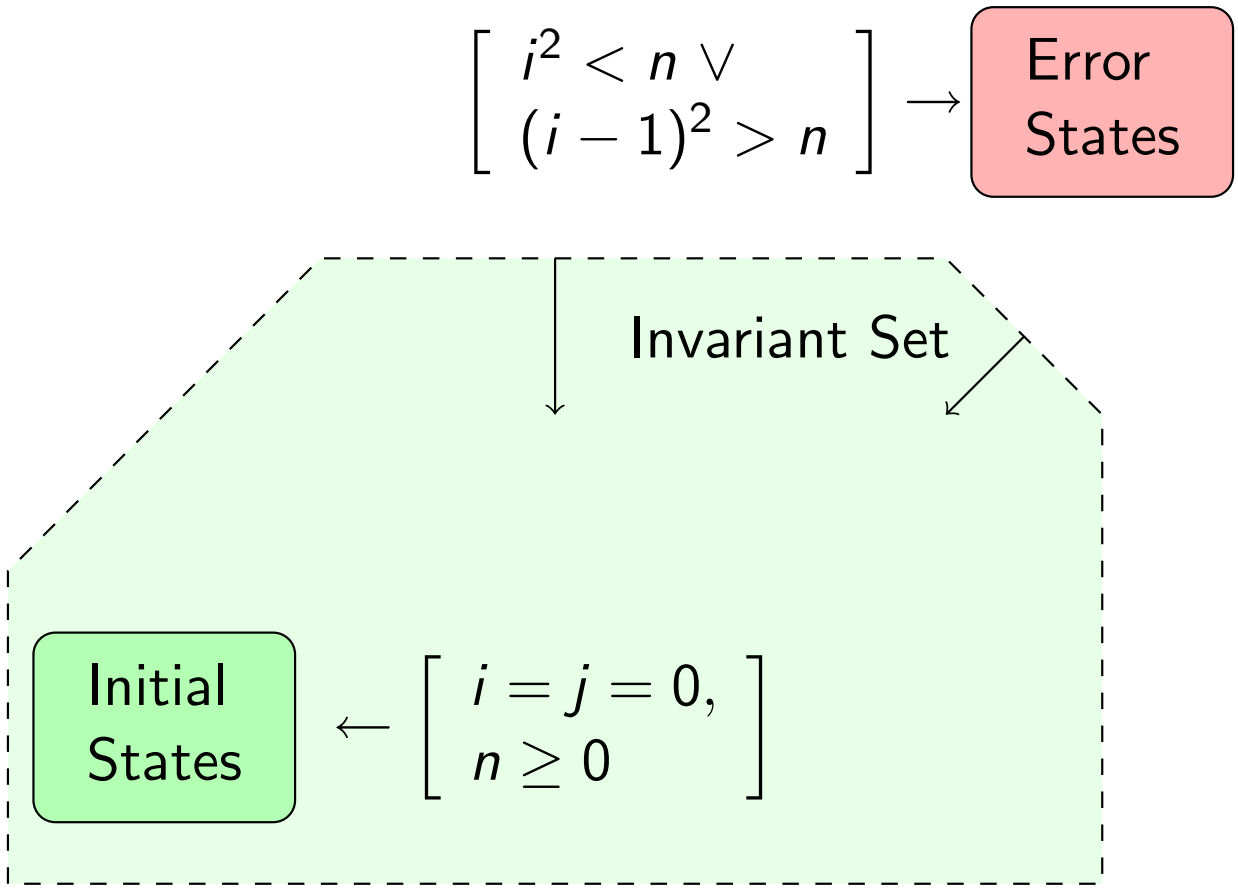
Finding an Error

Discover input n that violates property.

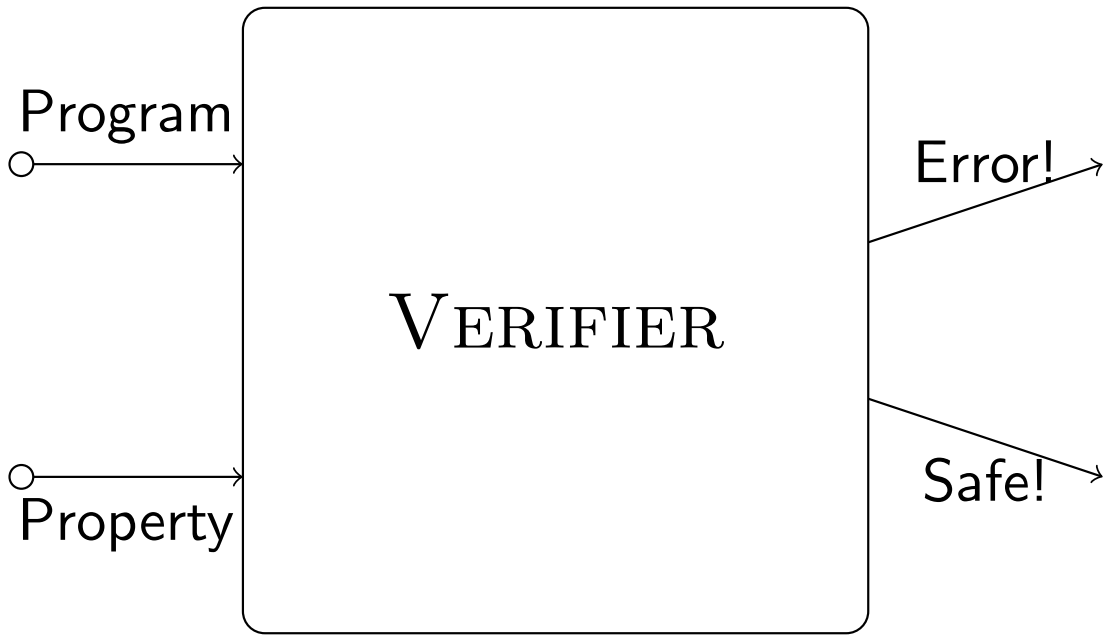


Proving Safety

Start inside invariant set $\Rightarrow$ remain in invariant set.



The Program Verifier



Undecidability

Theorem

There is no program P, that takes a program Q and a *non-trivial property* φ as inputs, and decides if every run of P satisfies φ.

Cf. Turing, Alan. M. (1936). "*On Computable Numbers, with an Application to the Entscheidungsproblem*". Proceedings of the London Mathematical Society. 2 42: 23065. 193637.

Verification Techniques

Nearly 40+ years of research on program verification.

- Model Checking: Best effort search for a counter example.
- Static Analysis: Best effort search for a property proof.
- Semi-Formal Methods: *Ad-hoc* approaches inspired by above.

Verification: Achievements

- Model Checking: Popular in hardware verification.
  - Arithmetic-Logic Unit Verification (Intel, AMD).
  - Cache Coherence Protocols.
  - Symbolic techniques (BDDs, SAT solvers): $\geq 10^{500}$ states.

Verification: Achievements

- Model Checking: Popular in hardware verification.
  - Arithmetic-Logic Unit Verification (Intel, AMD).
  - Cache Coherence Protocols.
  - Symbolic techniques (BDDs, SAT solvers): $\geq 10^{500}$ states.
- Static Analysis: Proving correctness of programs.
  - Astreé project: Runtime error freedom in Airbus A380 command & control systems.
  - Commercial products:
    - Absint: Timing verification in real-time systems.
    - Polyspace: Verification of control systems (Mathworks).

Verification: Achievements

- Model Checking: Popular in hardware verification.
  - Arithmetic-Logic Unit Verification (Intel, AMD).
  - Cache Coherence Protocols.
  - Symbolic techniques (BDDs, SAT solvers): $\geq 10^{500}$ states.
- Static Analysis: Proving correctness of programs.
  - Astreé project: Runtime error freedom in Airbus A380 command & control systems.
  - Commercial products:
    - Absint: Timing verification in real-time systems.
    - Polyspace: Verification of control systems (Mathworks).
- Semi-Formal Tools: Finding bugs in programs.
  - CoVerity: Gaining wide usage in software industry.
  - Varvel (NEC), Slam (Microsoft), Findbugs (Google),...

Proving Programs Correct

Example: Compute $\lceil \sqrt{n} \rceil$, for $n \geq 0$.

```
int computeSqrt ( int n )
    @Pre: n ≥ 0
1:  int i, j = (0, 0);
2:  while ( j ≤ n ) {
3:      i := i + 1;
4:      j := j + 2 * i − 1;
5:  }
    @Post: i^2 ≥ n ∧ (i − 1)^2 ≤ n
```

Verification Problem: Is this program "correct"?

Yes: Automatically synthesized invariant.

$$j = i^2,\ n \geq 0,\ j \leq n+1,\ j \leq n-2i-1\,.$$

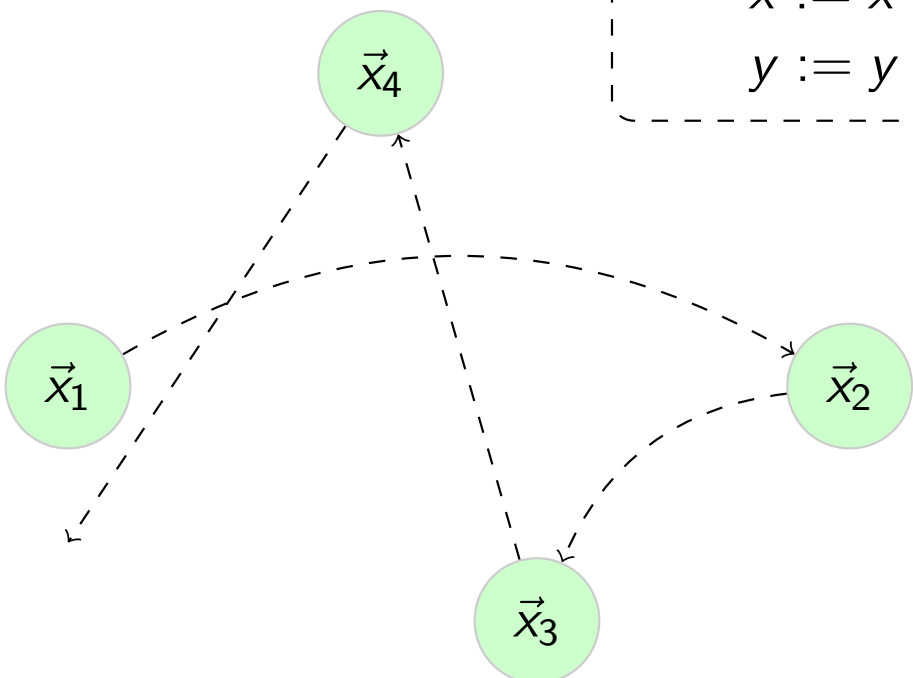
Challenge: Verify *Hybrid Systems*.

