Lecture 5
Deductive Verification of Control Protocols

Richard M. Murray
Nok Wongpiromsarn   Ufuk Topcu
California Institute of Technology

EECI, 16 May 2012

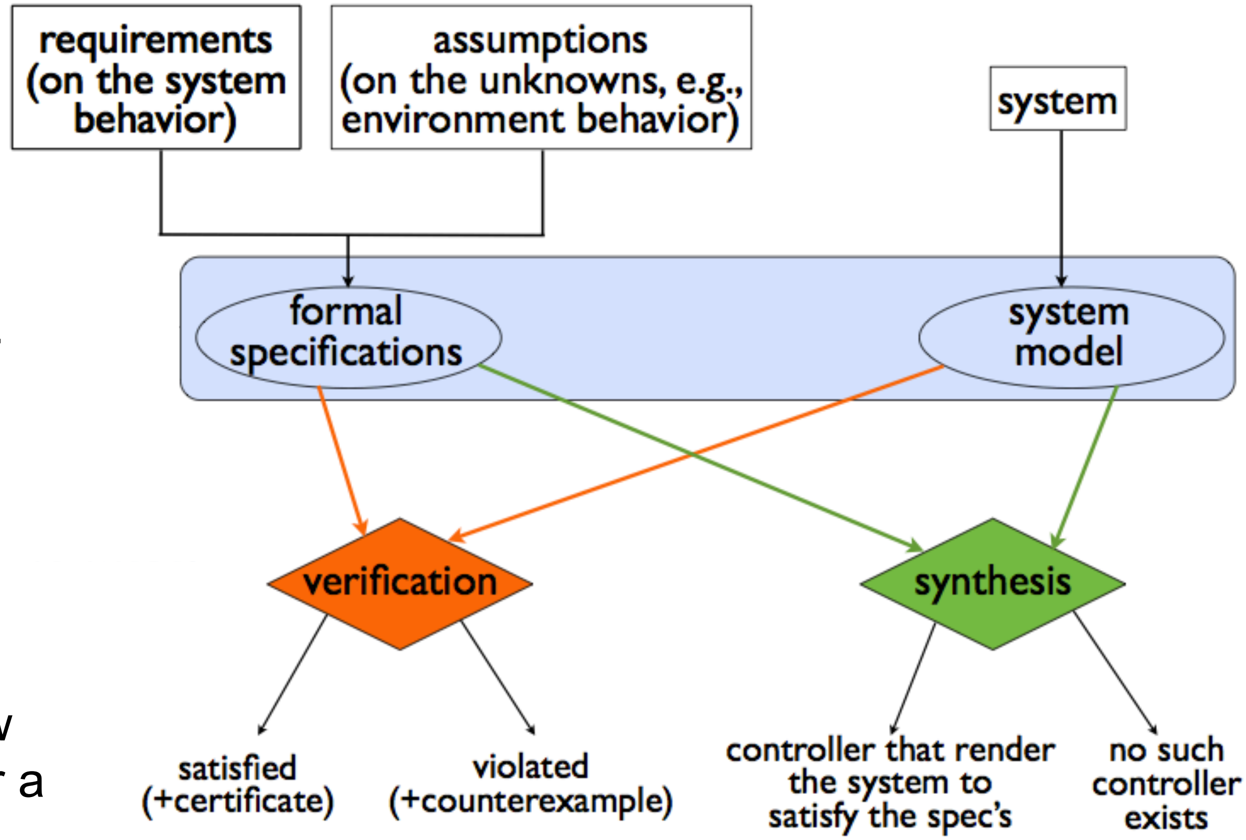
Outline

- Brief review: where we are at in the course so far
- Barrier certificates and verification of hybrid control systems
- Verification of async control protocols for multi-agent, cooperative control

Formal Methods for System Verification

Specification using LTL

- Linear temporal logic (LTL) is a math'l language for describing linear-time prop's
- Provides a particularly useful set of operators for constructing LT properties without specifying sets



Methods for verifying an LTL specification

- *Theorem proving*: use formal logical manipulations to show that a property is satisfied for a given system model
- *Model checking*: explicitly check all possible executions of a system model and verify that each of them satisfies the formal specification
  - Roughly like trying to prove stability by simulating *every* initial condition
  - Works because discrete transition systems have finite number of states
  - Very good tools now exist for doing this efficiently (SPIN, nuSMV, etc)

Hybrid, Multi-Agent System Description

Subsystem/agent dynamics - continuous

$$\dot{x}^i = f^i(x^i, \alpha^i, y^{\sim i}, u^i) \qquad x^i \in \mathbb{R}^n, u^i \in \mathbb{R}^m$$
$$y^i = h^i(x^i, \alpha^i) \qquad y^i \in \mathbb{R}^q$$

Agent mode (or "role") - discrete

- $\alpha \in \mathcal{A}$ encodes internal state + relationship to current task
- Transition $\alpha' = r(x, \alpha)$

Communications graph $\mathcal{G}$

- Encodes the system information flow
- Neighbor set $\mathcal{N}^i(x, \alpha)$

Communications channel

- Communicated information can be lost, delayed, reordered; rate constraints

$$y^i_j[k] = \gamma y^i(t_k - \tau_j) \quad t_{k+1} - t_k > T_r$$

- γ = binary random process (packet loss)

Task

- Encode task as finite horizon optimal control + temporal logic (assume coupled)

$$J = \int_0^T L(x, \alpha, u)\, dt + V(x(T), \alpha(T)),$$

$$(\varphi_{init} \wedge \Box \varphi_e) \implies (\Box \varphi_s \wedge \Diamond \varphi_g)$$

Strategy

- Control action for individual agents

$$u^i = \gamma(x, \alpha) \qquad \{g^i_j(x, \alpha) : r^i_j(x, \alpha)\}$$

$$\alpha^{i\,\prime} = \begin{cases} r^i_j(x, \alpha) & g(x, \alpha) = \text{true} \\ \text{unchanged} & \text{otherwise.} \end{cases}$$

***Decentralized* strategy**

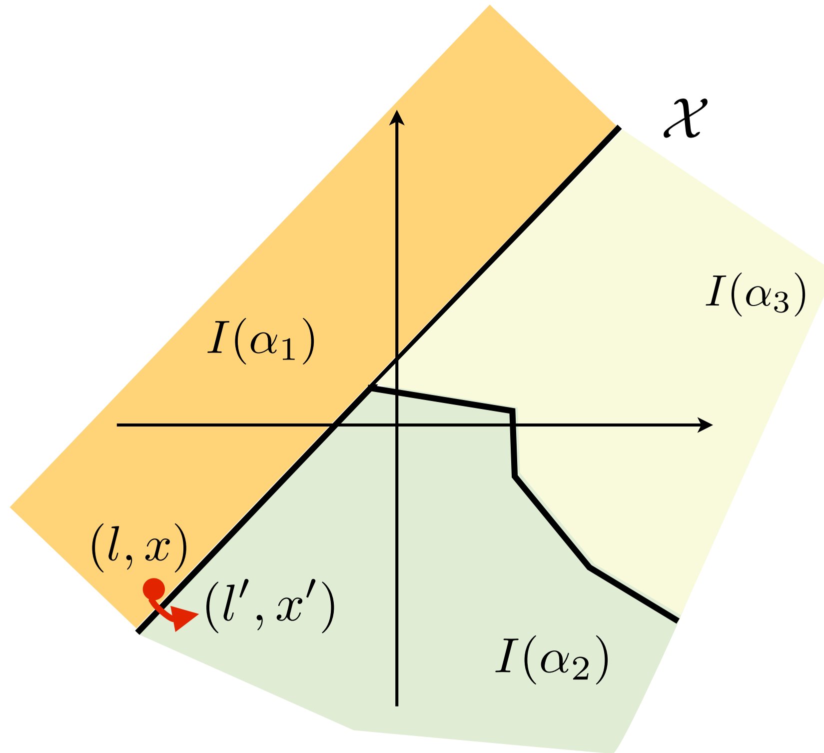
$$u^i(x, \alpha) = u^i(x^i, \alpha^i, y^{-i}, \alpha^{-i})$$
$$y^{-i} = \{y^{j_1}, \ldots, y^{j_{m_i}}\}$$
$$j_k \in \mathcal{N}^i \quad m_i = |\mathcal{N}^i|$$

- Similar structure for role update

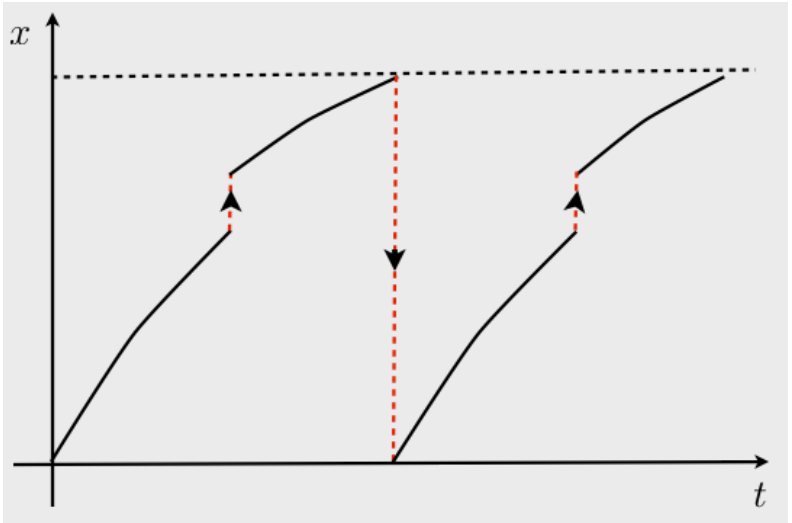
A (simple) hybrid system model

Hybrid system: $H = (\mathcal{X}, L, X_0, I, F, T)$ with

- $\mathcal{X}$, continuous state space;
- L, finite set of locations (modes);
- Overall state space $X = \mathcal{X} \times L$;
- $X_0 \subseteq X$, set of initial states;
- $I : L \to 2^{\mathcal{X}}$, *invariant* that maps $l \in L$ to the set of possible continuous states while in location l;
- $F : X \to 2^{\mathbb{R}^n}$, set of vector fields, i.e., $\dot{x} \in F(l, x)$;
- $T \subseteq X \times X$, relation capturing discrete transitions between locations.



$L = \{\alpha_1, \alpha_2, \alpha_3\}$



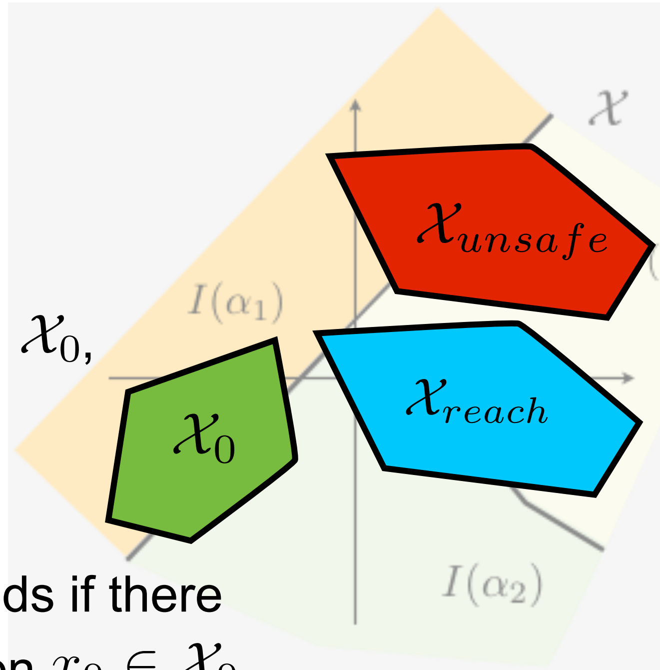
Specifications

Given: $H = (\mathcal{X}, L, X_0, I, F, T)$

Solution at time *t* with the initial condition $x_0 \in \mathcal{X}_0$: $\phi(t; x_0)$

- With the simple model *H*, specifying the initial state also specifies the initial mode.

