Lecture 5
Deductive Verification of Control Protocols

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

EECI, 20 Mar 2013

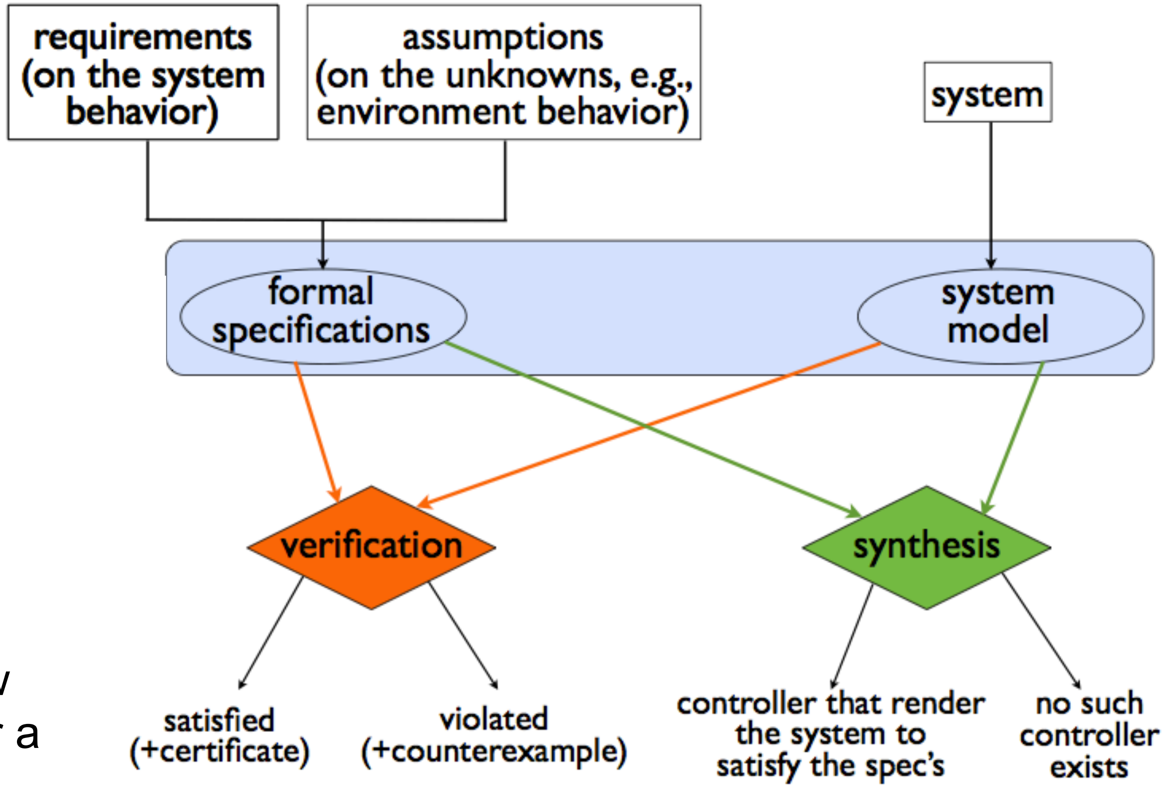
Outline

- 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**

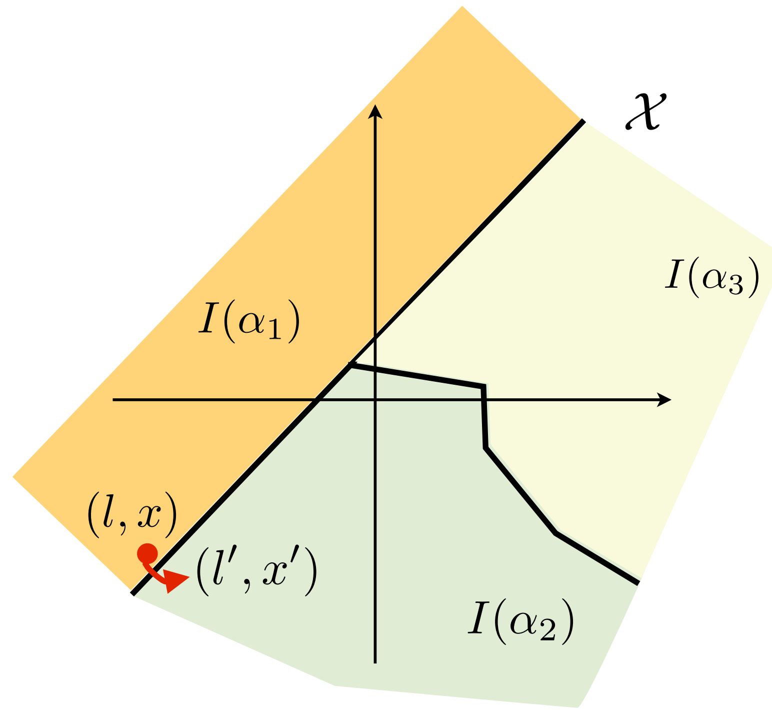
$$u^i(x, \alpha) = u^i(x^i, \alpha^i, y^{-i}, \alpha^{-i})$$
$$y^{-i} = \{y^{j_1}, \ldots, y^{j_{m_i}}\}$$
$$j_k \in \mathcal{N}^i \quad m_i = |\mathcal{N}^i|$$