Dynamical Systems

Discrete dynamical systems: defined by maps.

$$\vec{x}(n+1) = F(n, \vec{x}(n))\,.$$

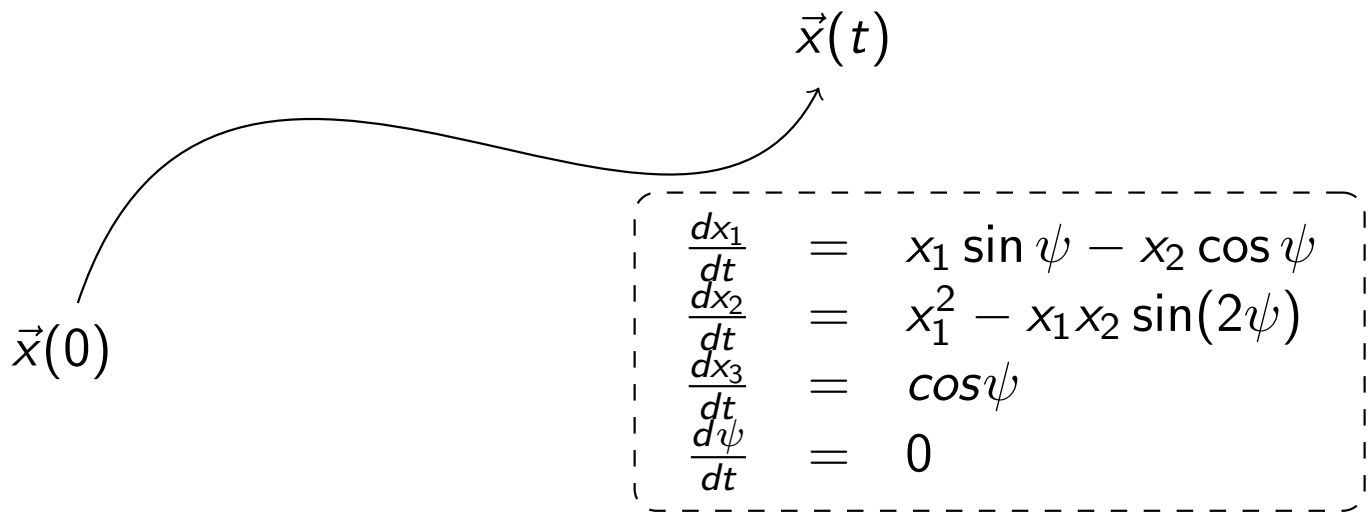
$$\begin{array}{c} \text{while}(x^2 + y^2 \leq 3) \\ x := x + y^{\frac{1}{2}} \\ y := y - x^{\frac{1}{2}} \end{array}$$



Dynamical Systems

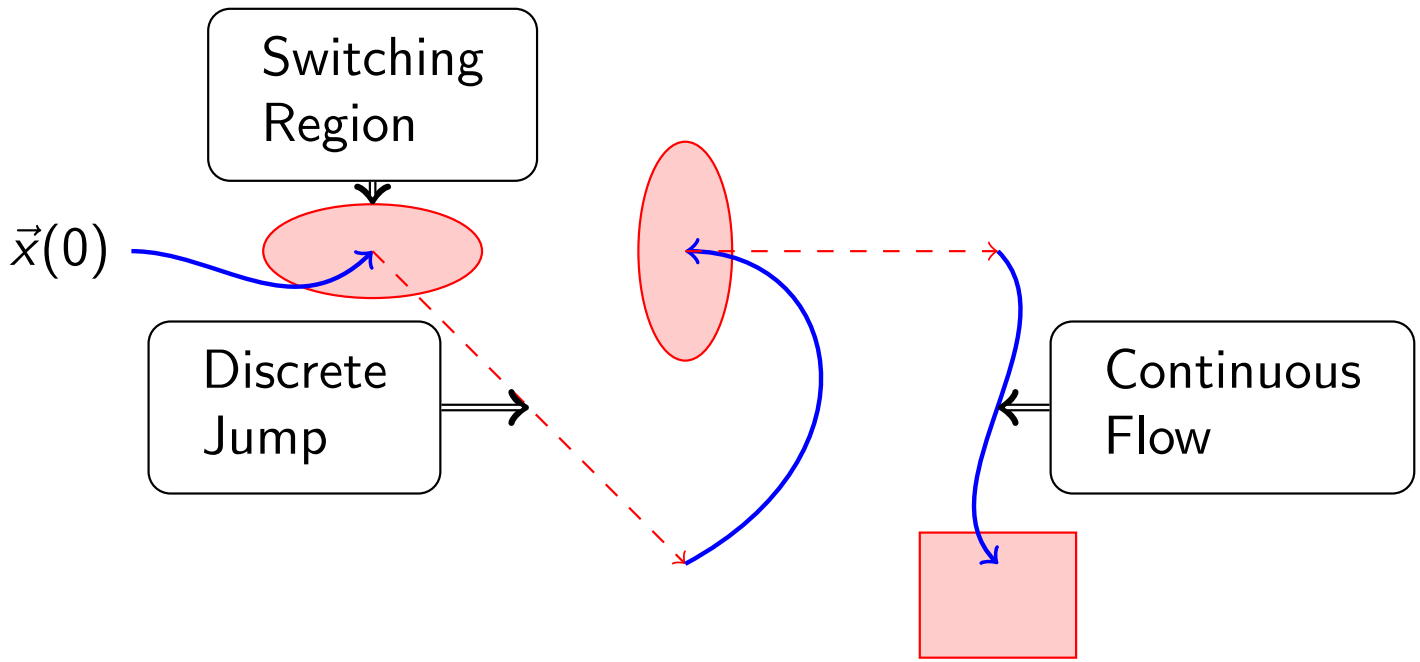
Continuous dynamical systems: defined by flows.