Sample temporal properties:



- *Stability*: Given equilibrium $x_e \in \mathcal{X}$, for all $x_0 \in \mathcal{X}_0 \subseteq \mathcal{X}$,

  $$\phi(t; x_0) \in \mathcal{X}, \ \forall t \ \text{ and } \ \phi(t; x_0) \to x_e, \ t \to \infty$$

- *Safety*: Given $\mathcal{X}_{unsafe} \subseteq \mathcal{X}$, safety property holds if there exists no t_{unsafe} and trajectory with initial condition $x_0 \in \mathcal{X}_0$,

  $$\phi(t_{unsafe}; x_0) \in \mathcal{X}_{unsafe}$$
  $$\phi(t; x_0) \in \mathcal{X}, \ \forall t \in [0, t_{unsafe}]$$

- *Reachability*: Given $\mathcal{X}_{reach} \subseteq \mathcal{X}$, reachability property holds if there exists finite $t_{reach} \geq 0$ and a trajectory with initial condition $x_0 \in \mathcal{X}_0$,

  $$\phi(t_{reach}; x_0) \in \mathcal{X}_{reach} \ \text{ and } \ \phi(t; x_0) \in \mathcal{X}, \ \forall t \in [0, t_{reach}]$$

- *Eventuality*: reachable from every initial condition

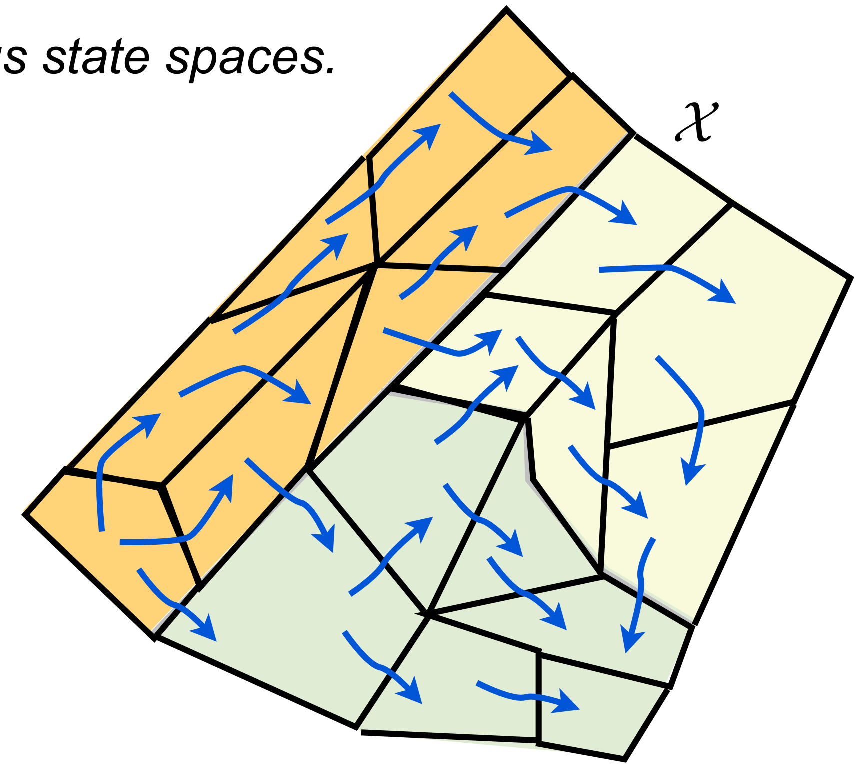
- Combinations of the above, e.g., starting in X_A, reach both X_B and X_C, but X_B will not be reached before X_C is reached while staying safe.

Verification of hybrid systems: Overview

Why not directly use model checking?

- Model checking applied to finite transitions systems
- Exhaustively search for counterexamples....
  - if found, property does not hold.
  - if there is no counterexample in all possible executions, the property is verified.

Exhaustive search is not possible over continuous state spaces.



Approaches for hybrid system verification:

1. Construct finite-state approximations and apply model checking
   - Preserve the meaning of the properties, i.e., proposition preserving partitions
   - Use "over"- or "under"-approximations

2. Deductive verification
   - Construct Lyapunov-type certificates
   - Account for the discrete jumps in the construction of the certificate

3. Explicitly construct the set of reachable states
   - Limited classes of temporal properties (e.g., reachability and safety)
   - Not covered in this lecture

What does deductive verification mean?

Example with continuous, nonlinear dynamics:

$$\dot{x}(t) = f(x(t))$$

where $x(t) \in \mathbb{R}^n$, $f(0) = 0, x = 0$ is an asymptotically stable equilibrium.

Region-of-attraction: $\mathcal{R} := \left\{ x : \lim_{t \to \infty} \phi(t; x) = 0 \right\}$

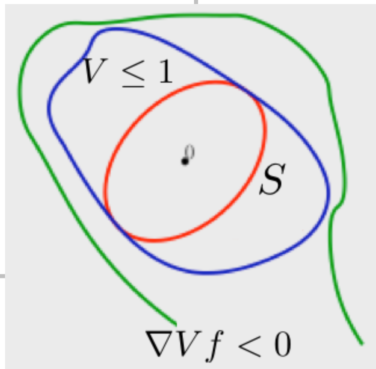
Question 1 (a system analysis question):
Given $S \subset \mathbb{R}^n$, is S invariant and $S \subseteq \mathcal{R}$?

the question we <u>want</u> to answer

the question we <u>attempt</u> to answer

Question 2 (an algebraic question):
Does there exist a continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ such that

- *V* is positive definite,
- *V(0) = 0*,
- $\Omega := \{x : V(x) \leq 1\} \subset \{x : \nabla V \cdot f(x) < 0\} \cup \{0\}$
- $S \subseteq \Omega$?

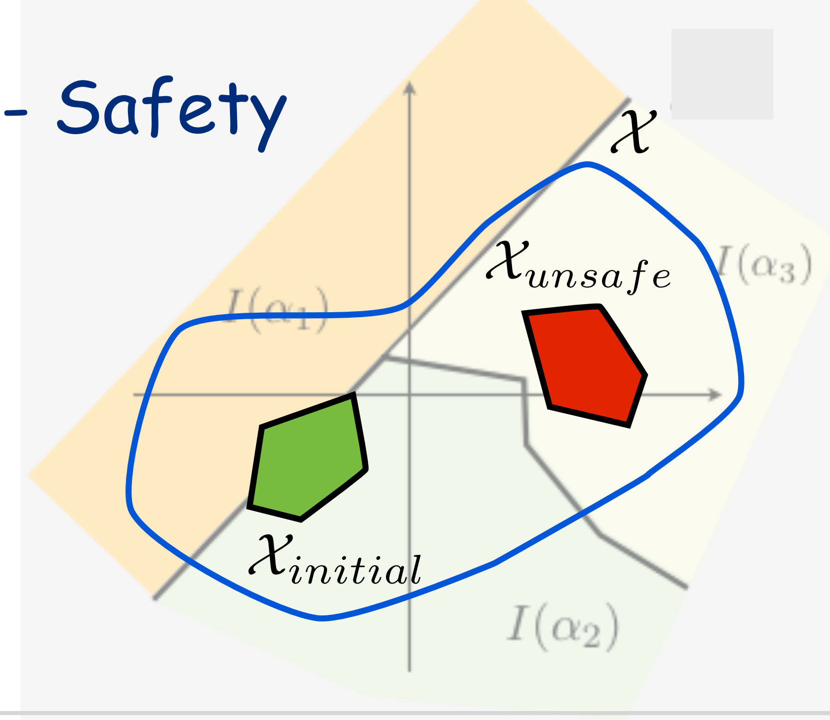


Yes to Question 2 $\rightarrow$ Yes to Question 1.

Barrier Certificates - Safety

Safety property holds if there exists
<u>no</u> $T \geq 0$ and trajectory such that:

$$x = \phi(0; x) \in \mathcal{X}_{initial}$$
$$\phi(T; x) \in \mathcal{X}_{unsafe}$$
$$\phi(t; x) \in \mathcal{X} \ \forall t \in [0, T].$$



Continuous dynamics:

$$\dot{x}(t) = f(x(t))$$

Suppose there exists a differentiable function B such that

$$B(x) \leq 0, \ \forall x \in \mathcal{X}_{initial}$$
$$B(x) > 0, \ \forall x \in \mathcal{X}_{unsafe}$$
$$\frac{\partial B}{\partial x} f(x) \leq 0, \ \forall x \in \mathcal{X}.$$

Then, the safety property holds.

Hybrid dynamics:

$$H = (\mathcal{X}, L, X_0, I, F, \mathcal{T})$$

Suppose there exist differentiable functions B_l (for each mode) such that

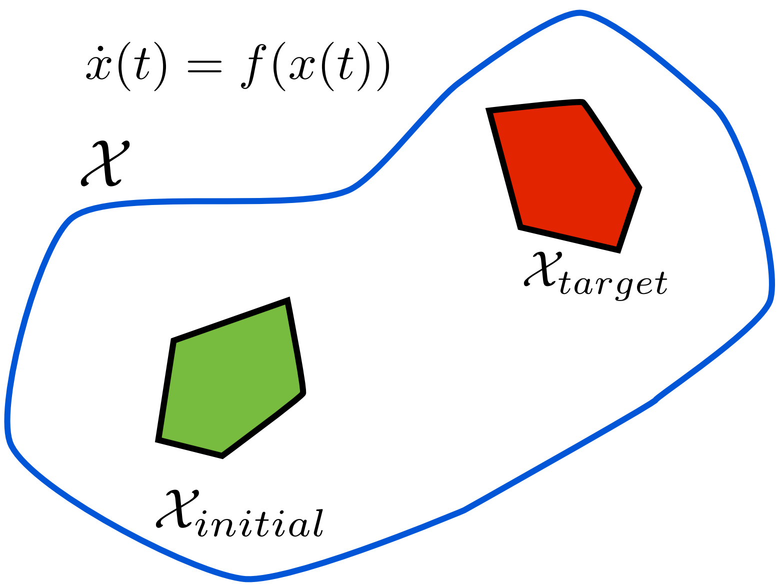
$$B_l(x) \leq 0, \ \forall x \in I(l) \cap \mathcal{X}_{initial}$$
$$B_l(x) > 0, \ \forall x \in I(l) \cap \mathcal{X}_{unsafe}$$
$$\frac{\partial B_l}{\partial x} F(x) \leq 0, \ \forall x \in I(l)$$
$$B_{l'}(x') - B_l(x) \leq 0, \ \text{for each jump} \quad (l, x) \to (l', x')$$

Then, the safety property holds.

Barrier Certificates - Eventuality

Don't quite understand conditions

$\dot{x}(t) = f(x(t))$



$\mathcal{X}$, $\mathcal{X}_{target}$, $\mathcal{X}_{initial}$ are bounded

Eventuality property holds if for $x_0 \in \mathcal{X}_{initial}$,

$$\phi(T; x_0) \in \mathcal{X}_{target}$$

$$\phi(t; x_0) \in \mathcal{X}, \ \forall t \in [0, T]$$

for some non-negative *T*.

notation: set closure

Suppose that f is continuously differentiable and there exists a continuously differentiable function *B* such that

$$B(x) \leq 0, \ \forall x \in \mathcal{X}_{initial}$$

don't leave $\mathcal{X}$ before reaching $\mathcal{X}_{target}$ →

$$B(x) > 0, \ \forall x \in \overline{\partial \mathcal{X} \backslash \partial \mathcal{X}_{target}}$$

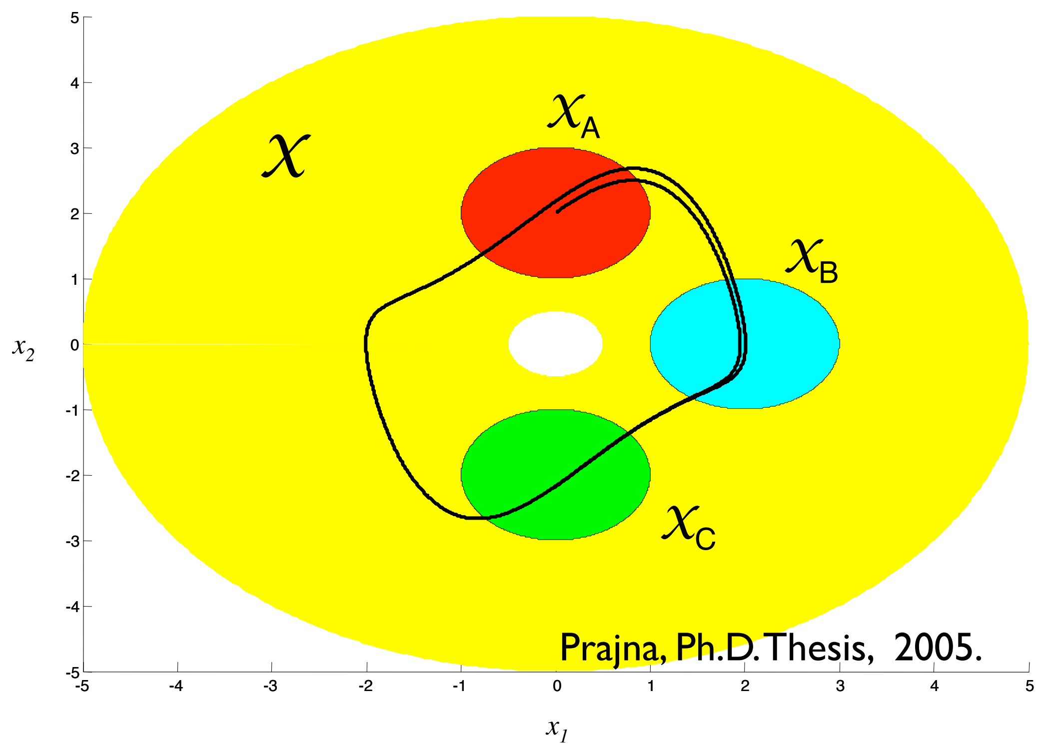
leave $\mathcal{X} \backslash \mathcal{X}_{target}$ in finite time →

$$\frac{\partial B}{\partial x}(x) \cdot f(x) < 0, \ \forall x \in \overline{\mathcal{X} \backslash \mathcal{X}_{target}}$$

Then, the eventuality property holds.