- Similar structure for role update

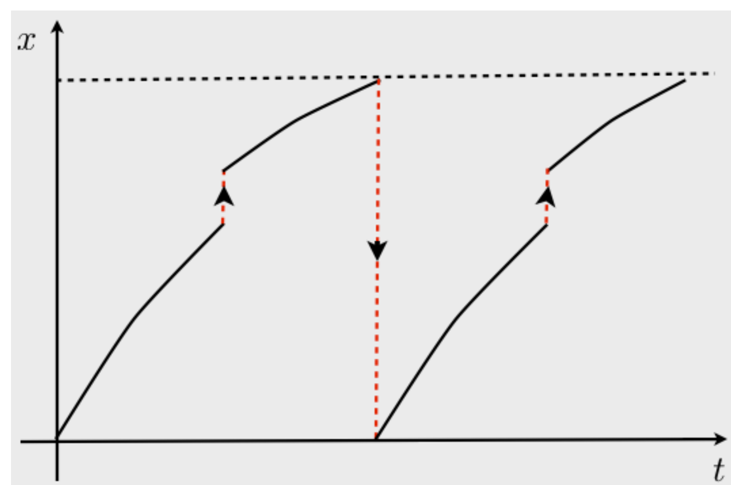
A (simple) hybrid system model

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



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

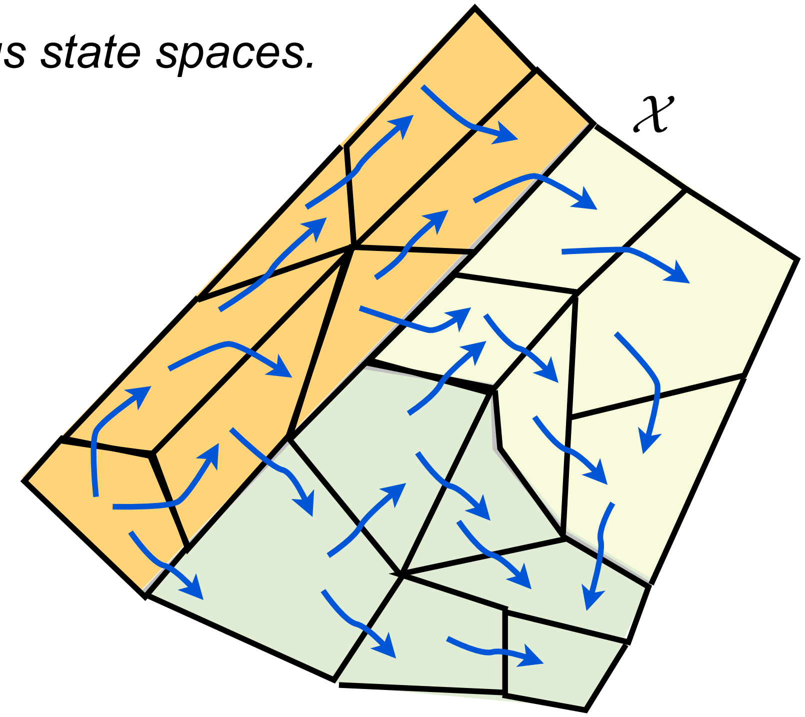


Verification of hybrid systems: Overview

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

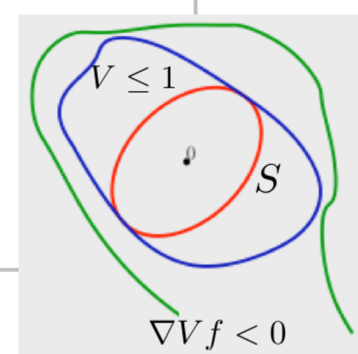
Question 1 (a system analysis question):
Given $S \subset \mathbb{R}^n$, is S invariant and $S \subseteq \mathcal{R}$?

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

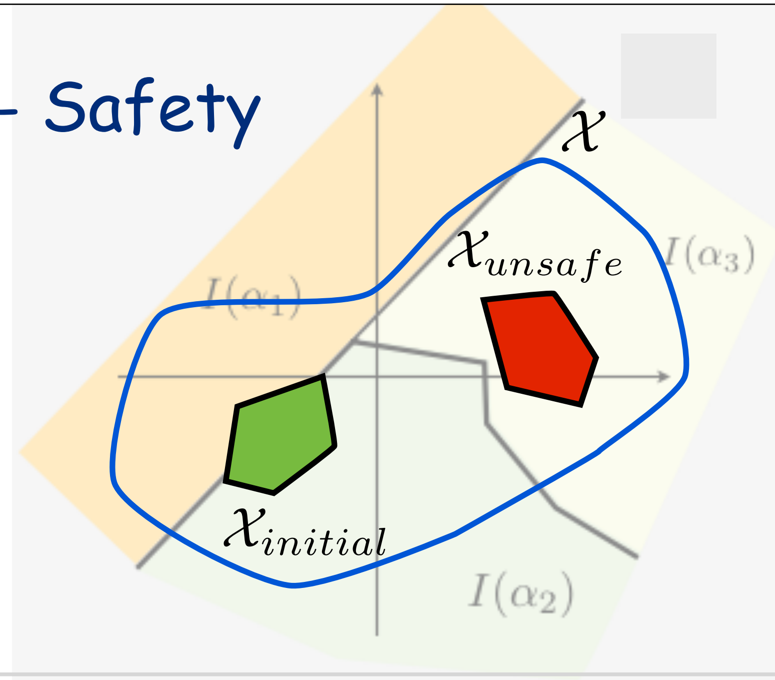


Yes to Question 2 → Yes to Question 1.

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:

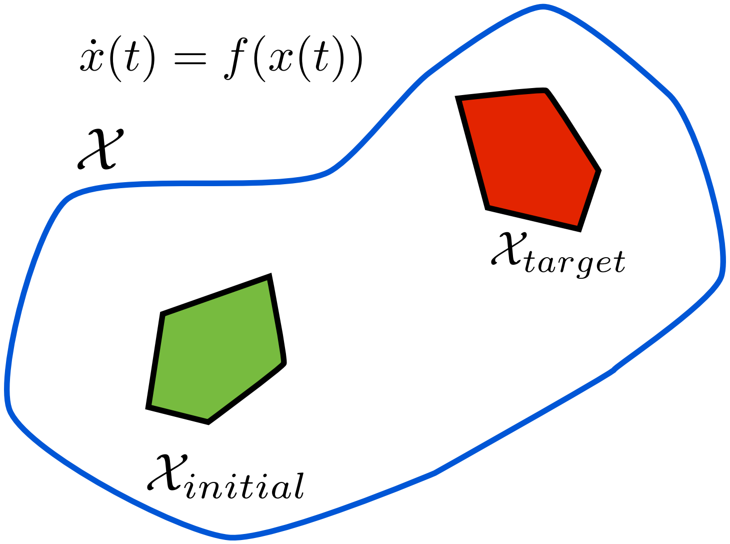
$$H = (\mathcal{X}, L, X_0, I, F, \mathcal{T})$$

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

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

Then, the safety property holds.