$$\frac{d\vec{x}}{dt} = F(t, \vec{x})\,.$$

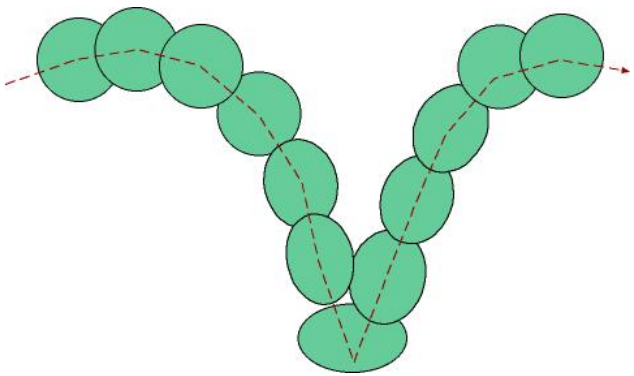


Hybrid Trajectories

1. Flows + Discrete jumps.
2. *Multi-Modal*: Dynamics depend on the mode.



Example # 1: Bouncing Ball

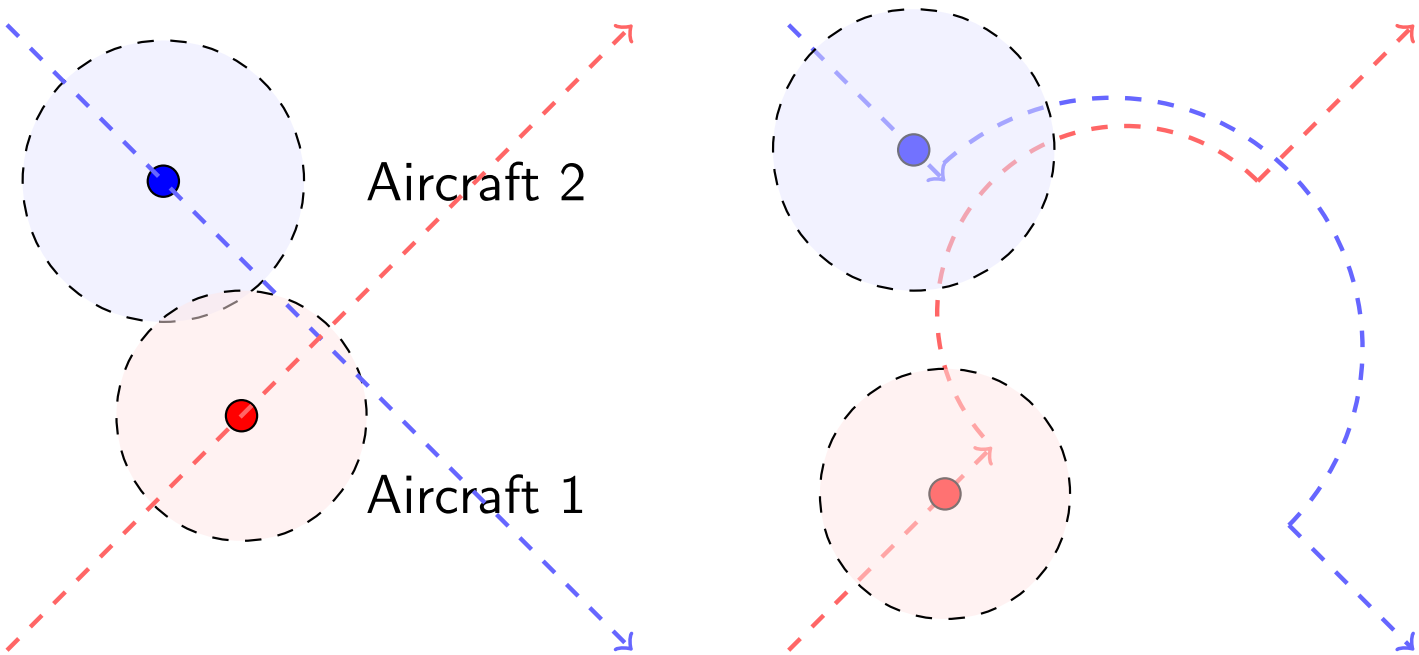


$$\text{if}(y = 0 \ \wedge \ v_y < 0)$$
$$\text{do} : v_y := -\tfrac{1}{2} v_y$$

$$\begin{aligned} \frac{dx}{dt} &= v_x \\ \frac{dy}{dt} &= v_y \\ \frac{dv_x}{dt} &= 0 \\ \frac{dv_y}{dt} &= -9.8 \end{aligned}$$

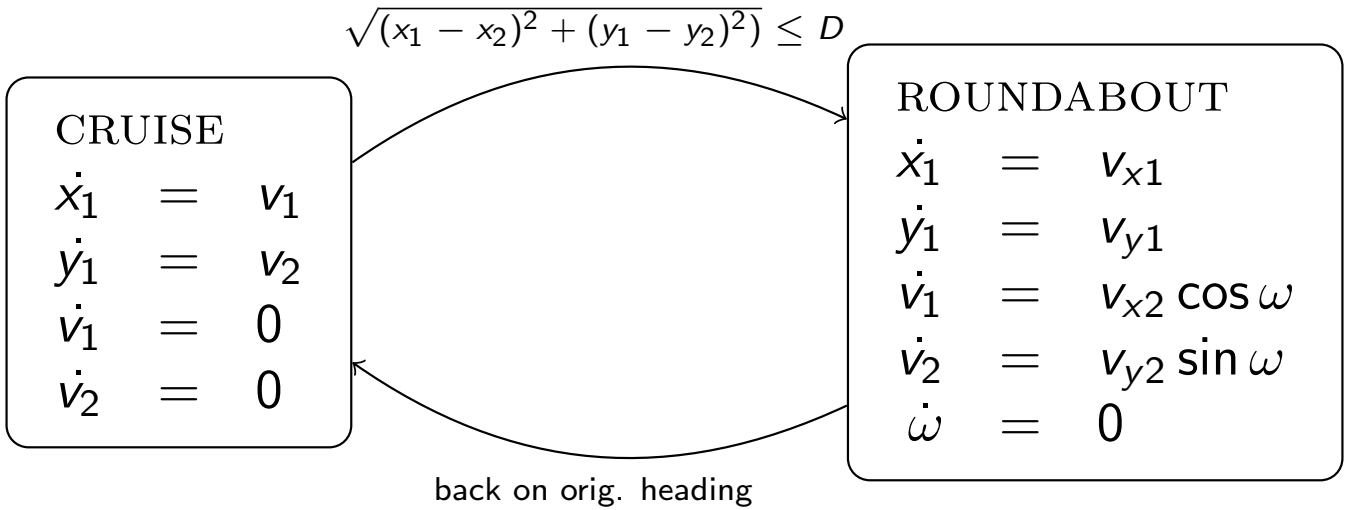
Example # 2: Conflict Resolution Maneuvers

Conflict resolution protocol. [Tomlin et al.'98]

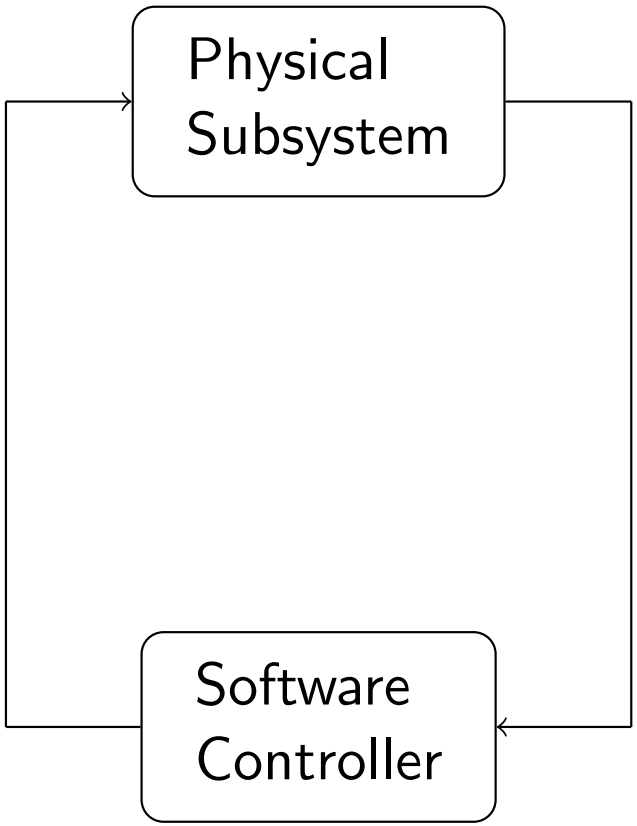


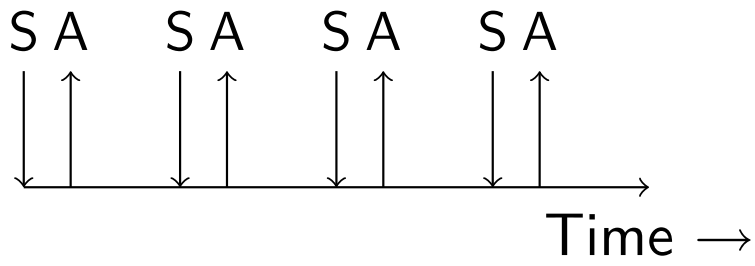
Collision Avoidance Model

Hybrid automaton for each aircraft:



Software Enabled Control

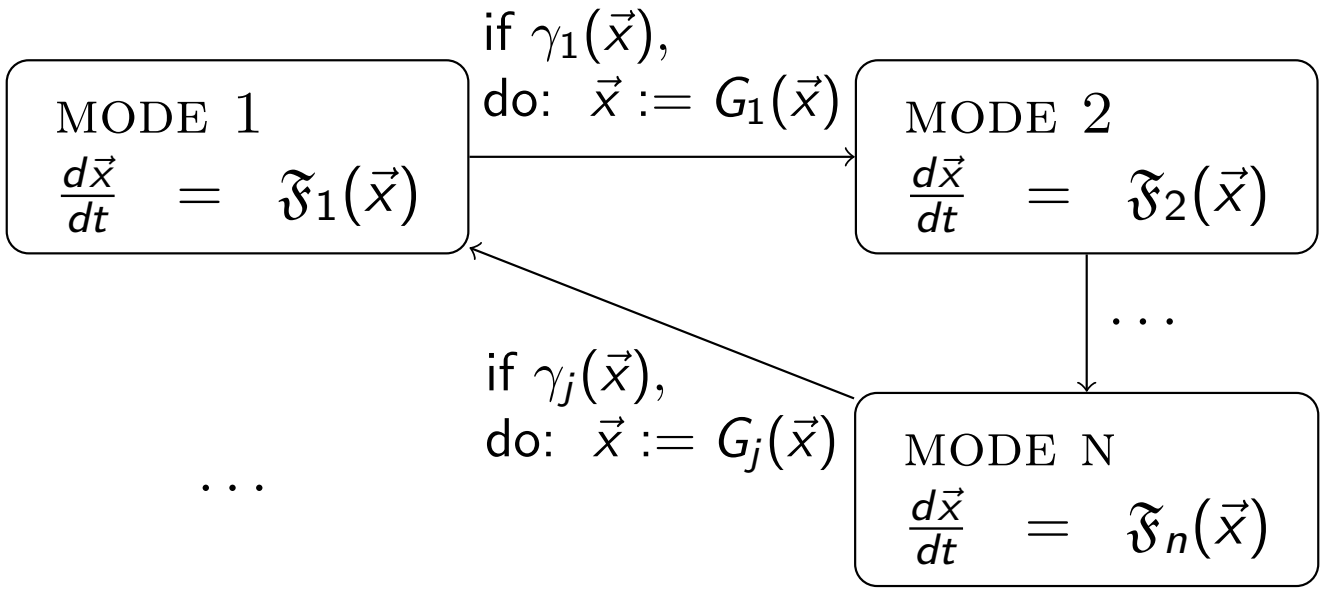




S: Sense, A: Actuate

Hybrid Automaton

[Alur et al.'96; Sastry et al. '98]

