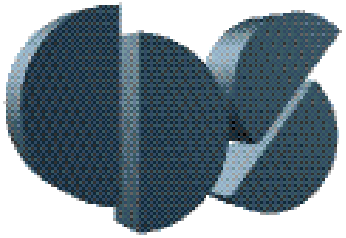
Lecture 5
(Manual) Design and Verification of Control Protocols



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

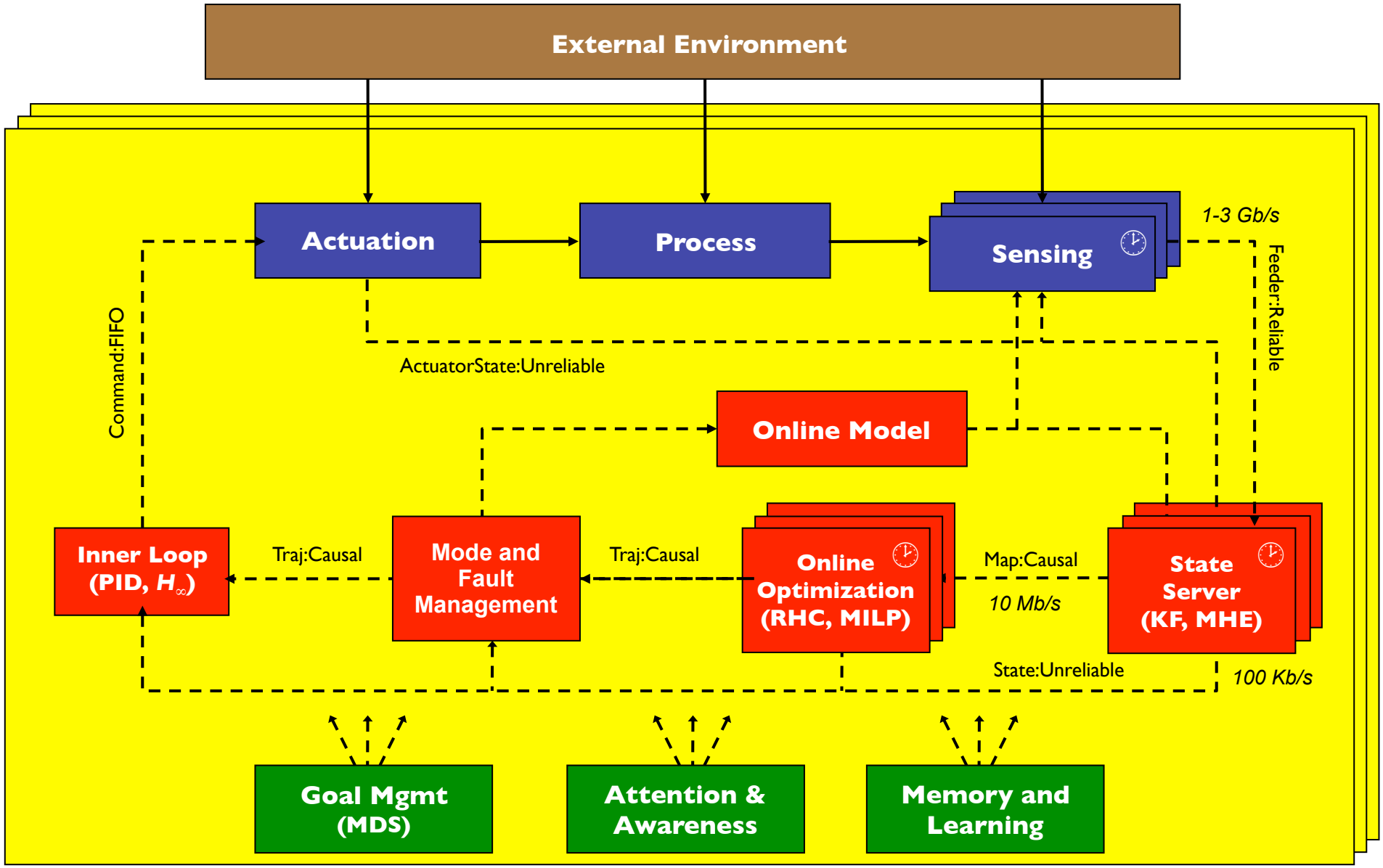
AFRL, 25 April 2012

Outline

- Review: networked control systems and cooperative control systems
- Asynchronous execution / group messaging systems (virtual synchrony)
- Verification of async control protocols for multi-agent, cooperative control
- Applications of model checking to Alice's actuation interface