Barrier Certificates - Eventuality



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

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

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

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 $\mathcal{X}_{target}$ →

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

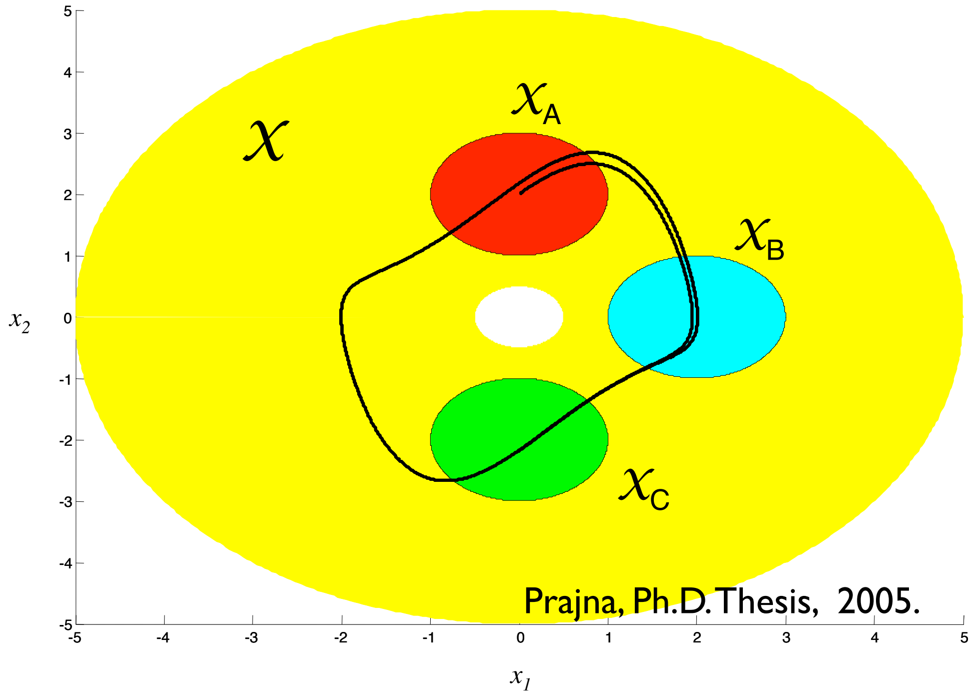
leave $\mathcal{X} \backslash \mathcal{X}_{target}$ in finite time →

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

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)

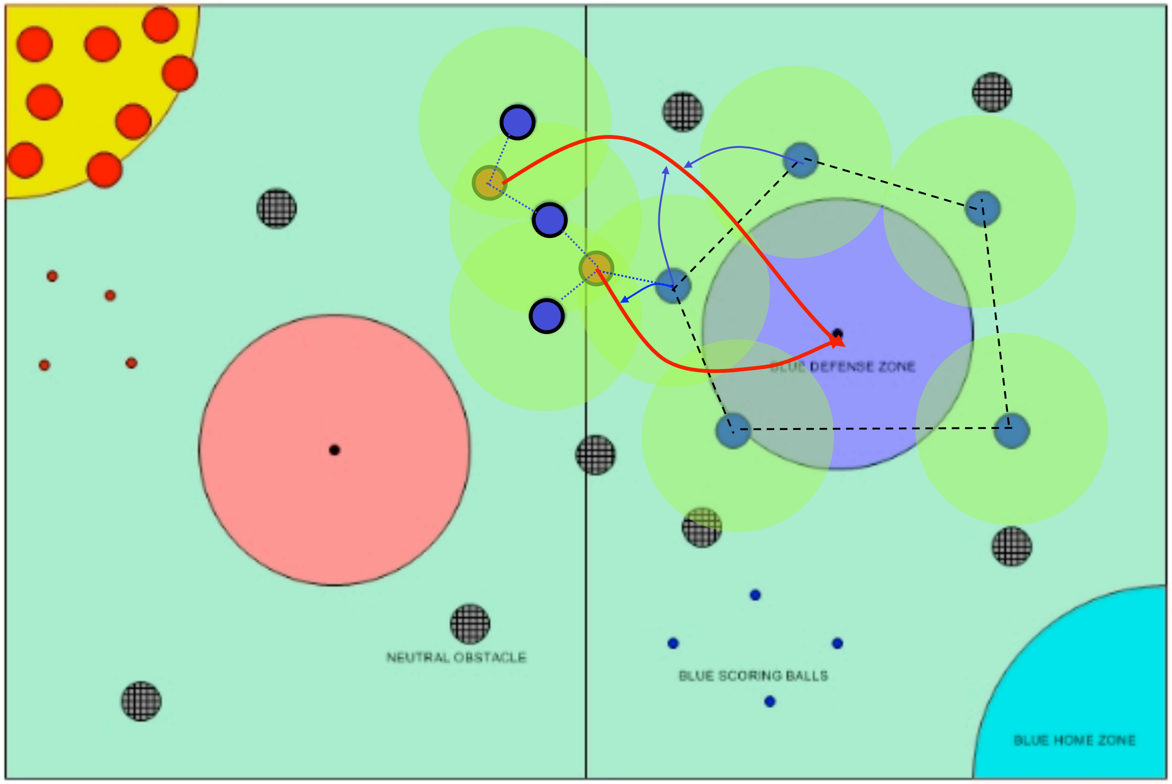
Step 4: Construct polynomial certificates based on SDP solutions

Generally taken care of by software packages.