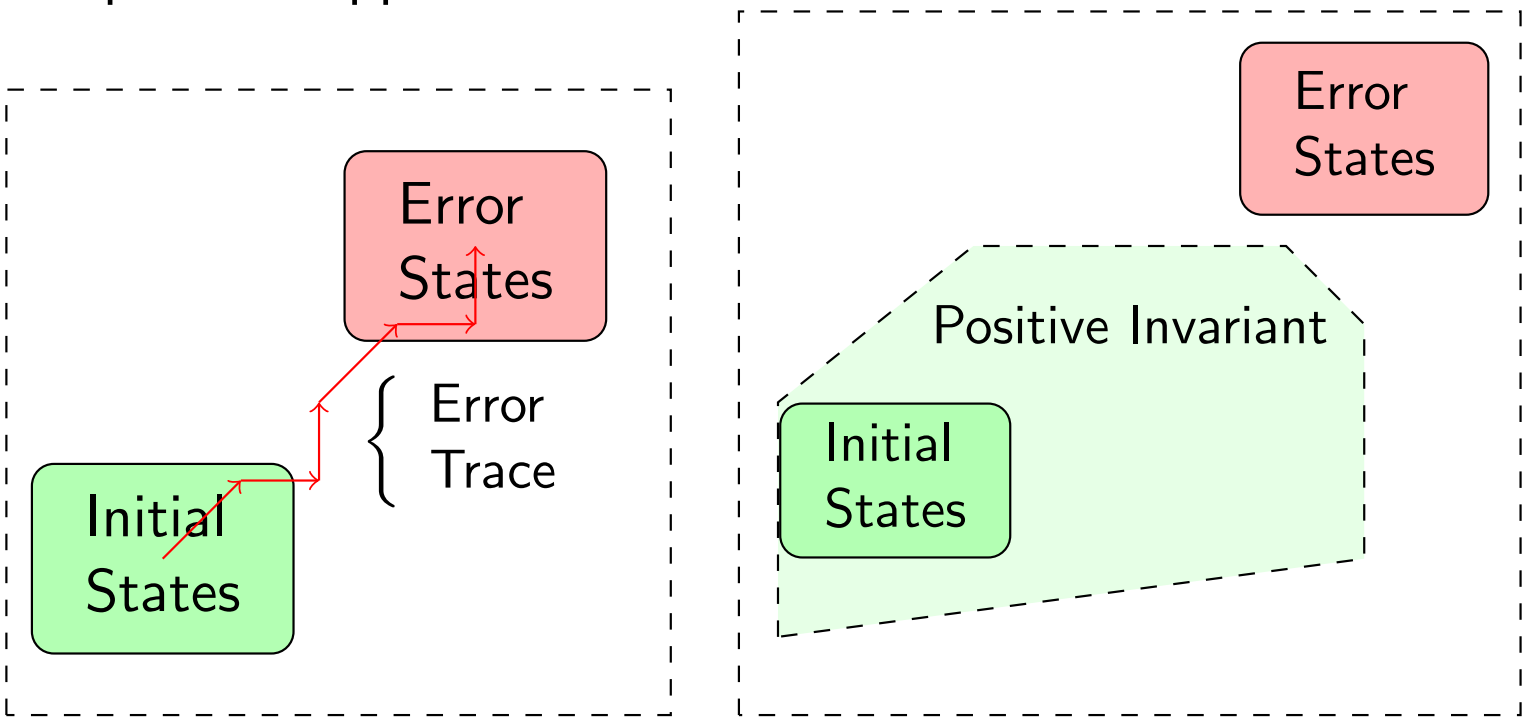


- Finite set of *modes*. $Q : \{q_1, \ldots, q_m\}$
- Continuous state variables. $\vec{x} : (x_1, \ldots, x_n)$.
- Dynamics for each mode.
- Discrete Transitions between modes.

Verification of Hybrid Systems.

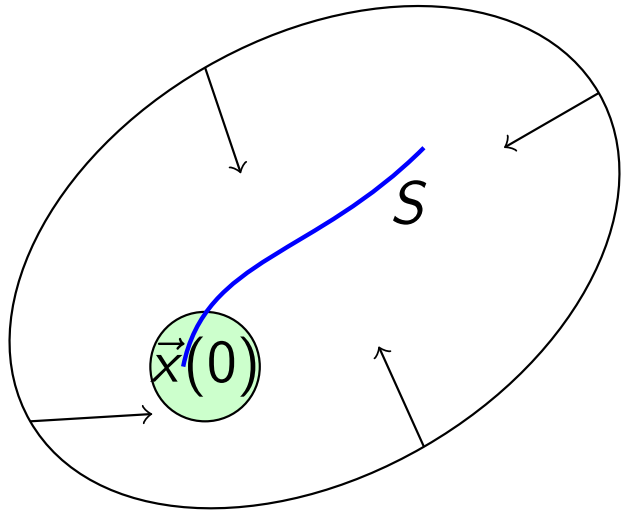
Verification of Hybrid Systems.

Two possible approaches.



Rest of this talk: Generating Positive Invariants.

Positive Invariant Set

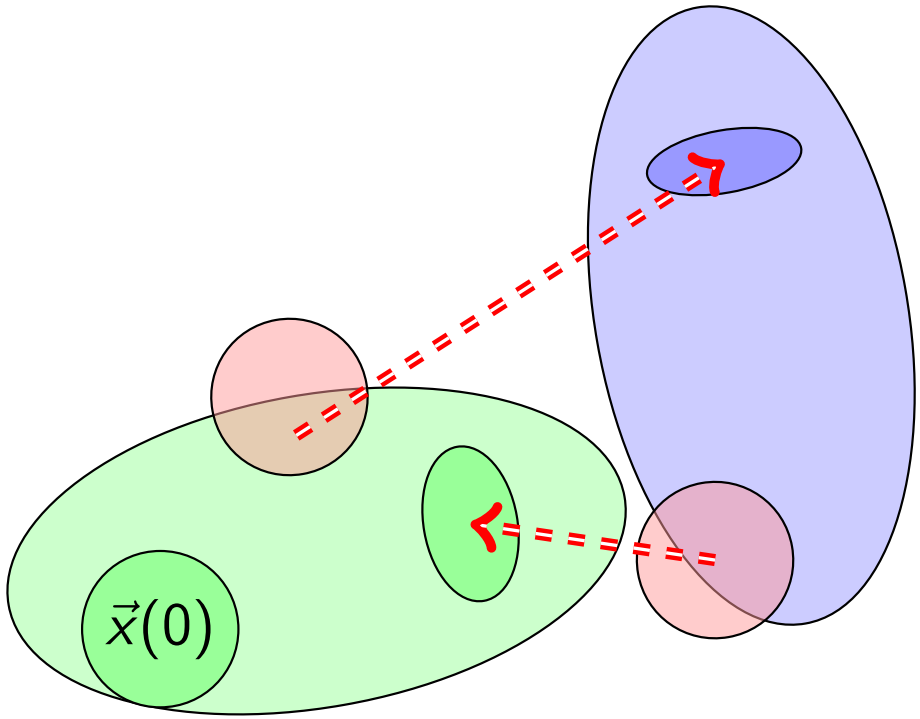


Set S is positive invariant for flow φ iff

$$\forall\ \vec{x}(0) \in S,\ \varphi(\vec{x}(0), t) \in S\,.$$

Start inside set $S \Rightarrow$ flow remains in S.

Positive Invariants for Hybrid Systems



- Disjoint union of sets.
- "Preserved" by discrete transitions.

Positive Invariant Computation.

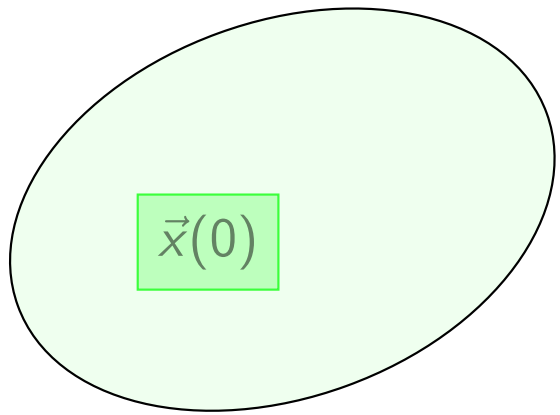
[Henzinger et al.'96,...]

$\vec{x}(0)$

- Start with Initial States.

Positive Invariant Computation.

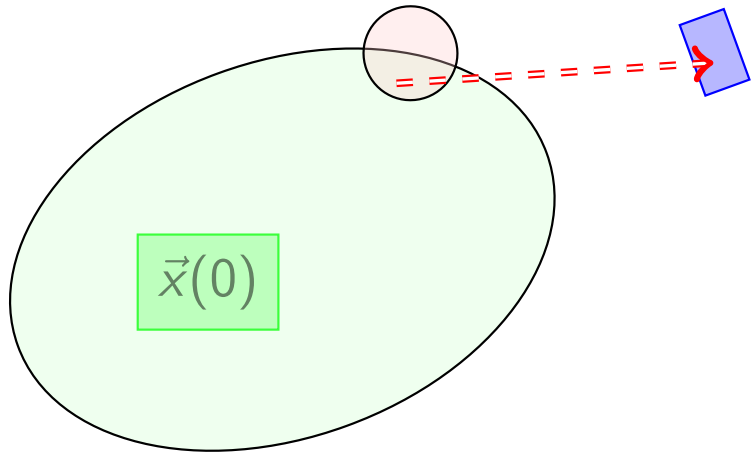
[Henzinger et al.'96,...]



- Start with Initial States.
- Positive Invariant computation for differential equations.

Positive Invariant Computation.