Networked Control Systems

(following P. R. Kumar)



(Multi-) Vehicle Control Systems Framework

Agent dynamics - continuous

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

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

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

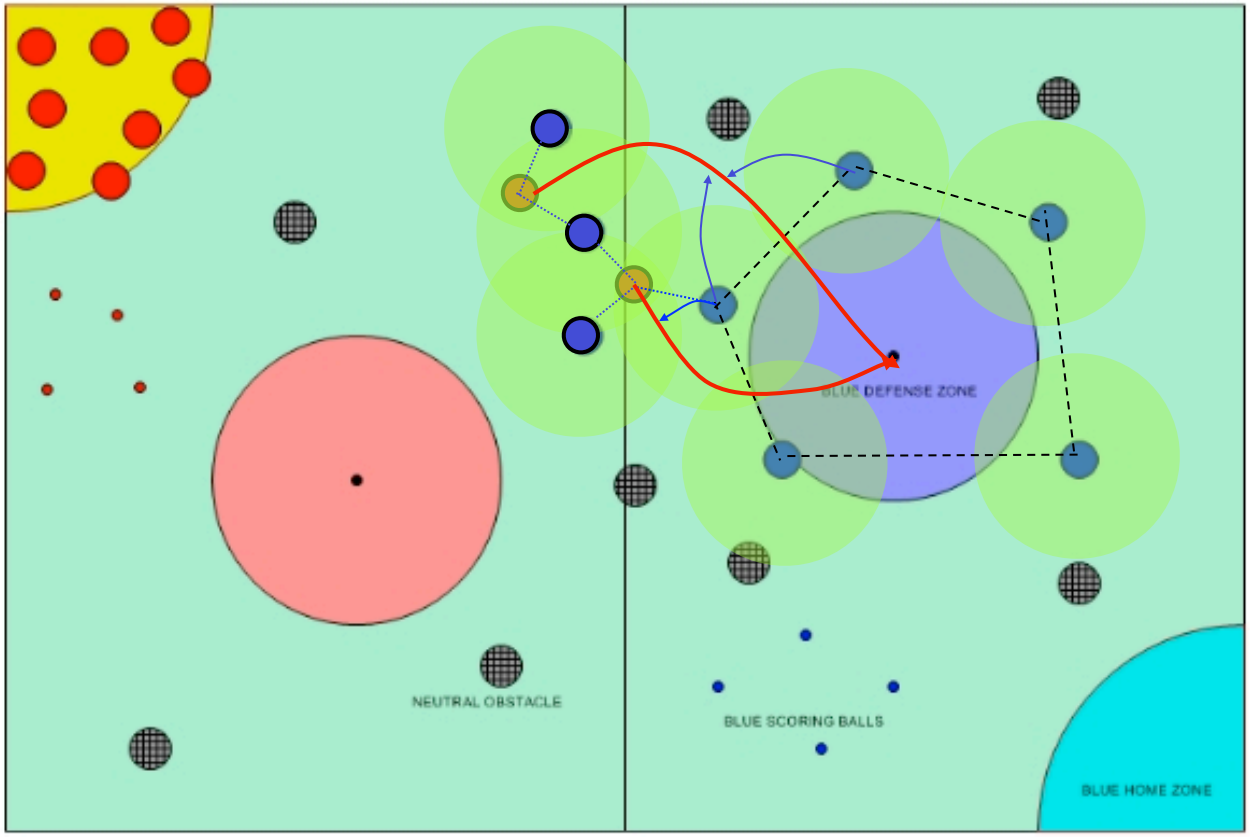
$$\alpha^{i\,\prime} = \begin{cases} r^i_j(x, \alpha) & g(x, \alpha) = \text{true} \\ \text{unchanged} & \text{otherwise.} \end{cases}$$