More details: see references on course web page

- Basic message: using barrier certificates we can verify some LTL-like properties for hybrid dynamical systems
- Problems: properties are somewhat limited; computations become intractable quickly

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)

Klavins
CDC, 03

Distributed Decision Making: RoboFlag Drill

Task description

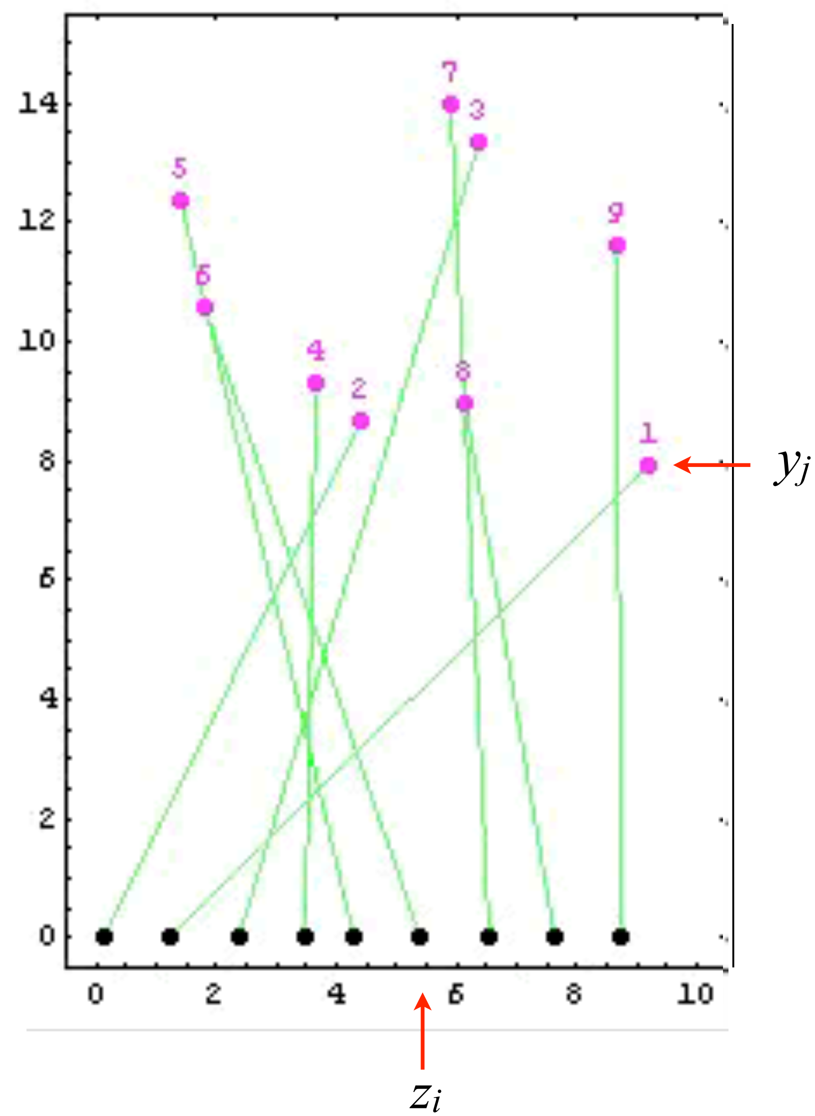
- Incoming robots should be blocked by defending robots
- Incoming robots are assigned randomly to whoever is free
- Defending robots must move to block, but cannot run into or cross over others
- Allow robots to communicate with left and right neighbors and switch assignments

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?



CCL: Computation and Control Language

Formal Language for Provably Correct Control Protocols

$P(k_1,k_2) := \{$

  initializers

  $guard_1 : rule_1$

  $guard_2 : rule_2$

  ...

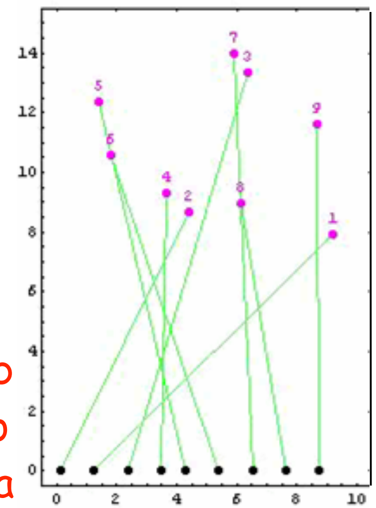
$\}$

"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.