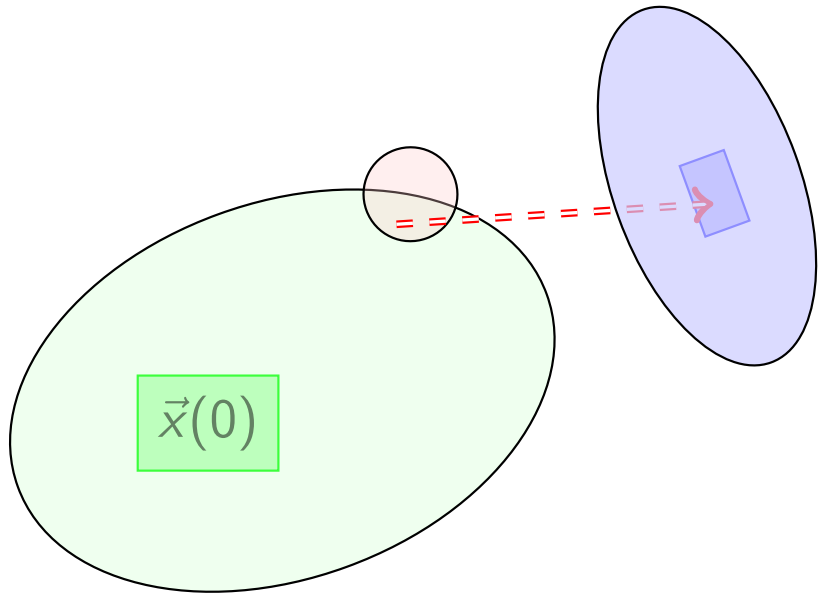
[Henzinger et al.'96,...]



- Start with Initial States.
- Positive Invariant computation for differential equations.
- Computing images across discrete transitions.

Positive Invariant Computation.

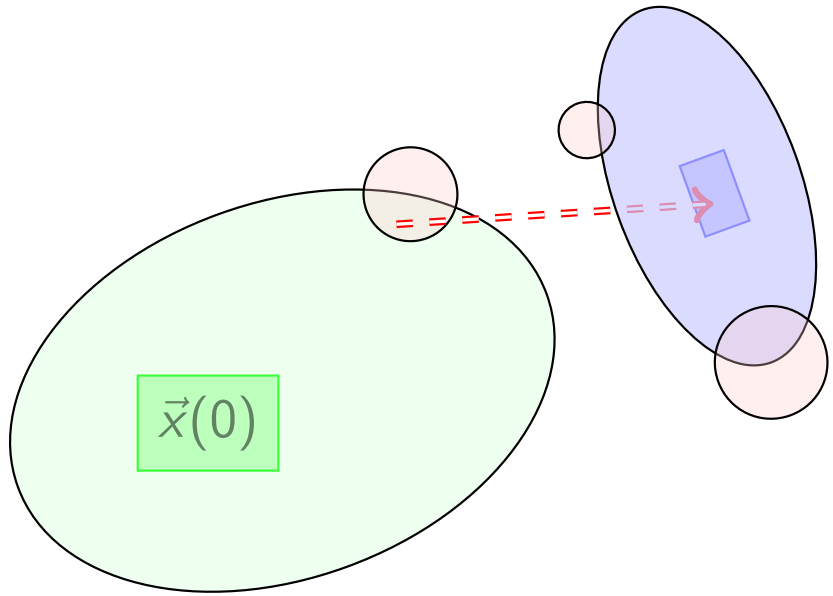
[Henzinger et al.'96,...]



- Start with Initial States.
- Positive Invariant computation for differential equations.
- Computing images across discrete transitions.
- Iterate until convergence.

Positive Invariant Computation.

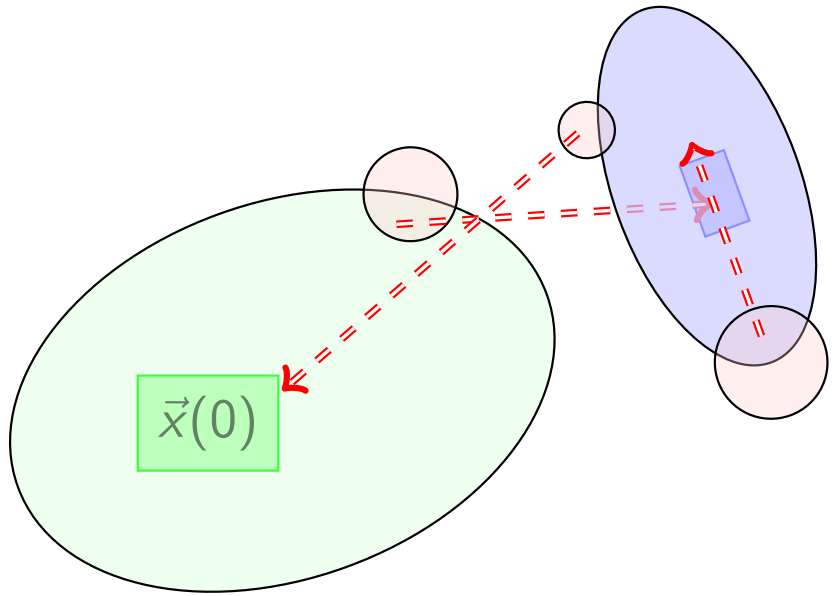
[Henzinger et al.'96,...]



- Start with Initial States.
- Positive Invariant computation for differential equations.
- Computing images across discrete transitions.
- Iterate until convergence.

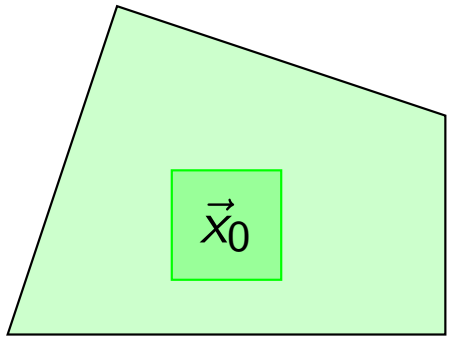
Positive Invariant Computation.

[Henzinger et al.'96,...]



- Start with Initial States.
- Positive Invariant computation for differential equations.
- Computing images across discrete transitions.
- Iterate until convergence.

Polyhedral Invariant Generation.



- Polyhedra [Halbwachs'94, Henzinger et al.'96, Sank.et al.'06]
- Zonotopes. [Girard'05]
- (Linear) Support Functions. [Girard'09,'10]
- Templatized Polyhedra. [Sank.et al.'08,'09]
- Orthogonal Polyhedra. [Asarin et al.'02]

- Tools: PHaver, Mattisse, TimePass,...

Q: What about non-linear invariants and non-linear dynamics?

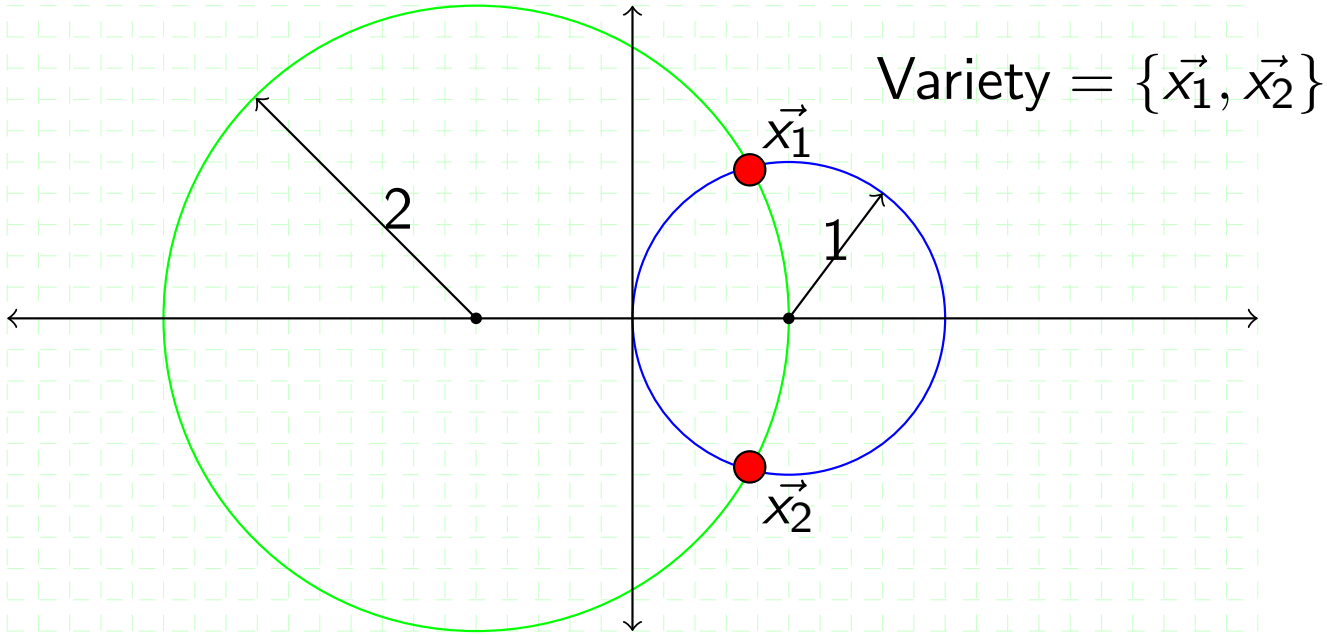
Algebraic Techniques for Non-Linear Invariants.

- Generate positive invariants that are *algebraic varieties*.

$$x_1^2 - 2x_3 - x_2^2 = 5 \ \wedge \ x_1 x_2 = x_3$$

- Using techniques from *computational algebraic geometry*.