- Straightforward extensions for hybrid dynamics as in safety verification are possible.

Composing Barrier Certificates



incorporating disturbances and uncertainties

If system starts in $\mathcal{X}_A$, then both $\mathcal{X}_B$ and $\mathcal{X}_C$ are reached in finite time, but $\mathcal{X}_C$ will not be reached before system reaches $\mathcal{X}_B$.

$$\begin{cases} B_1(x) \le 0 & \forall x \in \mathcal{X}_A, \\ B_1(x) > 0 & \forall x \in \partial\mathcal{X} \cup \mathcal{X}_C, \\ \frac{\partial B_1}{\partial x}(x) f(x,d) \le -\epsilon & \forall (x,d) \in (\mathcal{X} \setminus \mathcal{X}_B) \times \mathcal{D}, \end{cases}$$

$$\begin{cases} B_2(x) \le 0 & \forall x \in \mathcal{X}_A, \\ B_2(x) > 0 & \forall x \in \partial\mathcal{X}, \\ \frac{\partial B_2}{\partial x}(x) f(x,d) \le -\epsilon & \forall x \in (\mathcal{X} \setminus \mathcal{X}_C) \times \mathcal{D}, \end{cases}$$

Constructing Barrier Certificates

Step 1: System properties → algebraic conditions

- Lyapunov functions, barrier certificates, dissipation inequalities

} Problem-dependent

Step 2: Algebraic conditions → numerical optimization

- Restrict attention to polynomial vector fields, polynomial certificates
- S-procedure like conditions for set containment constraints
- Sum-of-square (SOS) relaxations for polynomial non-negativity
- Convert to semi-definite programming (SDP) problems

Step 3: Solve resulting set of SDPs

- Often in the form of linear matrix inequalities (LMIs)

Step 4: Construct polynomial certificates based on SDP solutions

} Generally taken care of by software packages.

This lecture (brief overview only):

- Positive semidefinite polynomials and sum-of- squares (SOS) programming
- Set containment conditions and S-procedure

Positive semidefinite polynomials

$\mathbb{R}[x_1, \ldots, x_n]$ or $\mathbb{R}[x]$ denotes the set of polynomials (with real coefficients) in the variables $\{x_1, \ldots, x_n\}$.

- $p \in \mathbb{R}[x]$ is positive semi-definite (PSD) if $p(x) \geq 0 \; \forall x$. The set of PSD polynomials in n variables $\{x_1, \ldots, x_n\}$ will be denoted $\mathcal{P}[x_1, \ldots, x_n]$ or $\mathcal{P}[x]$.

- Testing if $p \in \mathcal{P}[x]$ is NP-hard when the polynomial degree is at least four.

Reference: Parrilo, P., *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*, Ph.D. thesis, California Institute of Technology, 2000. (Chapter 4 of this thesis and the reference contained therein summarize the computational issues associated with verifying global non-negativity of functions.)

Sum-of-Squares Polynomials

p is a sum of squares (SOS) if there exist polynomials $\{f_i\}_{i=1}^N$ such that $p = \sum_{i=1}^N f_i^2$.
The set of SOS polynomials in n variables $\{x_1, \ldots, x_n\}$ will be denoted $\Sigma\left[x_1, \ldots, x_n\right]$ or $\Sigma\left[x\right]$.

If p is a SOS then p is PSD.

For every polynomial p of degree 2d, there exists a symmetric matrix Q such that

$$p(x) = z(x)^T Q z(x)$$

with

$$z(x) := \left[1, x_1, \ldots, x_n, x_1^2, x_1 x_2, \ldots, x_n^2, \ldots, x_n^d\right]^T$$

p is SOS if and only if there exists $Q \succeq 0$ s.t. $p(x) = z(x)^T Q z(x)$

SOS example

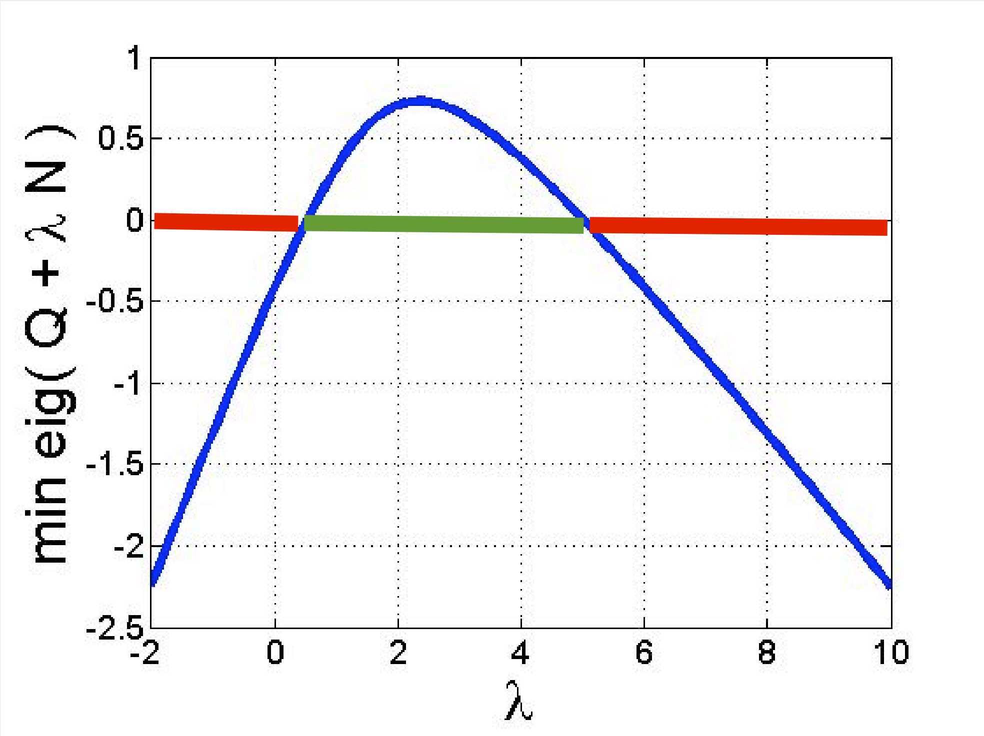
$$p(x) = z(x)^T Q z(x)$$
$$0 = z(x)^T N z(x)$$
$$(x_1 x_2) \cdot (x_1 x_2) = x_1^2 \cdot x_2^2$$

All possible Gram matrix representations of

$$p(x) = 2x_1^4 + 2x_1^3 x_2 - x_1^2 x_2^2 + 5x_2^4$$

are given by $z^T (Q + \lambda N) z$ where:

$$z = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}, \quad Q = \begin{bmatrix} 2 & 1 & -0.5 \\ 1 & 0 & 0 \\ -0.5 & 0 & 5 \end{bmatrix}, \quad N = \begin{bmatrix} 0 & 0 & -0.5 \\ 0 & 1 & 0 \\ -0.5 & 0 & 0 \end{bmatrix}$$



p is SOS iff

$$Q + \lambda N \succeq 0 \quad \longleftarrow \text{LMI}$$

for some $\lambda \in \mathbb{R}$.

SOS programming

SOS Programming: Given $c \in \mathbb{R}^m$ and polynomials $\{f_{j,k}\}_{j=1}^{N_s}\ {}_{k=0}^{m}$, solve:

$$V(x) = \sum \alpha_i \phi_i(x) \qquad \dot{V}(x) = \sum \alpha_i \frac{\partial \phi_i}{\partial x} f(x)$$

$$\min_{\alpha \in \mathbb{R}^m} c^T \alpha$$

$$V(x) \geq l_1(x) > 0 \implies -l(x) + V(x) > 0$$

subject to:

$$f_{1,0}(x) + f_{1,1}(x)\alpha_1 + \cdots + f_{1,m}(x)\alpha_m \in \Sigma[x]$$

$$\vdots$$

$$f_{N_s,0}(x) + f_{N_s,1}(x)\alpha_1 + \cdots + f_{N_s,m}(x)\alpha_m \in \Sigma[x]$$

There is freely available software (e.g. SOSTOOLS, YALMIP, SOSOPT) that:

1. Converts the SOS program to an SDP
2. Solves the SDP with available SDP codes (e.g. Sedumi)
3. Converts the SDP results back into polynomial solutions

Set containment conditions

Given polynomials g_1 and g_2, define sets S_1 and S_2:

$$S_1 := \{x \in \mathbb{R}^n \ : \ g_1(x) \leq 0\}$$
$$S_2 := \{x \in \mathbb{R}^n \ : \ g_2(x) \leq 0\}$$

Is $S_2 \subseteq S_1$?

Polynomial S-procedure

$$\exists \lambda \in \Sigma[x] \text{ s.t. } -g_1(x) + \lambda(x) g_2(x) \in \Sigma[x]$$
$$\Downarrow$$
$$\exists \lambda \text{ positive semidefinite polynomial s.t. } -g_1(x) + \lambda(x) g_2(x) \geq 0 \ \forall x$$
$$\Downarrow$$
$$\{x : g_2(x) \leq 0\} \subseteq \{x : g_1(x) \leq 0\}$$

Example: $B(x) \leq 0, \quad \forall x \in \mathcal{X}_{initial}$

Suppose $\mathcal{X}_{initial} = \{x : g(x) \leq 0\}$ for some *g*

Sufficient condition: There exists positive semidefinite function *s* such that

$$-B(x) + s(x)g(x) = -B(x) - s(x)(-g(x)) \geq 0, \quad \forall x \in \mathbb{R}^n$$

Global stability theorem

Theorem: Let $l_1, l_2 \in \mathbb{R}[x]$ satisfy $l_i(0) = 0$ and $l_i(x) > 0 \; \forall x \neq 0$ for $i = 1, 2$. If there exists $V \in \mathbb{R}[x]$ such that:

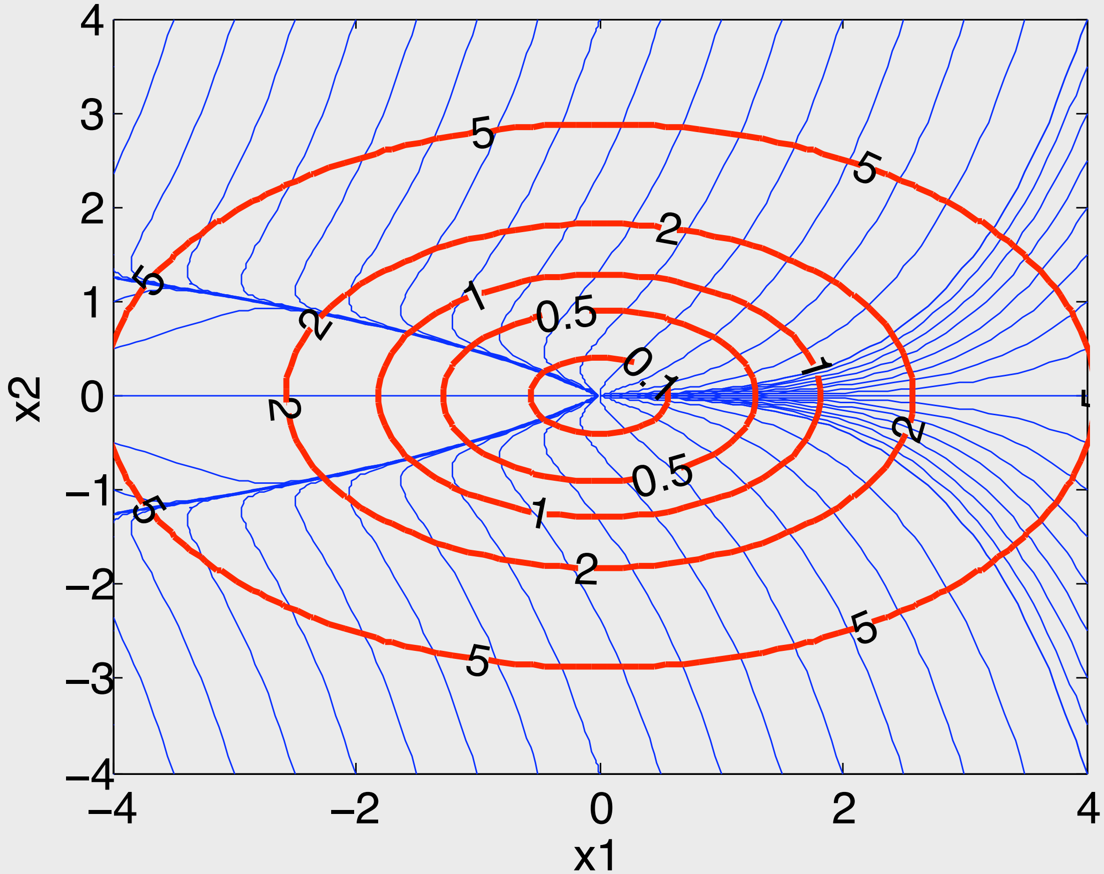
- $V(0) = 0$
- $V - l_1 \in \Sigma[x]$ ← *V* is positive definite, radially unbounded
- $-\nabla V \cdot f - l_2 \in \Sigma[x]$ ← *V* is decreasing along the vector field

Then $\mathcal{R}_0 = \mathbb{R}^n$.