***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

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

Task description

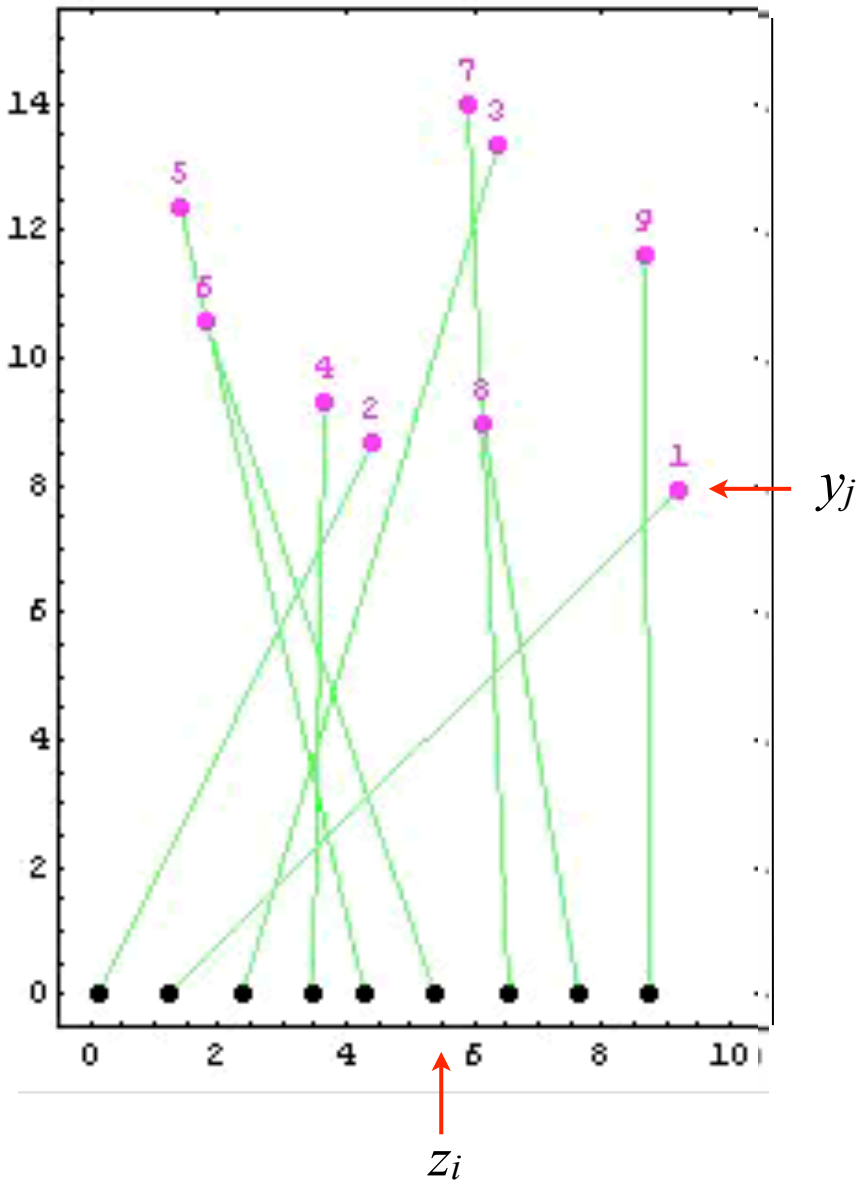
- Incoming robots should be blocked by defending robots
- Incoming robots are assigned randomly to whoever is free
- Defending robots must move to block, but cannot run into or cross over others
- Allow robots to communicate with left and right neighbors and switch assignments