Algebraic Varieties



Zeros of a set of multi-variate polynomials.

$$V : \{\vec{x} \in \mathcal{R}^n \mid p_1(\vec{x}) = 0,\ p_2(\vec{x}) = 0,\ \ldots,\ p_m(\vec{x}) = 0\}$$

Ideals

Set of polynomials I closed under

1. Addition.

$$p_1, p_2 \in I \;\Rightarrow\; p_1 + p_2 \in I$$

2. Multiplication by any other polynomial in the ring.

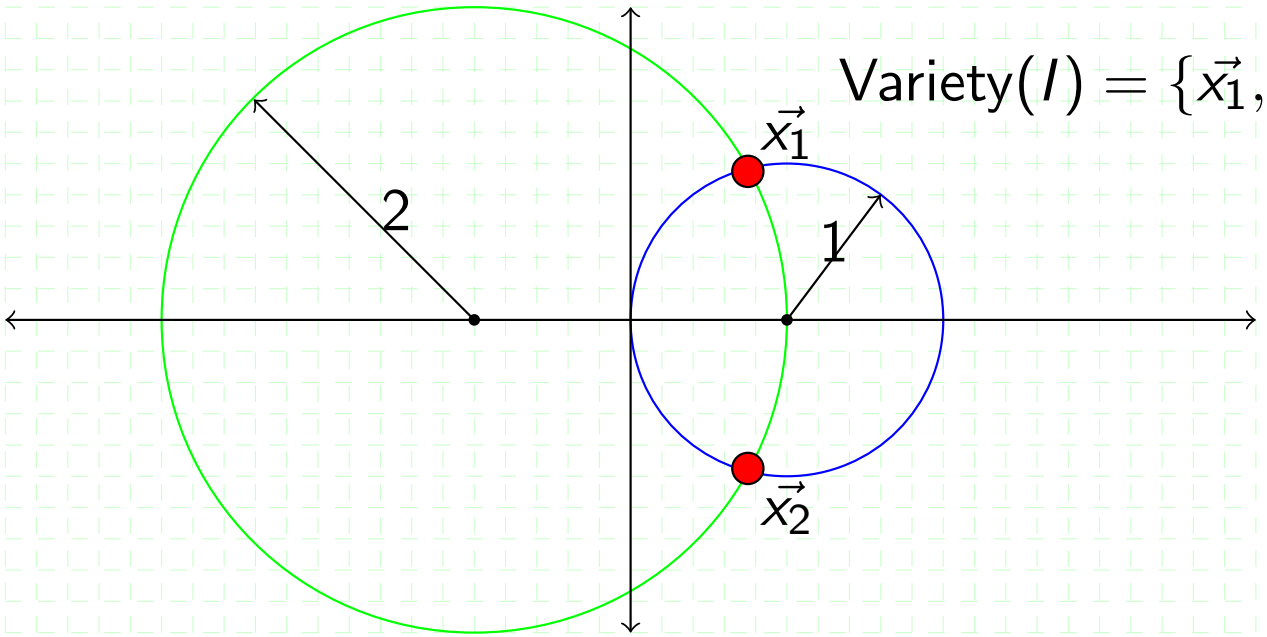
$$p \in I,\; g \in \mathcal{R}[\vec{x}],\; \Rightarrow\; g \cdot p \in I$$

Ideal Generated by Set: $\langle\langle p_1, \ldots, p_m \rangle\rangle$.

Background: Ideals and Varieties.

$$\begin{aligned} \text{IdealOf}(V) &: \{p \in K[\vec{x}] \mid p \text{ vanishes everywhere on } V\} \\ \text{VarietyOf}(I) &: \{\vec{x} \in K^n \mid \forall\ p \in I,\ p(\vec{x}) = 0\} \end{aligned}$$

Example: $I = \left\langle\left\langle \{\underbrace{(x+1)^2 + y^2 - 4}_{p_1}, \underbrace{(x-1)^2 + y^2 - 1}_{p_2}\} \right\rangle\right\rangle.$



Hilbert's Nullstellensatz

For any ideal I and polynomial p,

$$\underbrace{\text{Variety}(I) \subseteq \{\vec{x} \mid p(\vec{x}) = 0\}}$$

(i.e, p is a consequence of the equations in I)

Hilbert's Nullstellensatz

For any ideal I and polynomial p,

$$\underbrace{\text{Variety}(I) \subseteq \{\vec{x} \mid p(\vec{x}) = 0\}}$$

(i.e, p is a consequence of the equations in I)

if and only if there is some power p^m of p, such that

$$p^m \in I$$

Computing with Ideals

- Hilbert Finite Basis Theorem:

  Any ideal in $K[\vec{x}]$ for field K is finitely generated.

Computing with Ideals

- Hilbert Finite Basis Theorem:

  Any ideal in $K[\vec{x}]$ for field K is finitely generated.

- *Groebner Basis:* Finite basis with nice properties.
  [Buchberger'60]

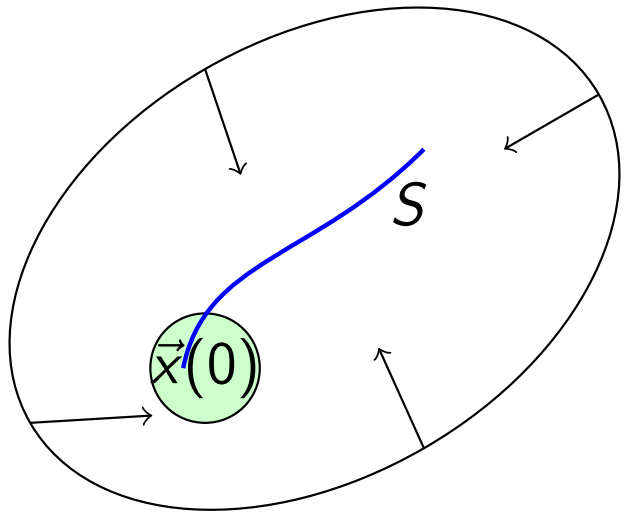
- Computing using Varieties.
  1. Ideal Membership (subsumption of varieties)
  2. Ideal Intersection (union of varieties)
  3. Ideal Union (intersection of varieties)
  4. Ideal Inclusion (inclusion of varieties)
  5. Image under a Map
  6. *Syzygy Modules*
  7. ...

Positive Algebraic Invariants.

Invariant synthesis for flows.

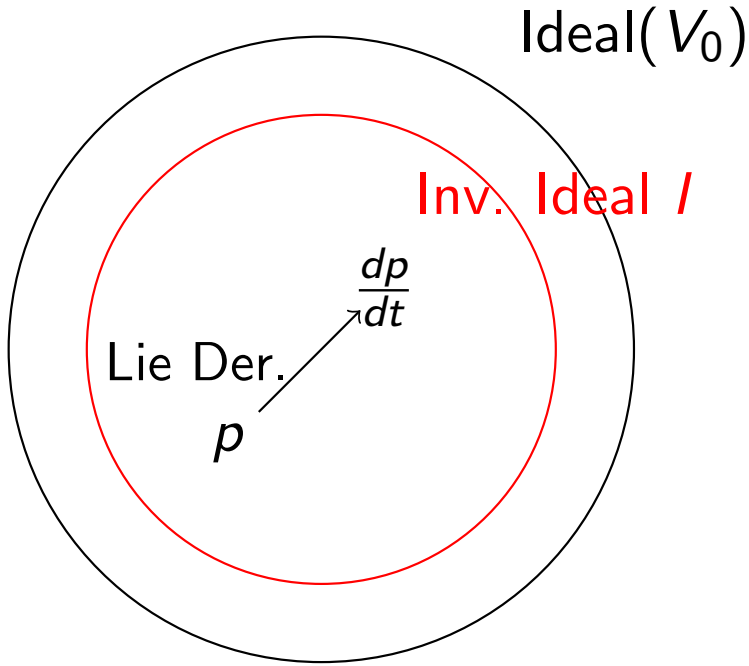
- Inputs:
  (a) Polynomial vector field: $\mathfrak{F}$.
  (b) Initial Set (variety): V_0.
- Problem: Generate positive invariant algebraic variety.
- "Smaller" variety is more desirable/useful.

Positive Invariant Variety



1. Contain initial variety
2. Vector field must lie on the tangent space.

"Positive Invariant" Ideal.



1. Contained in initial ideal: Ideal(V_0).
2. Closed under Lie-Derivatives w.r.t vector field.

Problem: Given V_0 and field $\mathfrak{F}$, compute *invariant ideal*.

Example # 1: Mechanical System

$$\begin{aligned}
\frac{dp_1}{dt} &= -2q_1q_2^2, \\
\frac{dp_2}{dt} &= -2q_1^2q_2, \\
\frac{dq_1}{dt} &= 2p_1, \\
\frac{dq_2}{dt} &= 2p_2
\end{aligned}$$

Initial condition:

$$V_0 : \{\vec{x} : (p_1, q_1, p_2, q_2) \mid p_1^2 + p_2^2 - 4 = 0,\ q^2 - 1 = 0, q_2 = 0\}.$$

Positive Invariant: $p_1^2 + p_2^2 + q_1^2q_2^2 = 4$.