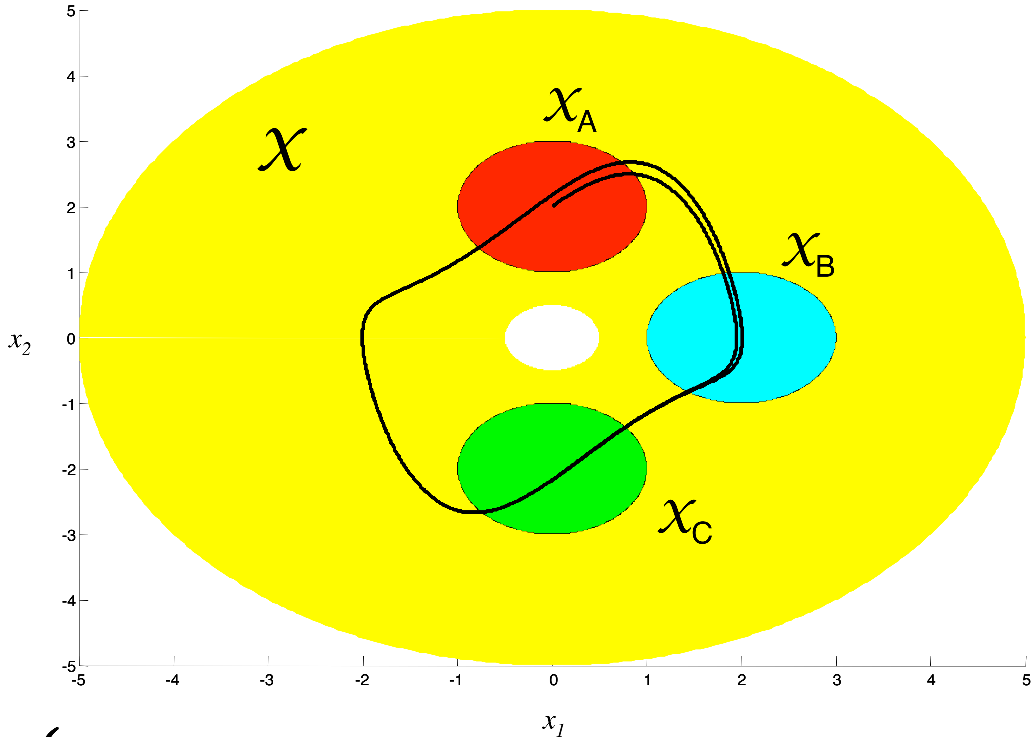
Reference: Vidyasagar, M., *Nonlinear Systems Analysis*, SIAM, 2002.
(Refer to Section 5.3 for theorems on Lyapunov's direct method.)

Global stability examples with sosopt

```
% Code from Parrilo1_GlobalStabilityWithVec.m

% Create vector field for dynamics
pvar x1 x2;
x = [x1;x2];
x1dot = -x1 - 2*x2^2;
x2dot = -x2 - x1*x2 - 2*x2^3;
xdot = [x1dot; x2dot];

% Use sosopt to find a Lyapunov function
% that proves x = 0 is GAS

% Define decision variable for quadratic
% Lyapunov function
zV = monomials(x,2);
V = polydecvar('c',zV,'vec');

% Constraint 1 : V(x) - L1 \in SOS
L1 = 1e-6 * ( x1^2 + x2^2 );
sosconstr{1} = V - L1;

% Constraint 2: -Vdot - L2 \in SOS
L2 = 1e-6 * ( x1^2 + x2^2 );
Vdot = jacobian(V,x)*xdot;
sosconstr{2} = -Vdot - L2;

% Solve with feasibility problem
[info,dopt,sossol] = sosopt(sosconstr,x);
Vsol = subs(V,dopt)
Vsol =
  0.30089*x1^2 + 1.8228e-017*x1*x2 + 0.6018*x2^2
```



Solving Hybrid Verification Problems using SOS



$$\begin{cases} B_1(x) \leq 0 & \forall x \in \mathcal{X}_A, \\ B_1(x) > 0 & \forall x \in \partial\mathcal{X} \cup \mathcal{X}_C, \\ \frac{\partial B_1}{\partial x}(x) f(x,d) \leq -\epsilon & \forall (x,d) \in (\mathcal{X} \setminus \mathcal{X}_B) \times \mathcal{D}, \end{cases}$$

$$\begin{cases} B_2(x) \leq 0 & \forall x \in \mathcal{X}_A, \\ B_2(x) > 0 & \forall x \in \partial\mathcal{X}, \\ \frac{\partial B_2}{\partial x}(x) f(x,d) \leq -\epsilon & \forall x \in (\mathcal{X} \setminus \mathcal{X}_C) \times \mathcal{D}, \end{cases}$$

If system starts in $\mathcal{X}_A$, then both $\mathcal{X}_B$ and $\mathcal{X}_C$ are reached in finite time, but $\mathcal{X}_C$ will not be reached before system reaches $\mathcal{X}_B$.

- Describe all sets using polynomials
  - $X_A = \{x : g_1(x) \leq 0\}$
- Use S procedure to convert set containment problems into polynomial inequalities
- Use SOS tools to search for coefficients of basis polynomials the give basis functions and multipliers

Hybrid, Multi-Agent System Description

Subsystem/agent dynamics - continuous

$$\dot{x}^i = f^i(x^i, \alpha^i, y^{\sim i}, u^i) \qquad x^i \in \mathbb{R}^n, u^i \in \mathbb{R}^m$$
$$y^i = h^i(x^i, \alpha^i) \qquad y^i \in \mathbb{R}^q$$

Agent mode (or "role") - discrete

- $\alpha \in \mathcal{A}$ encodes internal state + relationship to current task
- Transition $\alpha' = r(x, \alpha)$

Communications graph $\mathcal{G}$

- Encodes the system information flow
- Neighbor set $\mathcal{N}^i(x, \alpha)$

Communications channel

- Communicated information can be lost, delayed, reordered; rate constraints

$$y^i_j[k] = \gamma y^i(t_k - \tau_j) \qquad t_{k+1} - t_k > T_r$$

- γ = binary random process (packet loss)

Task

- Encode task as finite horizon optimal control + temporal logic (assume coupled)

$$J = \int_0^T L(x, \alpha, u)\, dt + V(x(T), \alpha(T)),$$

$$(\varphi_{init} \wedge \Box\varphi_e) \implies (\Box\varphi_s \wedge \Diamond\varphi_g)$$

Strategy

- Control action for individual agents

$$u^i = \gamma(x, \alpha) \qquad \{g^i_j(x, \alpha) : r^i_j(x, \alpha)\}$$

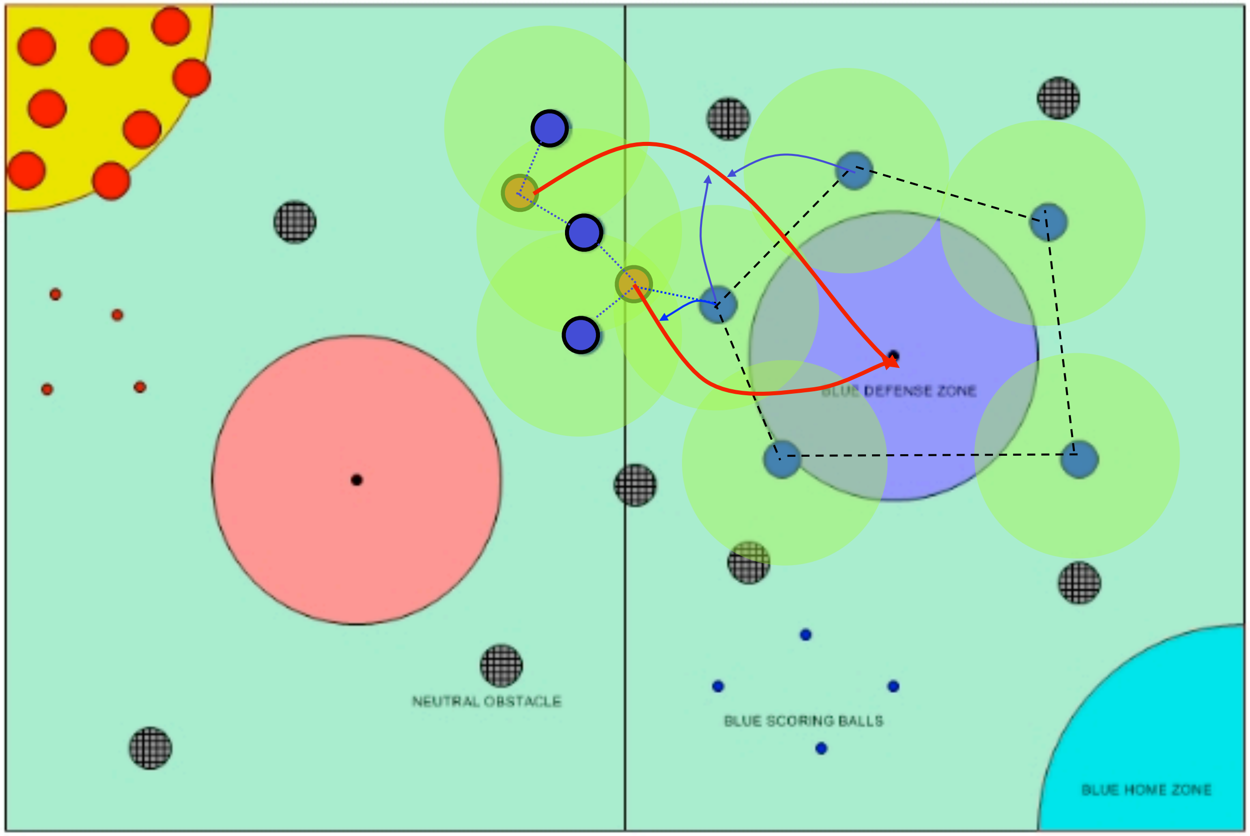
$$\alpha^{i\,\prime} = \begin{cases} r^i_j(x, \alpha) & g(x, \alpha) = \text{true} \\ \text{unchanged} & \text{otherwise.} \end{cases}$$

***Decentralized* strategy**

$$u^i(x, \alpha) = u^i(x^i, \alpha^i, y^{-i}, \alpha^{-i})$$
$$y^{-i} = \{y^{j_1}, \ldots, y^{j_{m_i}}\}$$
$$j_k \in \mathcal{N}^i \qquad m_i = |\mathcal{N}^i|$$

- Similar structure for role update

RoboFlag Subproblems



1.Formation control

- Maintain positions to guard defense zone

2.Distributed estimation

- Fuse sensor data to determine opponent location

3.Distributed assignment

- Assign individuals to tag incoming vehicles

Desirable features for designing and verifying distributed protocols

- Controls: stability, performance, robustness
- Computer science: safety, fairness, liveness
- Real-world: delays, asynchronous executions, (information loss)