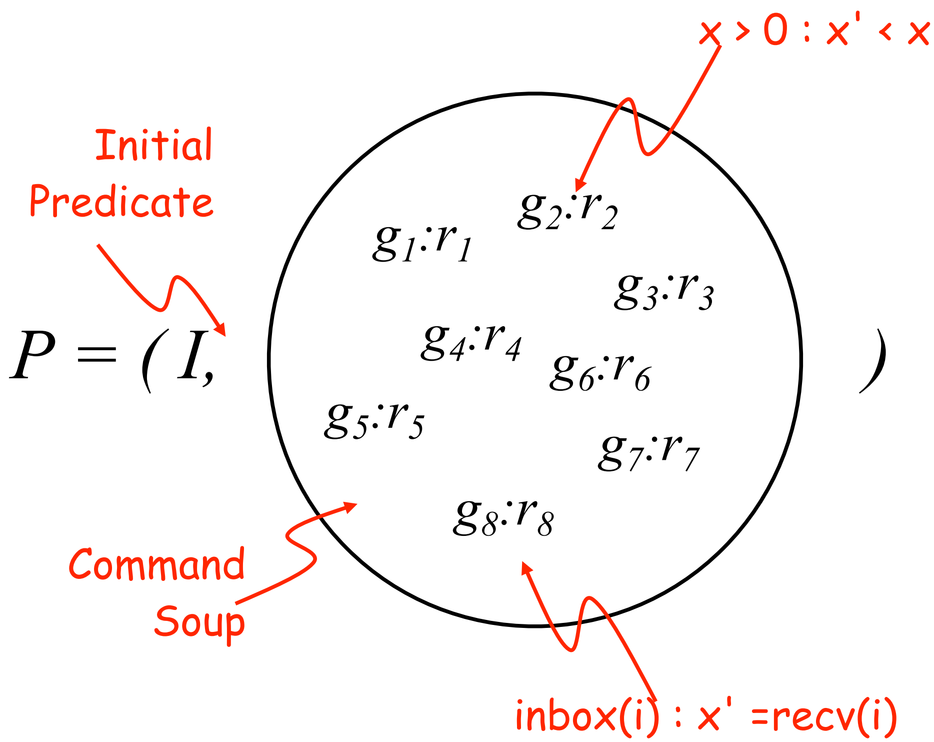
Formal Results

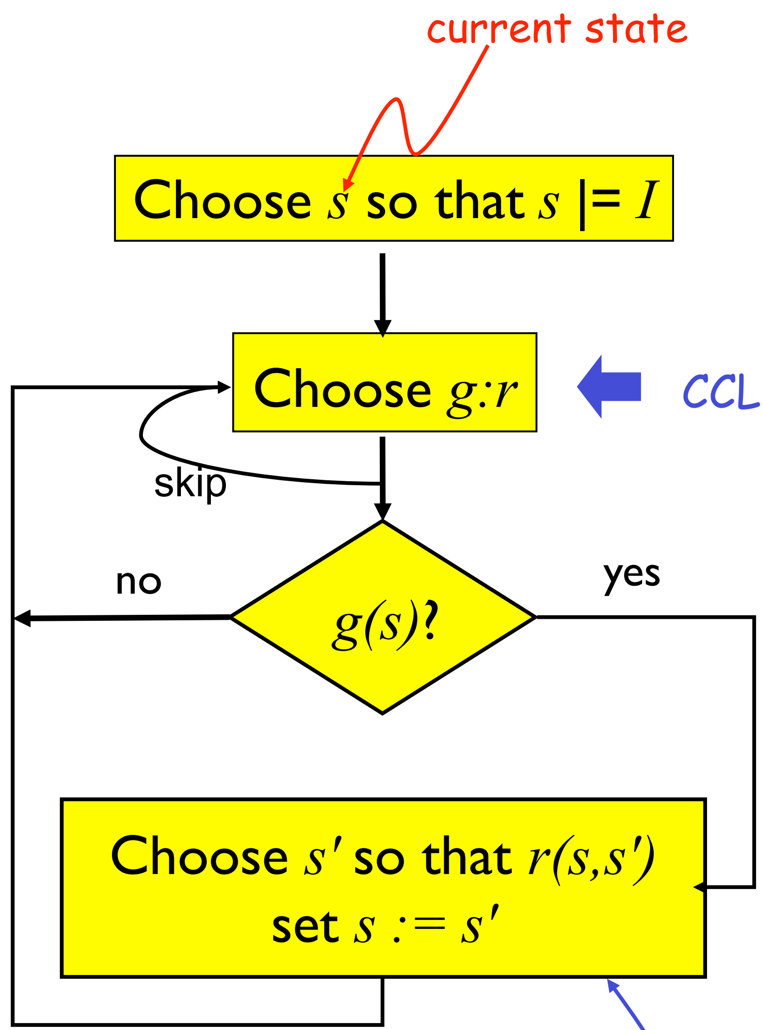
Formal semantics in transition systems and temporal logic. *RoboFlag* drill formalized and basic algorithms verified.

Automated Verification

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

Guarded Command Programs

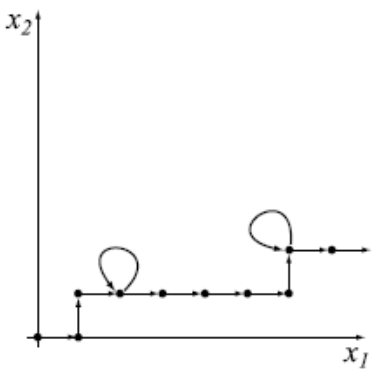




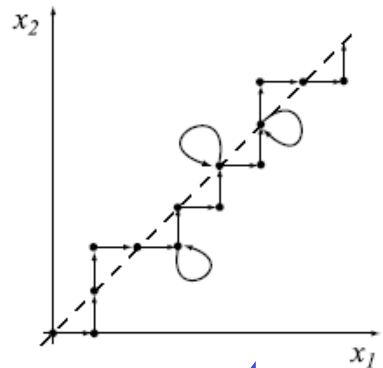
- Non-deterministic execution schedule models concurrency
- Easy to reason about programs
- Guarded commands = update functions

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

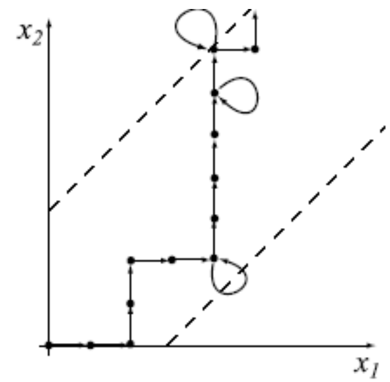
Scheduling and Composition



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	$true \;:\; x_i' = x_i + 1$