A More Complex Example

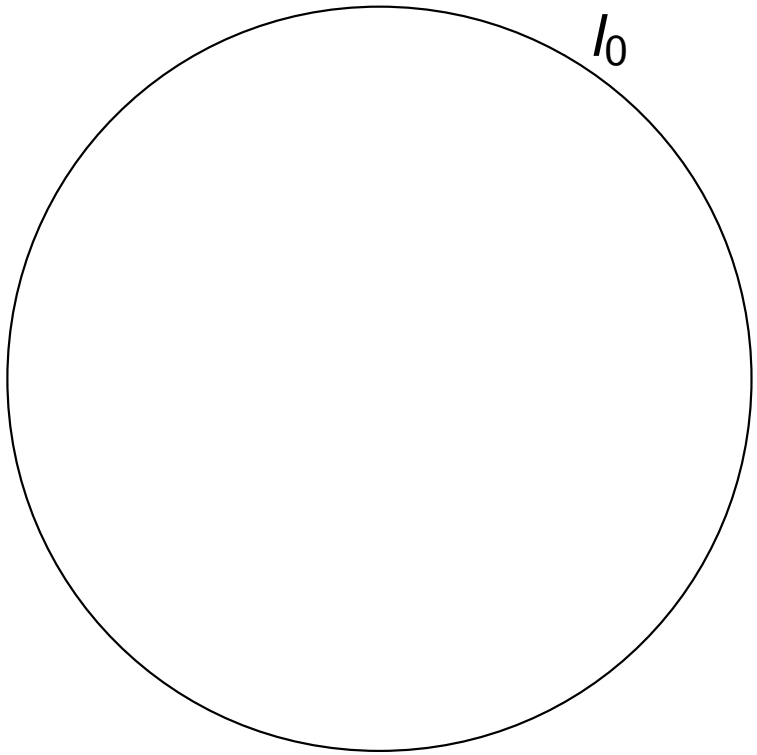
$$\begin{array}{rcl} \frac{dx_1}{dt} & = & v_1, \\ \frac{dx_2}{dt} & = & v_2, \\ \frac{dv_1}{dt} & = & -kx_1 - \frac{k}{5}(x_1 - x_2), \\ \frac{dv_2}{dt} & = & k(x_1 - x_2), \\ \frac{dk}{dt} & = & 0 \end{array}$$

$$V_0 : x_1 = x_2 = 0, v_1 = 1, v_2 = -1, k \in R$$

Invariant Ideal:

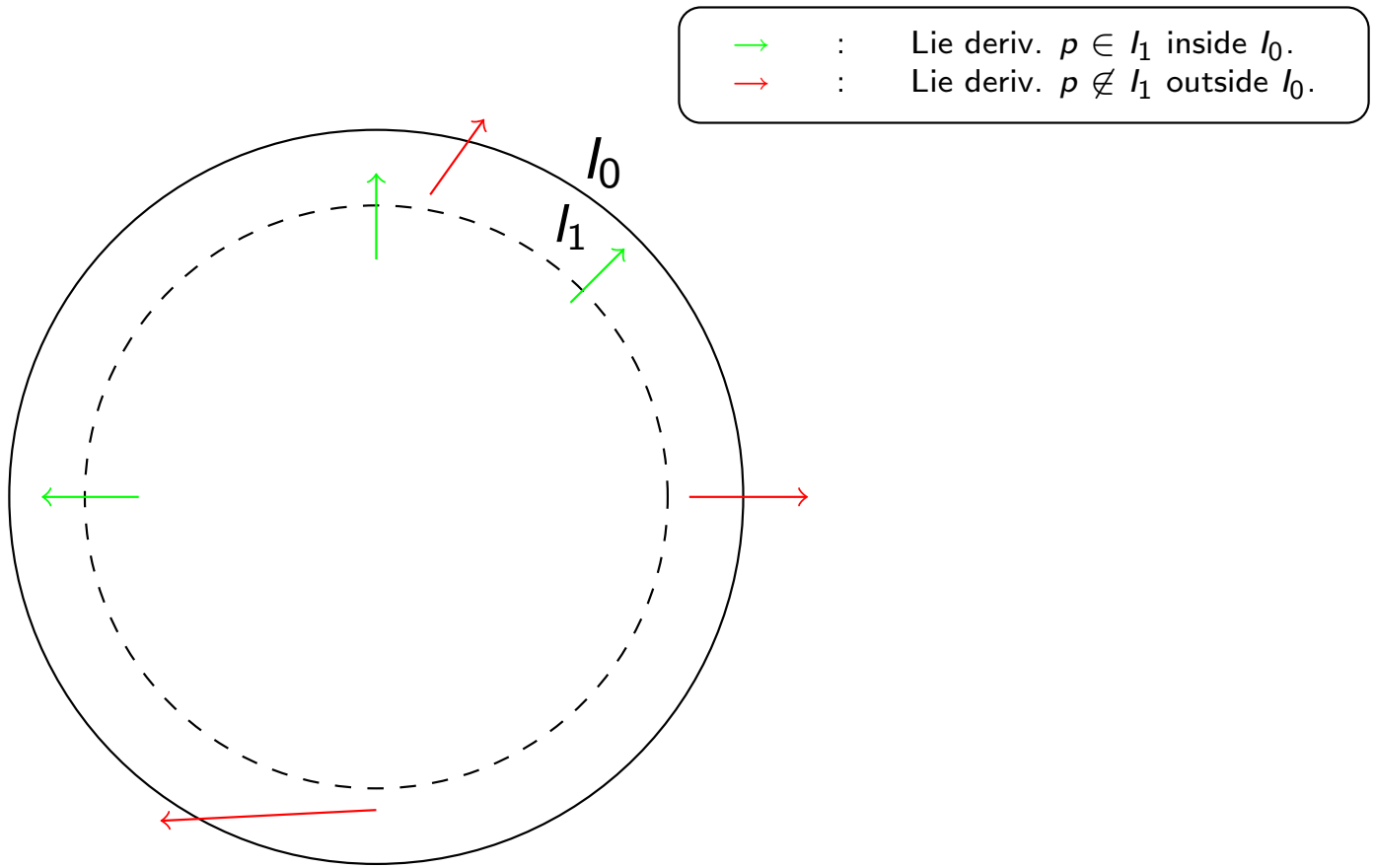
$$\begin{array}{lcl} p_1 & : & 576 + 1200v_1^2 + 625v_1^4 + 2880v_1v_2 + 3000v_1^3v_2 + \\ & & 528v_2^2 + 4150v_1^2v_2^2 + 1320v_1v_2^3 + 121v_2^4 \\ & & -1860kx_2^2 + 2750kv_1^2x_2^2 + 1600kv_1v_2x_2^2 \\ & & +710kv_2^2x_2^2 + 525k^2x_2^4 = 0, \\ p_2 & : & 240x_1 + 250v_1^2x_1 + 600v_1v_2x_1 + 110v_2^2x_1 \\ & & +396x_2 - 525v_1^2x_2 - 260v_1v_2x_2 - 131v_2^2x_2 - 105kx_2^3 = 0, \\ p_3 & : & 24 + 25v_1^2 + 60v_1v_2 + 11v_2^2 + 50kx_1x_2 + 5kx_2^2 = 0 \\ p_4 & & -21 + 25v_1^2 + 10v_1v_2 + 6v_2^2 + 25kx_1^2 + 5kx_2^2 = 0 \end{array}$$

Strategy



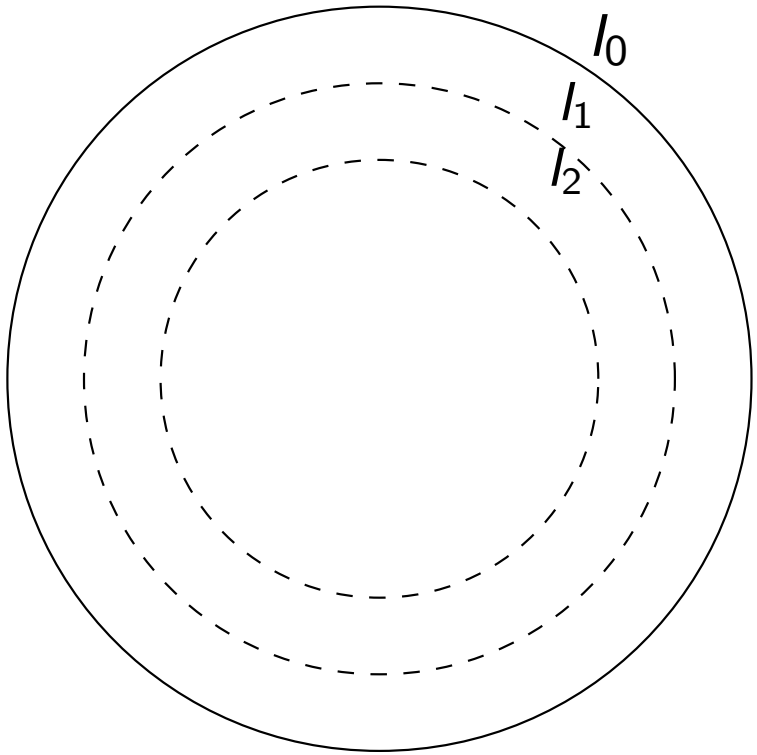
1. Start with initial ideal I_0 and iterate.
2. Iterative Step: $I_{j+1} = I_j \cap \{p \mid \mathrm{Lie}_F(p) \in I_j\}$.
3. Stop if $I_n = I_{n+1}$.