Distributed Decision Making: RoboFlag Drill

Klavins CDC, 03

Task description

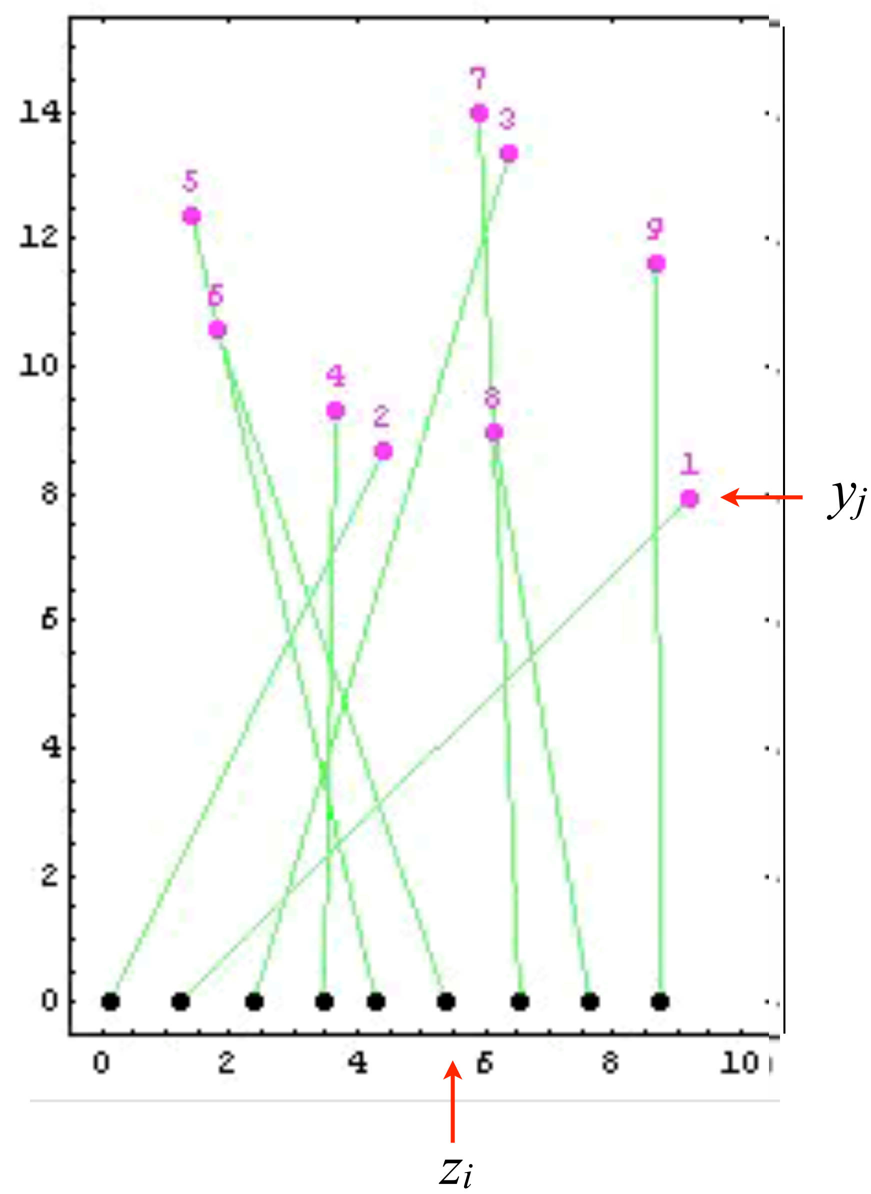
- Incoming robots should be blocked by defending robots
- Incoming robots are assigned randomly to whoever is free
- Defending robots must move to block, but cannot run into or cross over others
- Allow robots to communicate with left and right neighbors and switch assignments

Goals

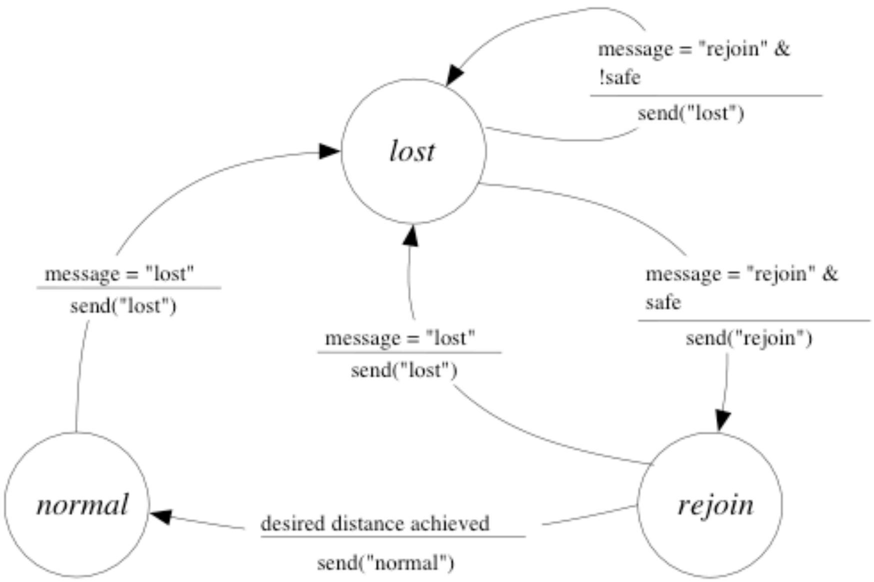
- Would like a provably correct, distributed protocol for solving this problem
- Should (eventually) allow for lost data, incomplete information

Questions

- How do we describe task in terms of LTL?
- Given a protocol, how do we prove specs?
- How do we design the protocol given specs?



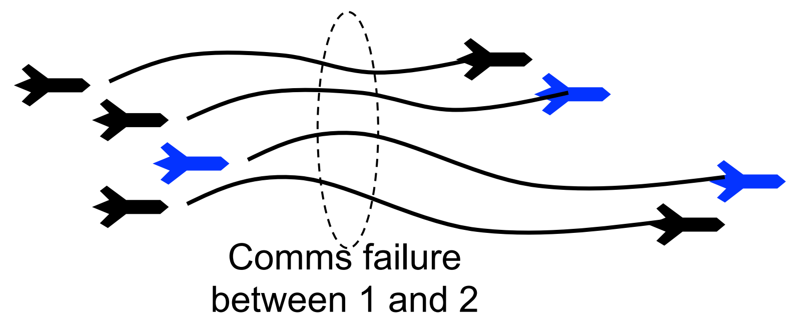
Lost Wingman Protocol Verification



Protocol specification in CCL

- Use guarded commands to implement finite state automaton
- Allows reasoning about controlled performance using semi-automated theorem proving
- Relies on Lyapunov certificates to provide information about controlled system

Lost wingman in fingertip formation



Temporal logic specification

$$\Psi_l \triangleq mode = lost \leadsto \Box d(\mathbf{x}_l, \mathbf{x}_f) > d_{sep}$$

- "Lost mode leads to the distance between the aircraft always being larger than d_{sep}"

CCL: Computation and Control Language

Formal Language for Provably Correct Control Protocols

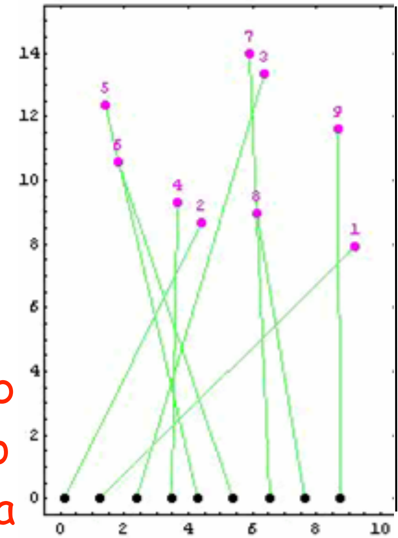
$P(k_1, k_2)$:= {
  initializers
  $guard_1 : rule_1$
  $guard_2 : rule_2$
  ...
}

"soup" of guarded commands

composition = union

non-shared variab remain local to component progra

$S(k_1, k_2) := P(k_1, k_2) + C(k_1+1)$ sharing y,u



CCL Protocol for Decentralized Target Allocation

CCL Interpreter

Formal programming language for control and computation. Interfaces with libraries in other languages.

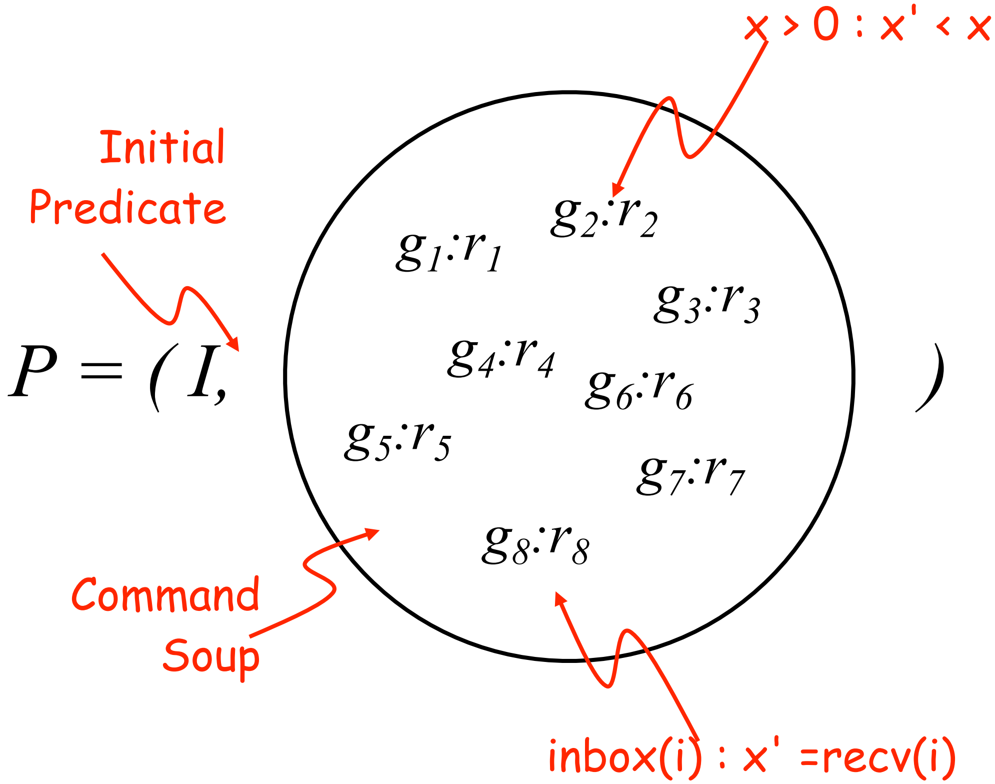
Formal Results

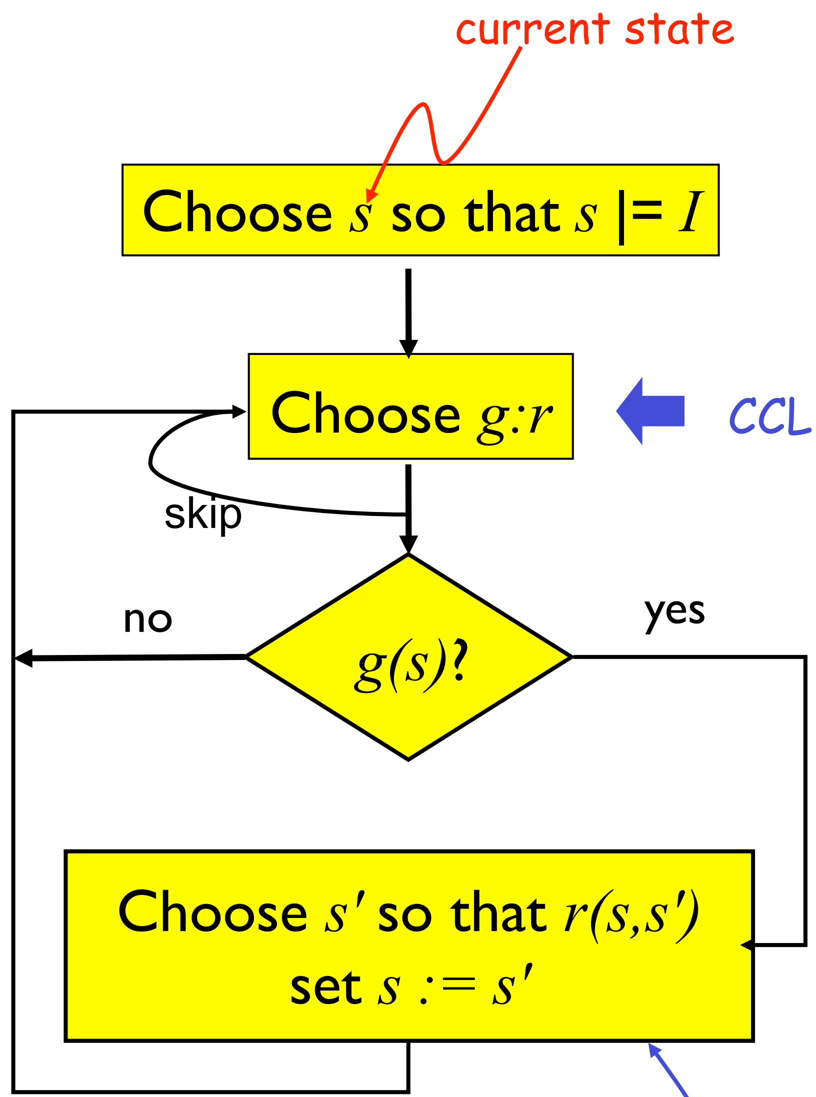
Formal semantics in transition systems and temporal logic. *RoboFlag* drill formalized and basic algorithms verified.