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

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

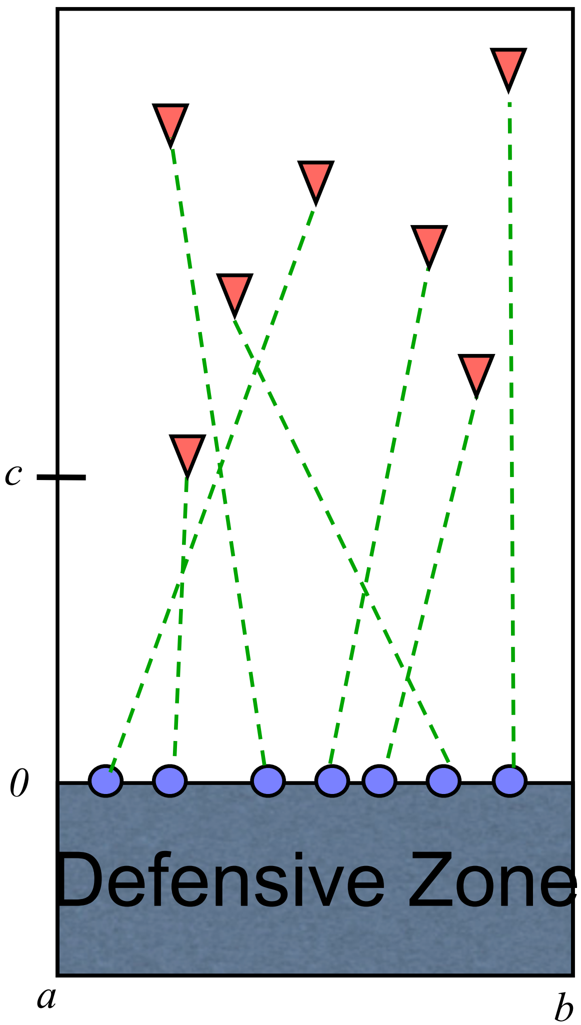
Thm: $SYNCH(1) \subseteq EPOCH \subseteq SYNCH(2) \subseteq SYNCH(3) \subseteq \ldots \subseteq UNITY$.

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



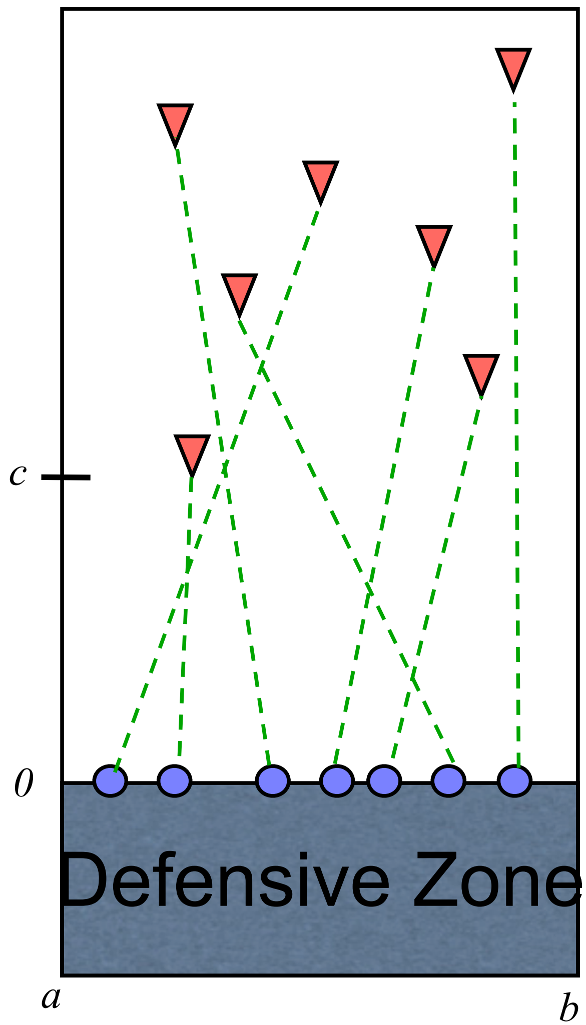
$Red(i)$	
Initial	$x_i \in [a,b] \wedge y_i > c$
Commands	$y_i > \delta \;:\; y_i' = y_i - \delta$
	$y_i \leq \delta \;:\; x_i' \in [a,b] \wedge y_i > c$

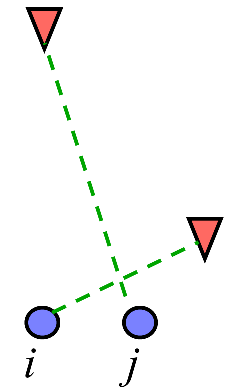
$$P_{Red}(n) = +_{i=1}^{n} Red(i)$$

$Blue(i)$	
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, $

Clauses:

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

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

Project 2: create a model of the RoboFlag drill in Promela and verify correctness using SPIN model checker

Specificatio

Initial:

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

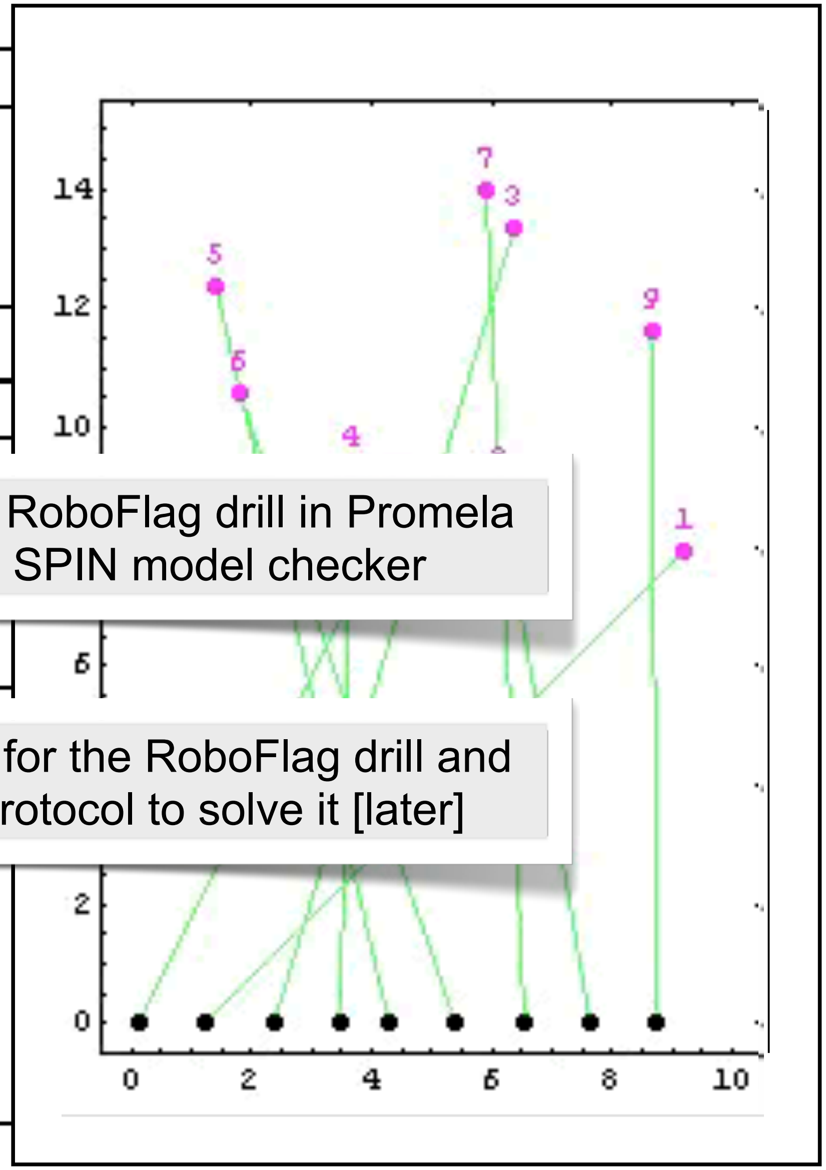
Clauses:

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

...

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

Project 3: create a specification for the RoboFlag drill and *synthesize* a (decentralized) protocol to solve it [later]



Planner Stack

Burdick et al, 2007

Mission Planner performs high level decision-making

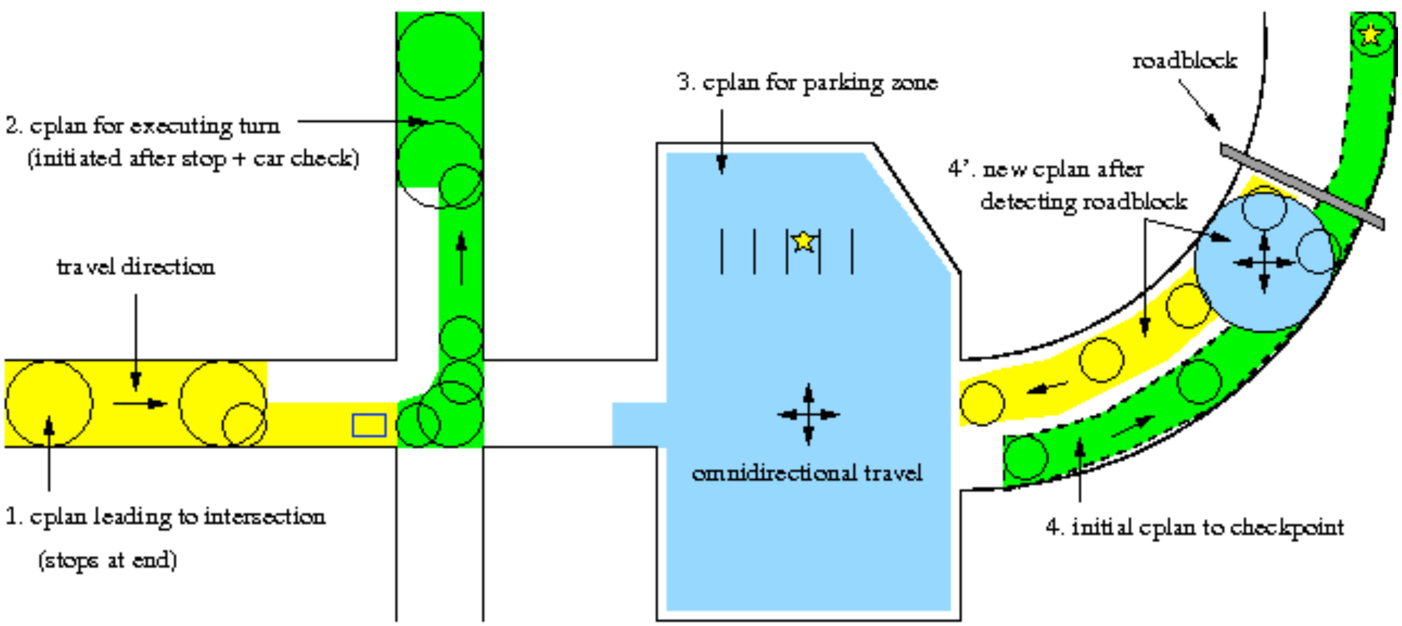
- Graph search for best routes; replan if routes are blocked

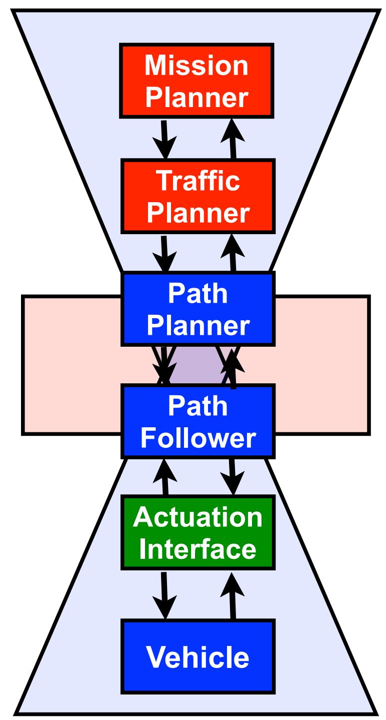
Traffic Planner handles rules of the road