Strategy



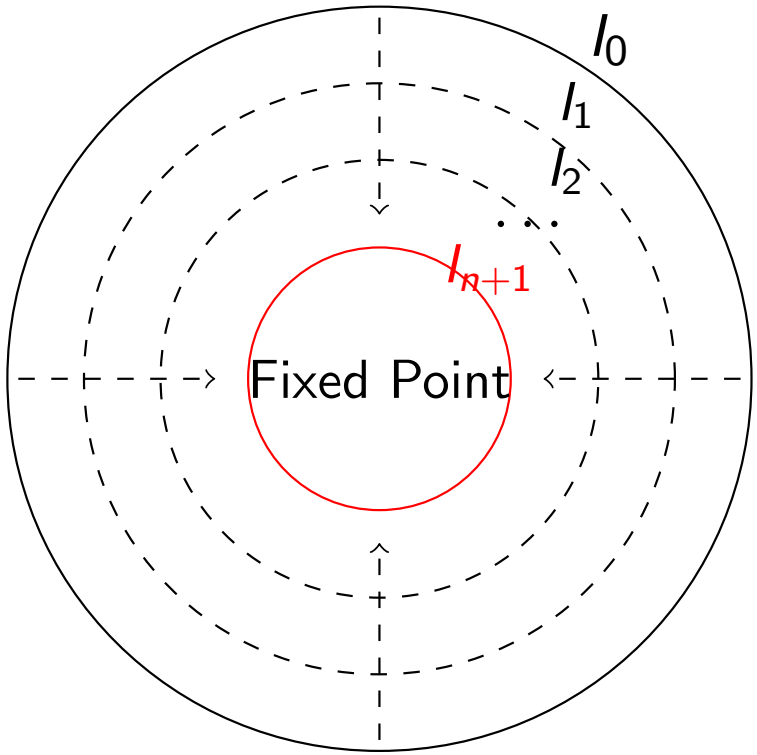
1. Start with initial ideal I_0 and iterate.
2. Iterative Step: $I_{j+1} = I_j \cap \{p \mid \mathrm{Lie}_F(p) \in I_j\}$.
3. Stop if $I_n = I_{n+1}$.

Strategy



1. Start with initial ideal I_0 and iterate.
2. Iterative Step: $I_{j+1} = I_j \cap \{p \mid \mathrm{Lie}_F(p) \in I_j\}$.
3. Stop if $I_n = I_{n+1}$.

Strategy



1. Start with initial ideal I_0 and iterate.
2. Iterative Step: $I_{j+1} = I_j \cap \{p \mid \mathrm{Lie}_F(p) \in I_j\}$.
3. Stop if $I_n = I_{n+1}$.

Ideal Refinement

$$\partial_F I = I \cap \{p \mid \mathsf{Lie}_F(p) \in I\}\,.$$

I.e, retain $p \in I$ such that $\mathcal{L}_F(p) \in I$.

Ideal Refinement

$$\partial_F I = I \cap \{p \mid \mathrm{Lie}_F(p) \in I\}.$$

I.e, retain $p \in I$ such that $\mathcal{L}_F(p) \in I$.
Algorithm:

1. *Compute Groebner basis* G of I.
2. *Compute Lie derivative* of each generator G.
3. *Intersect ideal* generated by Lie derivatives with I.
4. *Compute Syzygies* for each generator of the intersection.
5. *Multiply matrix* representing syzygies with generators in G.
6. Result is the set of generators of $\partial_F I$.

Convergence

- Convergence if the ring $K[\vec{x}]$ satisfies descending chain condition.
- But ideals in $\mathcal{R}[\vec{x}]$ do not satisfy *descending chain condition*.
- Solution: "Under-approximate" iteration inside a vector space.

Pseudo-Ideal

Finite dimensional vector-space inside an ideal. [Colón, 2004]

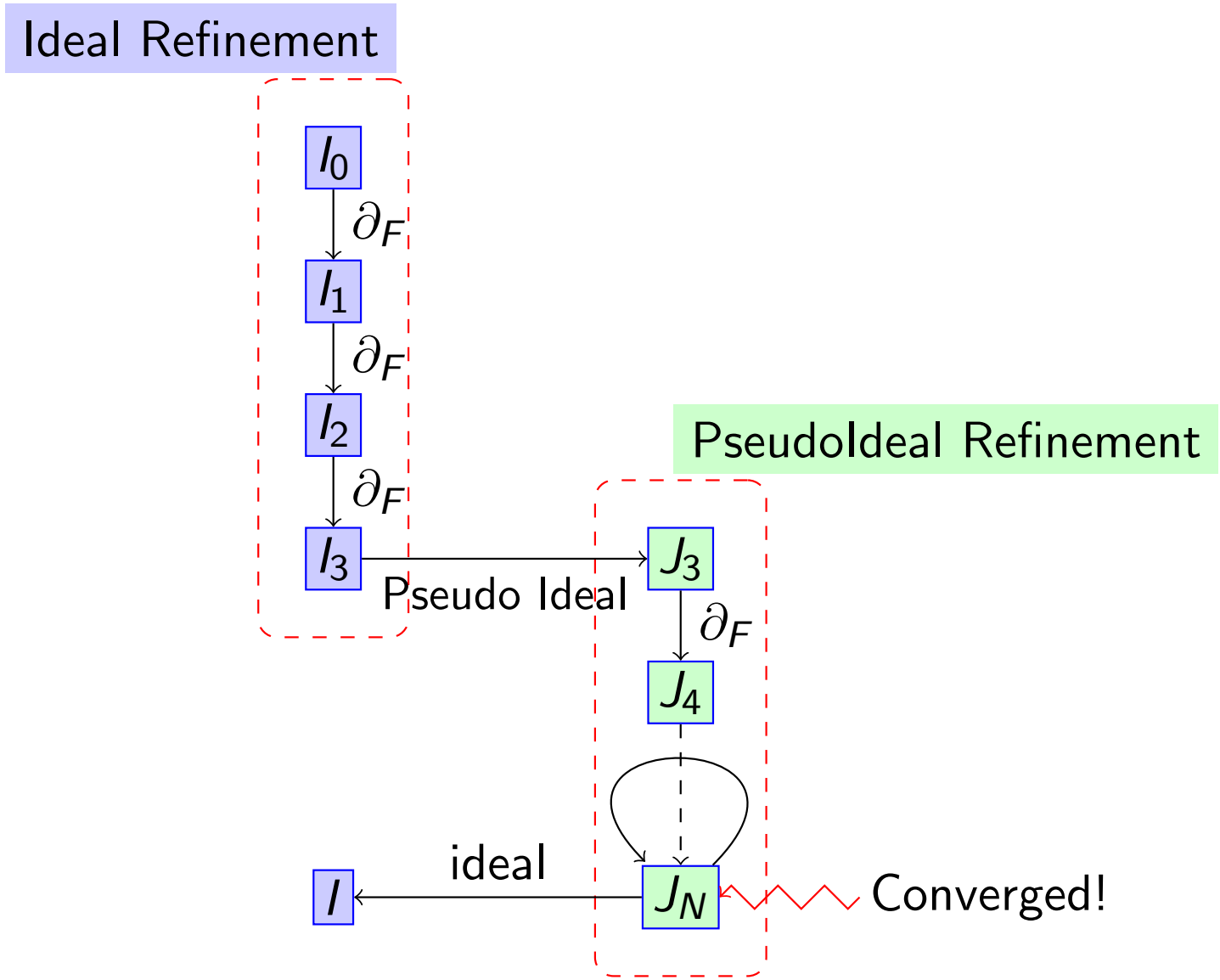
- Closure under addition.
- Multiplication closure with *polynomials with degree* $\leq d$.
- Vector space of polynomials.
- Descending chain condition holds.

Further Details

Sank., Automatic Invariant Generation for Algebraic Systems using Ideal Fixed Points, Hybrid Systems: Computation and Control, Apr. 2010

Implementation

Combine ideal and pseudo-ideal iterations:



Aircraft Collision Avoidance

Vector Field:

$$\begin{array}{cccc} x_1' = d_1 & x_2' = d_2 & d_1' = -\omega d_2 & d_2' = \omega d_1 \\ y_1' = e_1 & y_2' = e_2 & e_1' = -\theta e_2 & e_2' = \theta e_1 \\ a' = 0 & b' = 0 & r_1' = 0 & r_2' = 0 \end{array}$$

Initial set

$$\left[\begin{array}{c} x_1 = y_1 = r_1 \ \wedge \ x_2 = y_2 = r_2 \ \wedge \ d_1 = a \ \wedge \\ d_2 = 0 \ \wedge \ e_1 = b \ \wedge \ e_2 = 0 \end{array} \right]$$

Positive Invariant Set Obtained:

$$
\begin{aligned}
&\mathbf{p_1}: e_1^2 + e_2^2 - b^2, \\
&\mathbf{p_2}: d_1^2 + d_2^2 - a^2 \\
&\mathbf{p_3}: e_1 - r_2\theta + \theta y_2, \\
&\mathbf{p_4}: -a + d_1 - r_2\omega + \omega x_2 \\
&\mathbf{p_5}: b - e_2 - r_1\theta + \theta y_1, \\
&\mathbf{p_6}: -br_1 + by_1 + e_1 r_2 - e_1 y_2 - e_2 r_1 + e_2 y_1 \\
&\mathbf{p_7}: br_2 - by_2 - e_1 r_1 + e_1 y_1 - e_2 r_2 + e_2 y_2 \\
&\mathbf{p_8}: -d_2 - r_1\omega + \omega x_1 \\
&\mathbf{p_9}: ad_2 r_2 - ad_2 x_2 + d_1 d_2 r_2 - d_1 d_2 x_2 - r_1 d_2^2 r_1 + d_2^2 x_1 \\
&\mathbf{p_{10}}: ar_1 - ax_1 - d_1 r_1 + d_1 x_1 - d_2 r_2 + d_2 x_2
\end{aligned}
$$

Ongoing Work

- Polynomial inequality invariants:
  - Refinement over cones of positive semi-definite (psd) polynomials.
  - Using *sum-of-squares* relaxation and semi-definite programming. [Shor'87, Parillo'03]

Ongoing Work

- Polynomial inequality invariants:
  - Refinement over cones of positive semi-definite (psd) polynomials.
  - Using *sum-of-squares* relaxation and semi-definite programming. [Shor'87, Parillo'03]
- Study *homomorphisms* between dynamical systems.
  - Similar *topological semi-conjugacy*.
  - Algorithm for computing *linearizing homomorphisms*.
  - Encouraging results. [Sank.(draft) '10]