Automated Verification

CCL encoded in the *Isabelle* theorem prover; basic specs verified semi-automatically. Investigating various model checking tools.

Guarded Command Programs

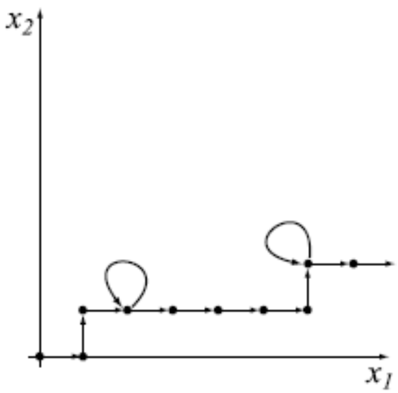




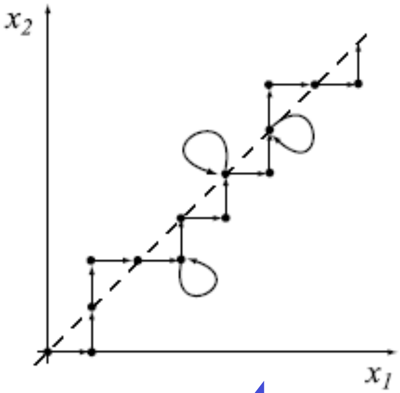
- Non-deterministic execution schedule models concurrency
- Easy to reason about programs
- Guarded commands = update functions

Any sequence of states produced by this process is a possible behavior of the system. We want to reason about them all.

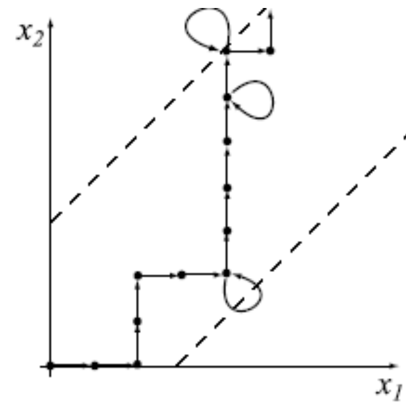
Scheduling and Composition



UNITY
Each command must be executed infinitely often.



EPOCH ⬅ CCL
Each command is executed before any are again.



SYNCH(τ)
In any interval, the difference in the number of times any two commands are executed is ≤ τ.

$P(i)$	
Initial	$x_i = 0$
Commands	$\mathit{true} \;:\; x_i' = x_i + 1$

$$Q = P(1) + P(2)$$

Program composition:
$(I_1,C_1) + (I_2,C_2) = (I_1 \wedge I_2, C_1 \cup C_2)$

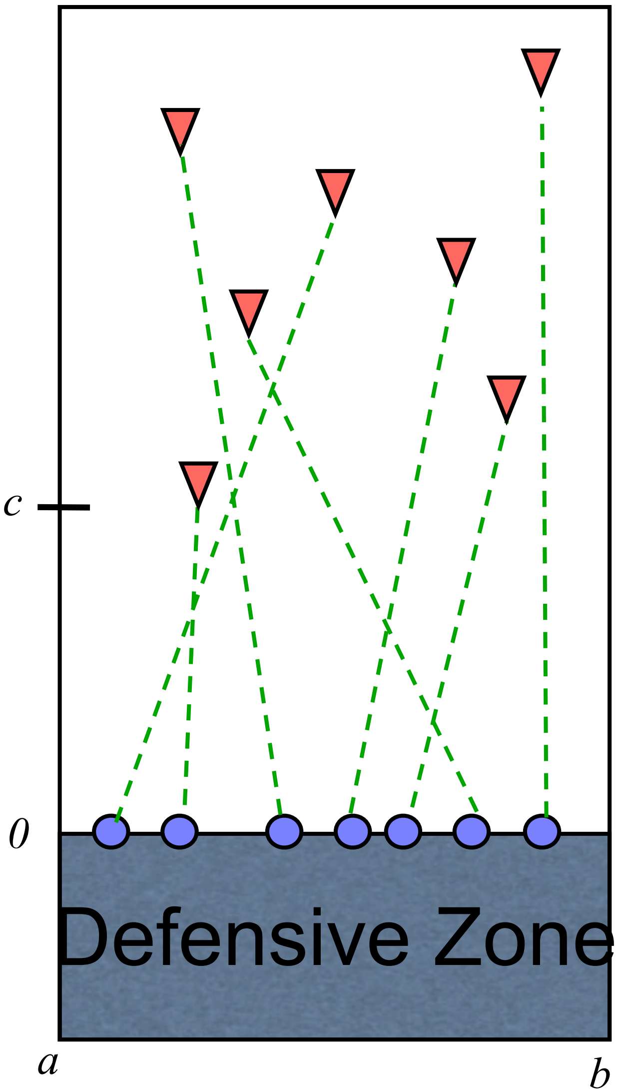
Thm: $\mathit{SYNCH}(1) \subseteq \mathit{EPOCH} \subseteq \mathit{SYNCH}(2) \subseteq \mathit{SYNCH}(3) \subseteq ... \subseteq \mathit{UNITY}$.

An Example CCL Program

```
include standard.ccl

program plant ( a, b, x0, delta ) := {
  x := x0;
  y := x;
  u := 0.0;
  true : {
    x := x + delta * ( a * x + b * u ),
    y := x,
    print ( " x = ", x, "\n" )
  };
};

program control() := {
  y := 0.0;
  u := 0.0;
  true : { u := -y };
};

program sys ( a, b, x0 ) := plant ( a, b, x0, 0.1 ) +
                             control ( 2*a/b ) sharing u, y;

exec sys ( 3.1, 0.75, 15.23 );
```

```
x = 3.216250
x = 3.095641
x = 2.979554
x = 2.867821
x = 2.760278
x = 2.656767
x = 2.557138
x = 2.461246
x = 2.368949
x = 2.280113
x = 2.194609
x = 2.112311
x = 2.033100
x = 1.956858
x = 1.883476
x = 1.812846
x = 1.744864
x = 1.679432
x = 1.616453
      ...
```

Example: RoboFlag Drill



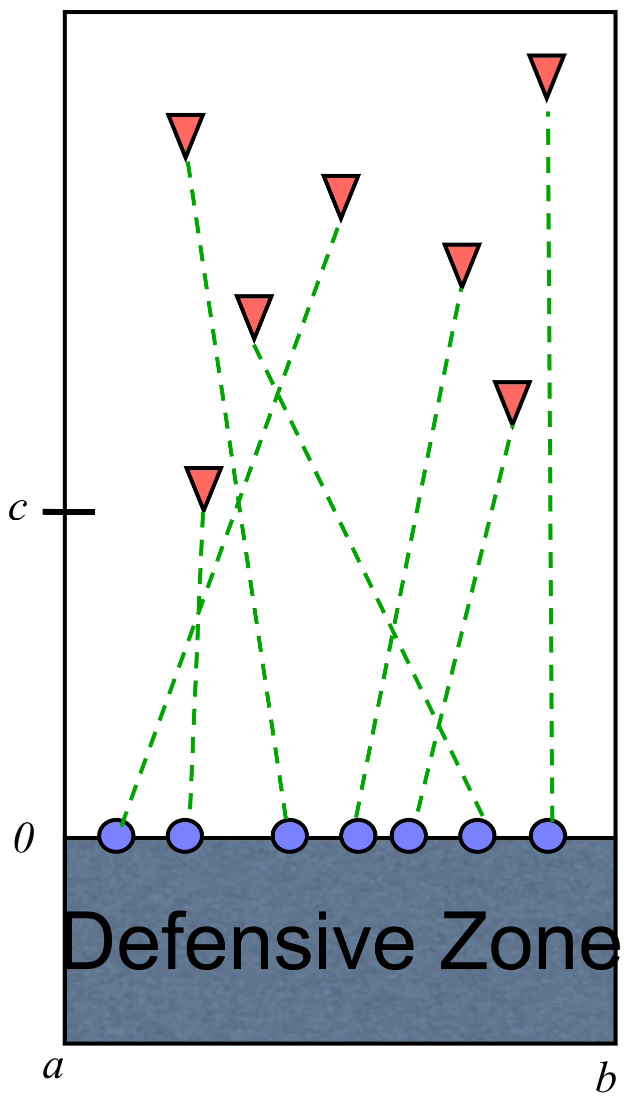
$Red(i)$	
Initial	$x_i \in [a,b] \wedge y_i > c$
Commands	$y_i > \delta \ : \ y_i' = y_i - \delta$
	$y_i \leq \delta \ : \ x_i' \in [a,b] \wedge y_i > c$

$$P_{Red}(n) = +_{i=1}^{n} Red(i)$$

$Blue(i)$	
Initial	$z_i \in [a,b] \wedge z_i < z_{i+1}$
Commands	$z_i < x_{\alpha(i)} \wedge z_i < z_{i+1} - \delta \ : \ z_i' = z_i + \delta$
	$z_i > x_{\alpha(i)} \wedge z_i > z_{i-1} + \delta \ : \ z_i' = z_i - \delta$

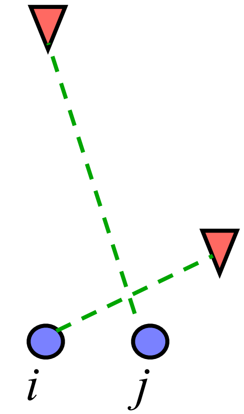
$$P_{Blue}(n) = +_{i=1}^{n} Blue(i)$$

RoboFlag Control Protocol



$\alpha(j)$ is too far down for i to get

$$r(i,j) = \begin{cases} 1 \text{ if } y_{\alpha(j)} < |z_i - x_{\alpha(j)}| \\ 0 \text{ otherwise} \end{cases}$$



$$\begin{aligned} switch(i,j) &= r(i,j)+r(j,i) < r(i,i)+r(j,j) \\ &\vee \ (r(i,j)+r(j,i) = r(i,i)+r(j,j) \\ &\quad \wedge x_{\alpha(i)} > x_{\alpha(j)}) \end{aligned}$$

$Proto(i)$	
Initial	$i \neq j \Rightarrow \alpha(i) \neq \alpha(j)$
Commands	$switch(i, i+1) \ : \ \alpha(i)' = \alpha(i+1)$ $\alpha(i+1)' = \alpha(i)$

$$P_{Proto}(n) = +_{i=1}^{n-1} Proto(i)$$

CCL Program for Switching Assignments

```
program Blue ( i ) := {

  red[alpha[i]][0] > blue[i] & blue[i] +
delta < toplimit i : {
    blue[i] := blue[i] + delta
  }

  red[alpha[i]][0] < blue[i] & blue[i] -
delta > botlimit i : {
    blue[i] := blue[i] - delta
  }

};
```

```
program Red ( i ) := {

  red[i][1] > delta : {
    red[i][1] := red[i][1] - delta
  }

  red[i][1] < delta : {
    red[i] := { rrand 0 n, rrand lowerlimit
n }
  }

};
```

```
fun r i j .
  if red[alpha[j]][1] < abs ( blue[i] -
red[alpha[j]][0] )
    then 1
    else 0
  end;

fun switch i j .
  r i j + r j i < r i i + r j j
  | ( r i j + r j i = r i i + r j j
    & red[alpha[i]][0]  > red[alpha[j][0] );

program ProtoPair ( i, j ) := {

  temp := 0;

  switch i j : {
    temp := alpha[i],
    alpha[i] := alpha[j],
    alpha[j] := temp,
  }

};
```

Properties for RoboFlag program

Safety (Defenders do not collide)

$$z_i < z_{i+1} \ \mathbf{co} \ z_i < z_{i+1}$$

Stability (switch predicate stays false)

- skip ∀v . v' = v state remains unchanged
- p **co** q □(p → [(○q ∨ skip) ∧ ◊○q])
  "if p is true, then next time state changes, q will be true"

$$\forall i \ . \ y_i > 2\delta \wedge z_i + 2\delta < z_{i+1} \wedge \neg switch_{i,i+1} \ \mathbf{co} \ \neg switch_{i,i+1}$$

Robots are "far enough" apart.