- Control execution of path following & planning (multi-point turns)
- Encode traffic rules - when can we change lanes, proceed thru intersection, etc

Path Planner/Path Follower generate trajectories and track them

- Optimized trajectory generation + PID control (w/ anti-windup)
- Substantial control logic to handle failures, command interface, etc



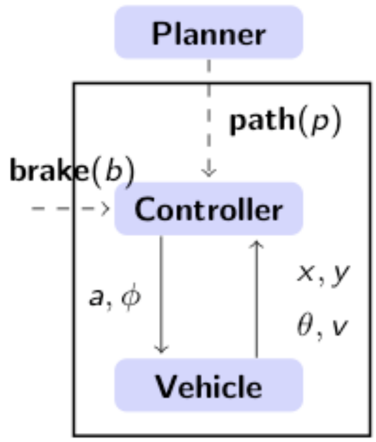


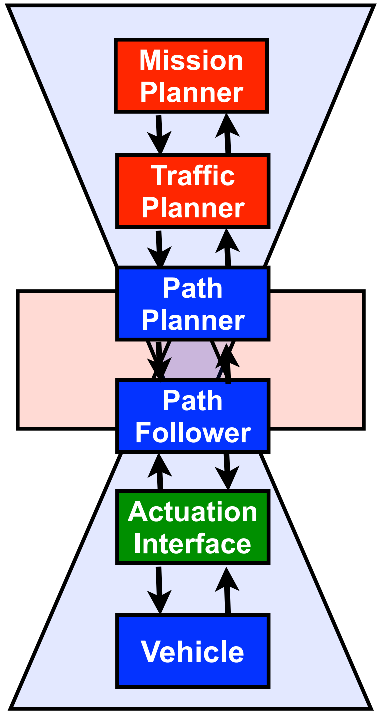
Wongpiromsarn, Mitra and M
HSCC09

Verification of Periodically Controlled Hybrid Systems

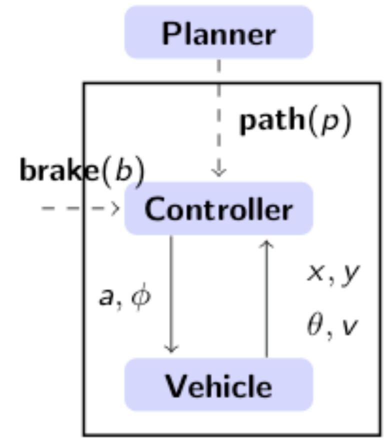
Hybrid system: continuous dynamics + discrete updates

- Vehicle
  - Captures the state (position, orientation and velocity) of the vehicle.
  - Specifies the dynamics of the autonomous ground vehicle with respect to the acceleration and the angle of the steering wheel.
  - Limits the magnitude of the steering input to ϕ_{max}.
- Controller
  - Receives the state of the vehicle, a path and an externally triggered brake input.
  - Periodically computes the input steering
  - Restricts the steering angle to δv for mechanical protection of the steering.
  - Sampling period: $\Delta \in R_+$.
- Desired properties
  - (Safety) At all reachable states, the deviation of the vehicle from the current path is upper-bounded by e_{max}.
  - (Progress) The vehicle reaches successive waypoints.





Periodically Controlled Hybrid Automata (PCHA)

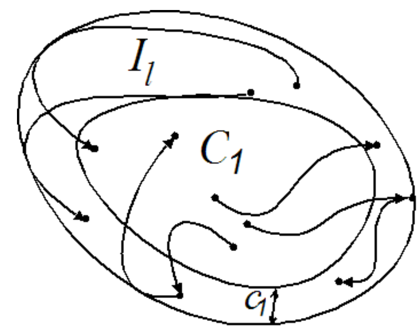


PCHA setup

- Continuous dynamics with piecewise constant inputs
- Controller executes with period $T \in [\Delta_1, \Delta_2]$
- 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

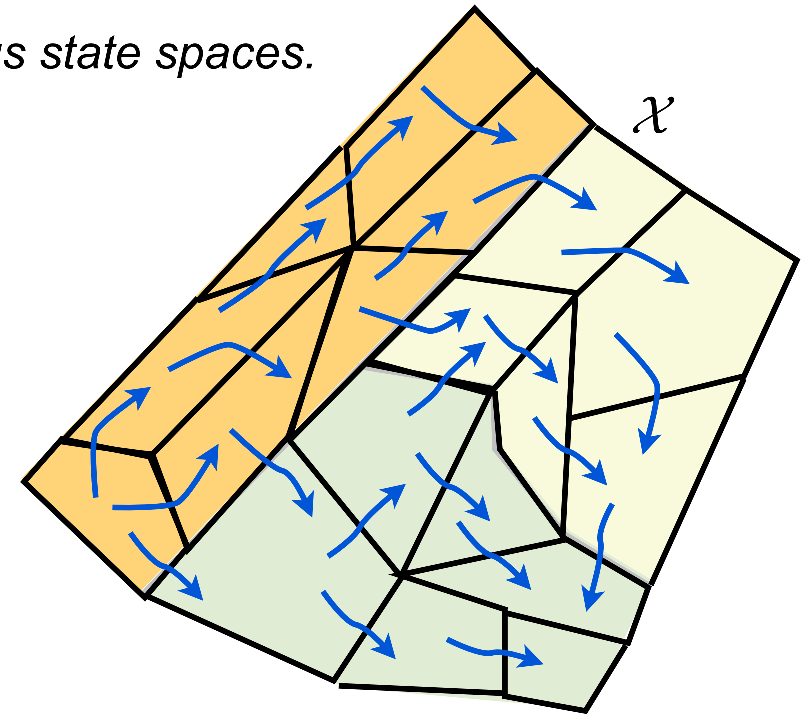
- Can use this to show that settings in Alice were not properly chosen; modified settings lead to proper operation (after the fact)
- Very difficult to find invariant sets (barrier functions) for given control system...

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