True if robots i and i +1 have targets that cause crossed paths

"Lyapunov" stability

- Let ρ be the number of blue robots that are too far away to reach their red robots
- Let β be the total number of conflicts in the current assignment
- Define the Lyapunov function that captures "energy" of current state (V = 0 is desired)

$$V = \left[\binom{n}{2} + 1\right]\rho + \beta \qquad \rho = \sum_{i=1}^{n} r(i,i) \qquad \beta = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \gamma(i,j) \quad \text{where} \quad \gamma(i,j) = \begin{cases} 1 & \text{if } x_{\alpha(i)} > x_{\alpha(j)} \\ 0 & \text{otherwise} \end{cases}$$

- Can show that V always decreases whenever a switch occurs

$$\forall i \ . \ z_i + 2\delta m < z_{i+1} \wedge \exists j \ . \ switch_{j,j+1} \wedge V = m \ \ \mathbf{co} \ V < m$$

Sketch of Proof for RoboFlag Drill

Thm Prf(n) $\models \Box\, z_i < z_{i+1}$

- For the RoboFlag drill with n defenders and n attackers, the location of defender will always be to the left of defender i+1.

More notation:

- Hoare triple notation: $\{p\}\ a\ \{q\} \equiv \forall\ s \xrightarrow{a} t,\ s \models p \rightarrow t \models q$
  - $\{p\}\ a\ \{q\}$ is true if the predicate p being true implies that q is true after action a

Lemma (Klavins, 5.2) Let $P = (I, C)$ be a program and p and q be predicates. If for all commands c in C we have $\{p\}\ c\ \{q\}$ then $P \models p$ **co** q.

- If p is true then any action in the program P that can be applied in the current state leaves q true
- Thus to check if p **co** q is true for a program, check each possible action

Proof. Using the lemma, it suffices to check that for all commands c in C we have $\{p\}\ c\ \{q\}$, where $p = q = z_i < z_{i+1}$. So, we need to show that if $z_i < z_{i+1}$ then any command that changes z_i or z_{i+1} leaves the order unchanged. Two cases: i moves or i+1 moves. For the first case, {p} c {q} becomes

$$z_i < z_{i+1} \wedge (z_i < x_{\alpha(i)} \wedge z_i < z_{i+1} - \delta : z_i' = z_i + \delta) \quad \Longrightarrow \quad z_i' < z_{i+1}'$$

From the definition of the guarded command, this is true. Similar for second case.

RoboFlag Simulation

Specification 1 Red Robot Dynamics: $\Pi_{red}(i)$

Initial:

$x_i \in [min, max] \wedge y_i > \max$

Clauses:

$y_i - \delta > 0 \; : \; y_i' = y_i - \delta$

Specification 2 Blue Robot Control: $\Pi_{blue}(i)$

Initial:

$z_i \in [min, m$

Clauses:

$z_i < x_{\alpha(i)} \wedge$

$z_i > x_{\alpha(i)} \wedge z_i > z_{i-1} + 2\delta \; : \; z_i' = z_i - \delta$

Exercise 1: create a model of the RoboFlag drill in Promela and verify correctness using SPIN model checker

Exercise 2: create a specification for the RoboFlag drill and *synthesize* a (decentralized) protocol to solve it [later]

Specificatio

Initial:

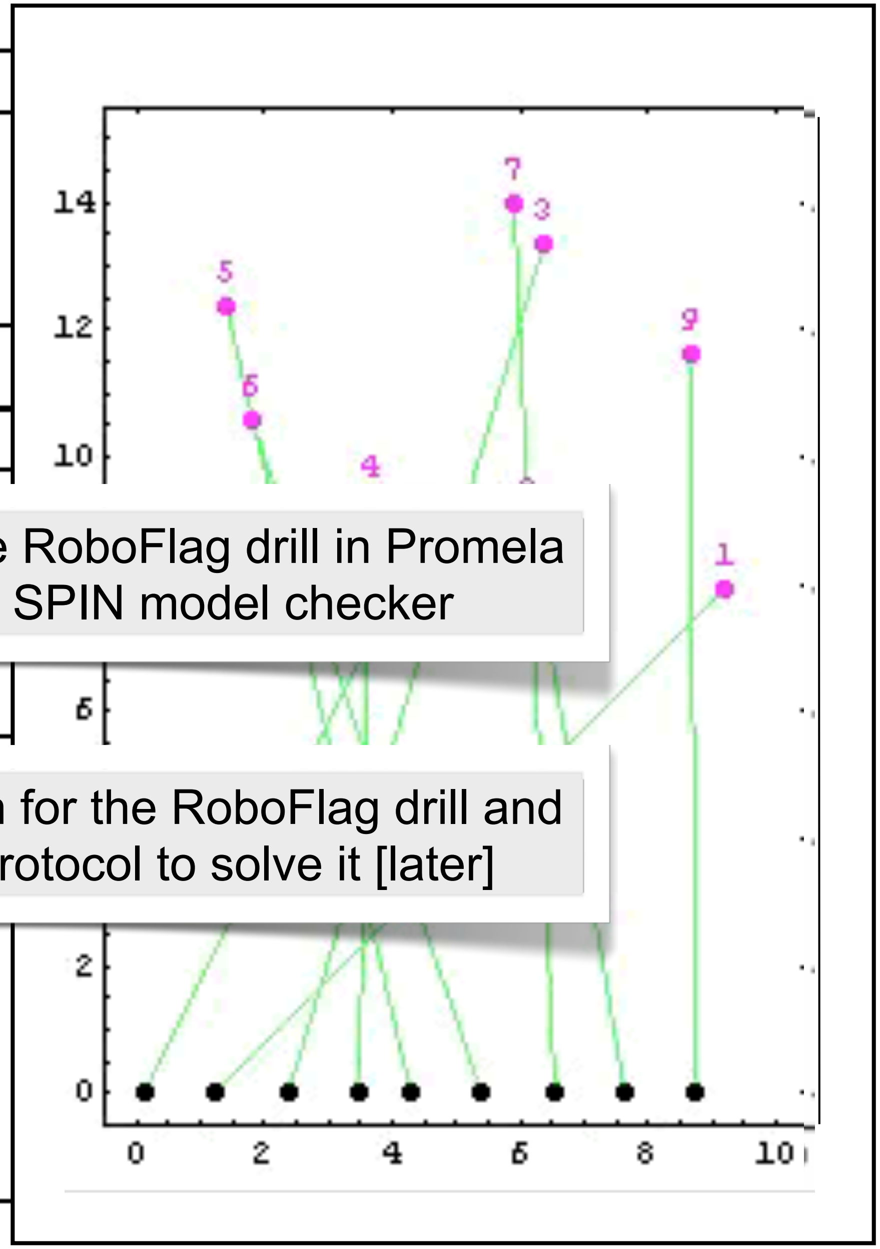
α is a bijection from $\{1, ..., n\}$ to $\{1, ..., n\}$.

Clauses:

$switch_{1,2} \; : \; (\alpha(1)', \alpha(2)') = (\alpha(2), \alpha(1))$

...

$switch_{n-1,n} \; : \; (\alpha(n-1)', \alpha(n)') = (\alpha(n), \alpha(n-1))$

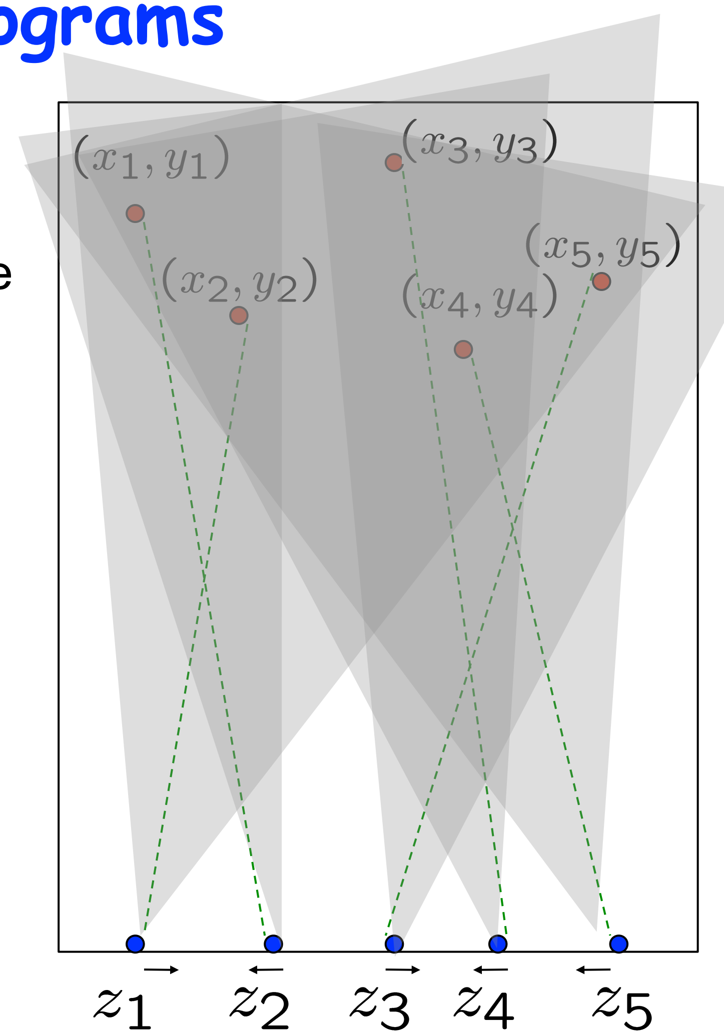


Del Vecchio, Klavins and M
Automatica, 2006

Observation of CCL Programs

Goal: determine assignments by watching motion

- Assume CCL program describing protocol is known
- Brute force: enumerate all N! possibilities and eliminate cases that are inconsistent with motion (over time)



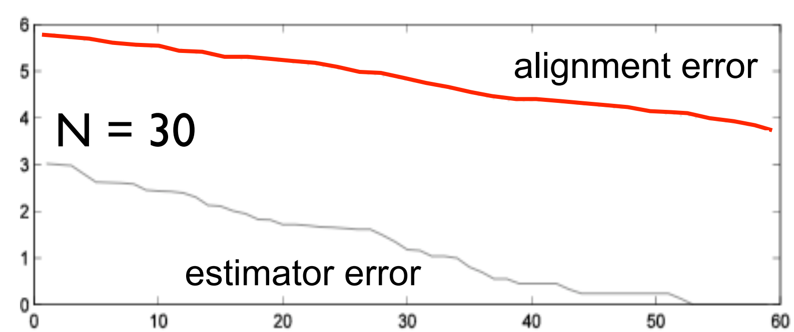
Alternative approach: exploit structure

- Keep track of upper and lower bounds for each z_i
- Can show this provides a *partial order* on sets of possible assignments
- Extended CCL update law preserves the order:

$$\tilde{f}([l, u]) = [\tilde{f}(l), \tilde{f}(u)]$$ ⇒ fast computation

General case: observers for hybrid systems

- Construct a partial order on discrete states
- Extend CCL program to provide order-isomorphic map (always possible with power set)
- Can construct observer if system is observable: predict + correct on upper/lower bounds (fast)



Real-World Example: Lost Wingman Protocol

Control T33 to follow F15 and to execute "lost wingman" during simulated communications loss.

sec.ccl

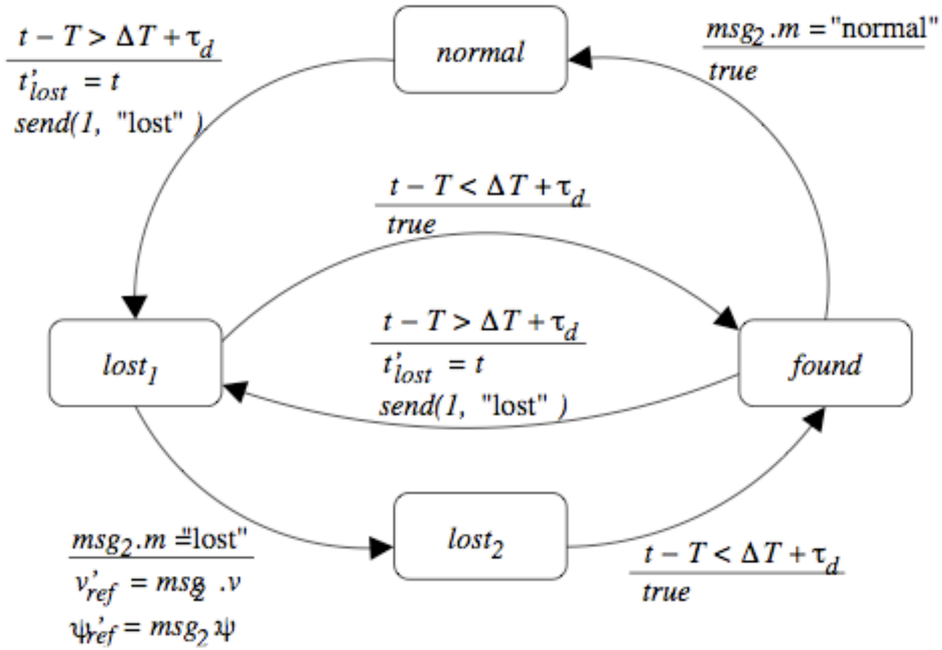
actual F-15 software

```
// F15 closed-loop dynamics and software
program F15() := F15_statemachine( 0 )                                          // updates mode
               + F15_data_sender()                         sharing u1, q1      // sends state to T33
               + F15_controller()                          sharing u1, q1
               + F15_dynamics( {0.0, 0.0, 0.0, 30.0} ) sharing u1, q1;
```

model of dynamics

```
// T33 closed-loop dynamics and software
program T33() := T33_statemachine( 0 )                                          // updates mode
               + T33_data_receiver()         sharing u2, q2, q1_hat            // gets state from F15
               + T33_controller( 50, pi/6, {4.47, 0.32, 1.28 , 0.32 } ) sharing u2, q2, q1_hat
               + T33_dynamics( {-100.0, -40.0, 0.0, 30.0} ) sharing u2, q2;

program main() := F15()
                + T33() sharing q1, q2  // shares q1 just so draw_scene() works
                + draw_scene(400, 400) sharing q1, q2
                + sim_clock()
                + window_manager();
```

CCL Specification for Lost Wingman



CCL-based protocol

- High speed link used to communicate state information between aircraft
- Low speed link used to confirm status
- Update timers based on when we last sent/received data
- Change modes if data is not received within expected period (plus delay)

Program P_{comm}

Initial $T_s = t_0 \wedge T \in (T_s, T_s + \Delta T]$

Commands

$$c_{data} \equiv t > T \wedge data\_on : T' \in (T_s + \Delta T, T_s + \Delta T + \tau_d] \wedge T'_s = T_s + \Delta T$$

$$c_{msg,1} \equiv in(1) : msg'_1 = recv(1)$$

$$c_{msg,2} \equiv in(2) : msg'_2 = recv(2)$$

Program T_{sm}

Initial $m_2 = n$

Commands

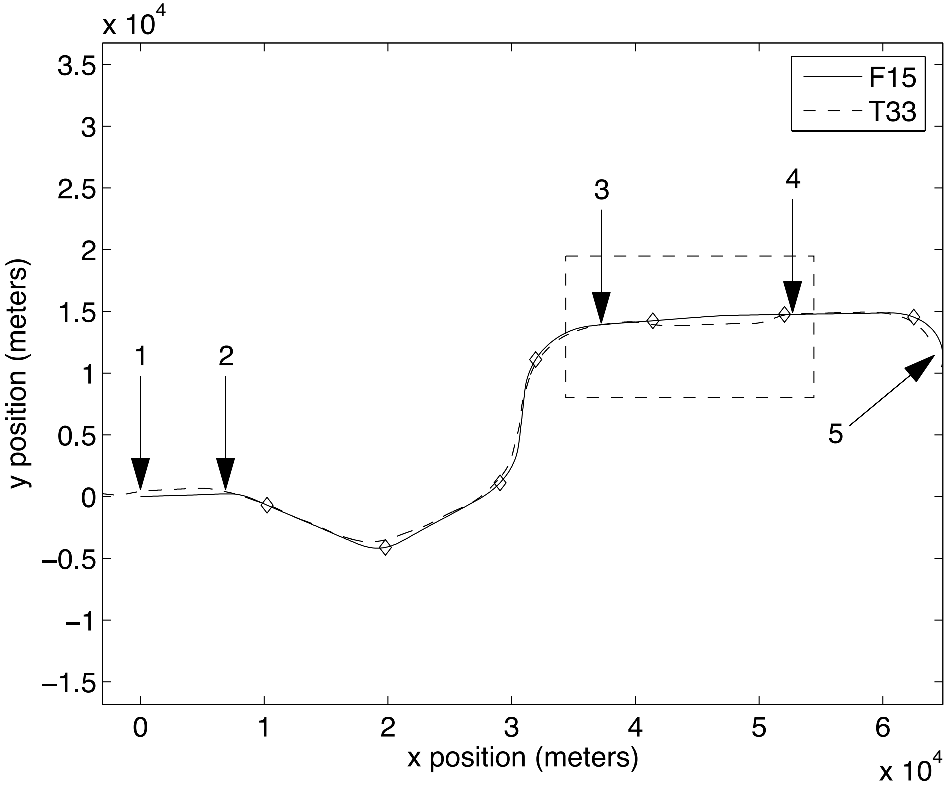
$$c_{lost} \equiv m_2 \in \{n, f\} \wedge t - T > \Delta T + \tau_d : m'_2 = l_1 \wedge t'_{lost} = t \wedge send(1, \text{“lost”})$$

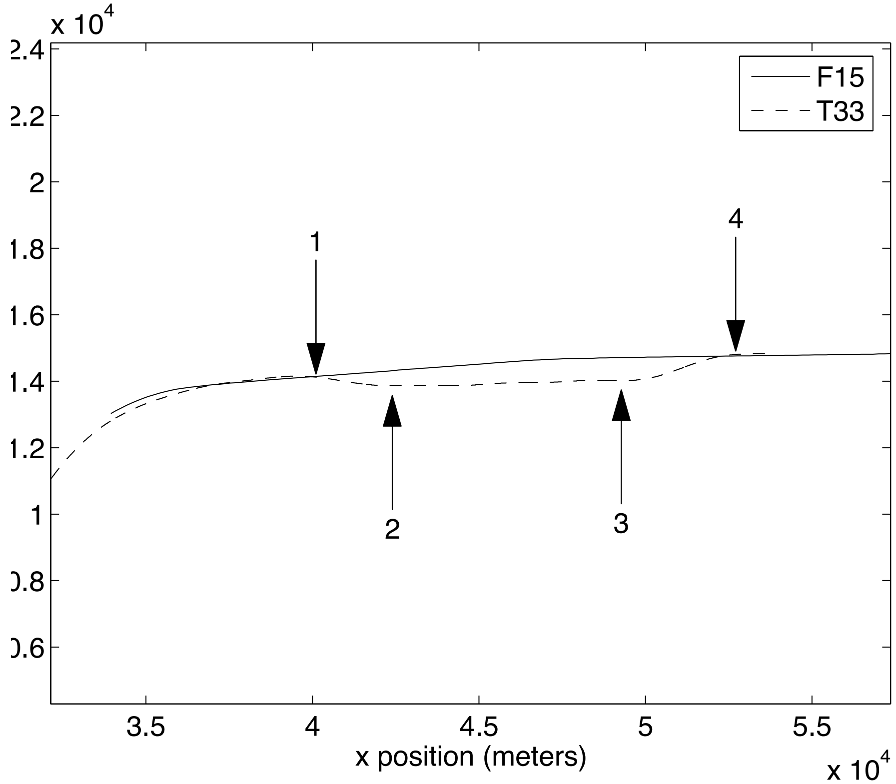
$$c_{found} \equiv m_2 \in \{l_1, l_2\} \wedge t - T < \Delta T + \tau_d : m'_2 = f$$

$$c_{lost2} \equiv m_2 = l_1 \wedge msg_2.m = \text{“lost”} : m'_2 = l_2 \wedge v'_{ref} = msg_2.v \wedge \psi'_{ref} = msg_2.\psi$$

...

Flight Test Results





Event timeline (right figure)

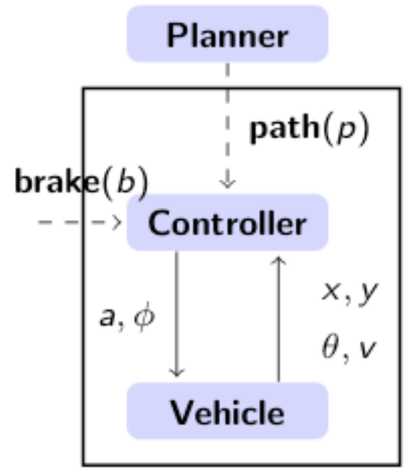
- Event 1: communications lost; T-33 executes tight turn; signals lost comms (slow link)
- Event 2: F-15 confirms communication lost message received
- Event 3: communications restored; T-33 requests rejoin (granted)
- Event 4: rejoin confirmed; return to normal operation

Wongpiromsarn, Mitra and M
HSCC09

Periodically Controlled Hybrid Automata (PCHA)

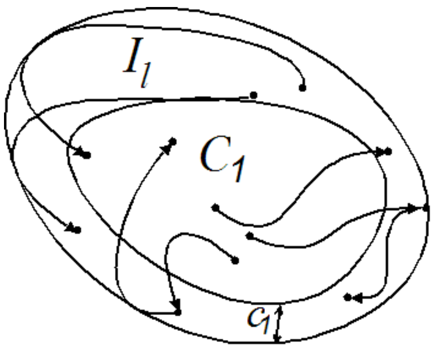
PCHA setup

- Continuous dynamics with piecewise constant inputs
- Controller executes with period T ∈ [Δ_1, Δ_2]
- Input commands are received asynchronously
- Execution consists of trajectory segments + discrete updates
- Verify safety (avoid collisions) + performance (turn corner)



Proof technique: verify invariant (safe) set via barrier functions

- Let I be an (safe) set specified by a set of functions $F_i(x) \geq 0$
- Step 1: show that the control action renders I invariant
- Step 2: show that between updates we can bound the continuous trajectories to live within appropriate sets
- Step 3: show progress by moving between nested collection of invariant sets $I_1 \rightarrow I_2$, etc



Remarks

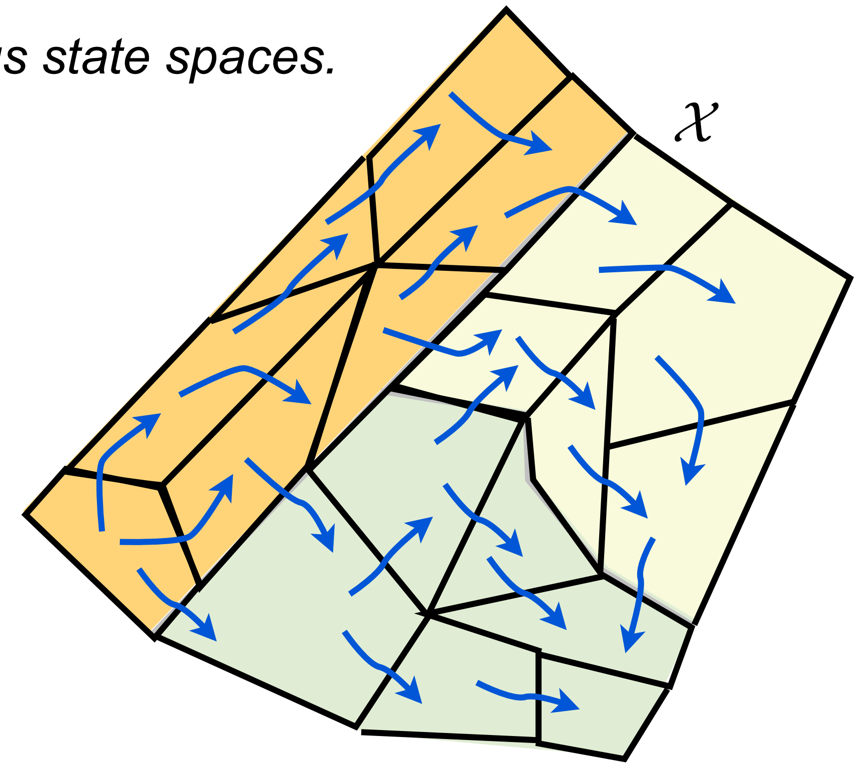
- Can use this to show that settings in Alice were not properly chosen; modified settings lead to proper operation (after the fact)
- Very difficult to find invariant sets (barrier functions) for given control system...

Verification of hybrid systems: Overview

Why not directly use model checking?

- Model checking applied to finite transitions systems
- Exhaustively search for counterexamples....
  - if found, property does not hold.
  - if there is no counterexample in all possible executions, the property is verified.

Exhaustive search is not possible over continuous state spaces.



Approaches for hybrid system verification:

1. Construct finite-state approximations and apply model checking
   - Preserve the meaning of the properties, i.e., proposition preserving partitions
   - Use “over”- or “under”-approximations
2. Deductive verification
   - Construct Lyapunov-type certificates
   - Account for the discrete jumps in the construction of the certificate
3. Explicitly construct the set of reachable states
   - Limited classes of temporal properties (e.g., reachability and safety)
   - Not covered in this lecture