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

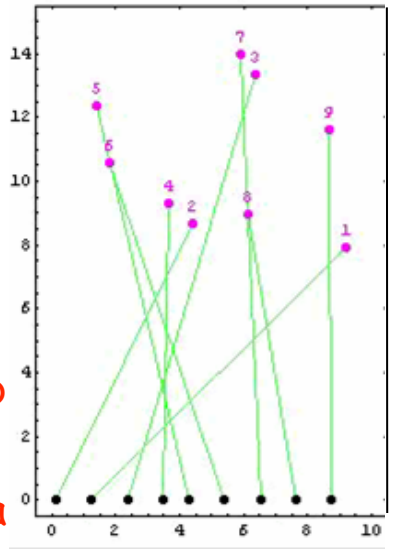
```
P(k1,k2) := {
   initializers
   guard1:rule1
   guard2:rule2
   ...
}
```

"soup" of guarded commands

composition = union

non-shared variab remain local to component progra

```
S(k1,k2):=P(k1,k2)+C(k1+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.

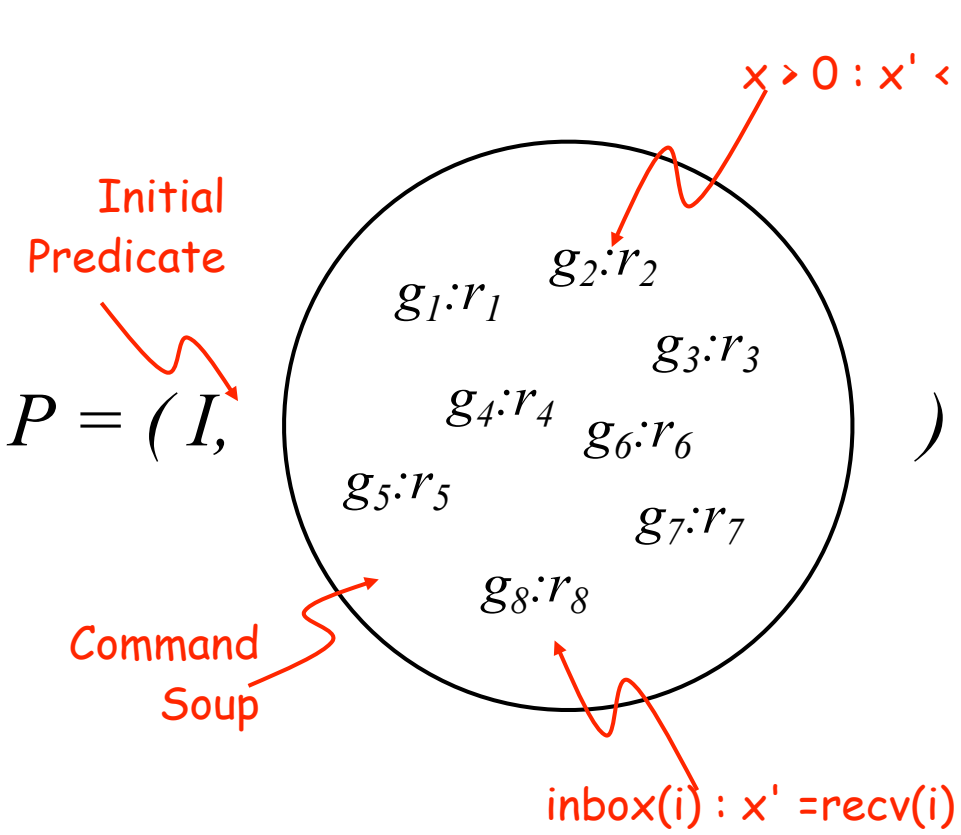
Formal Results

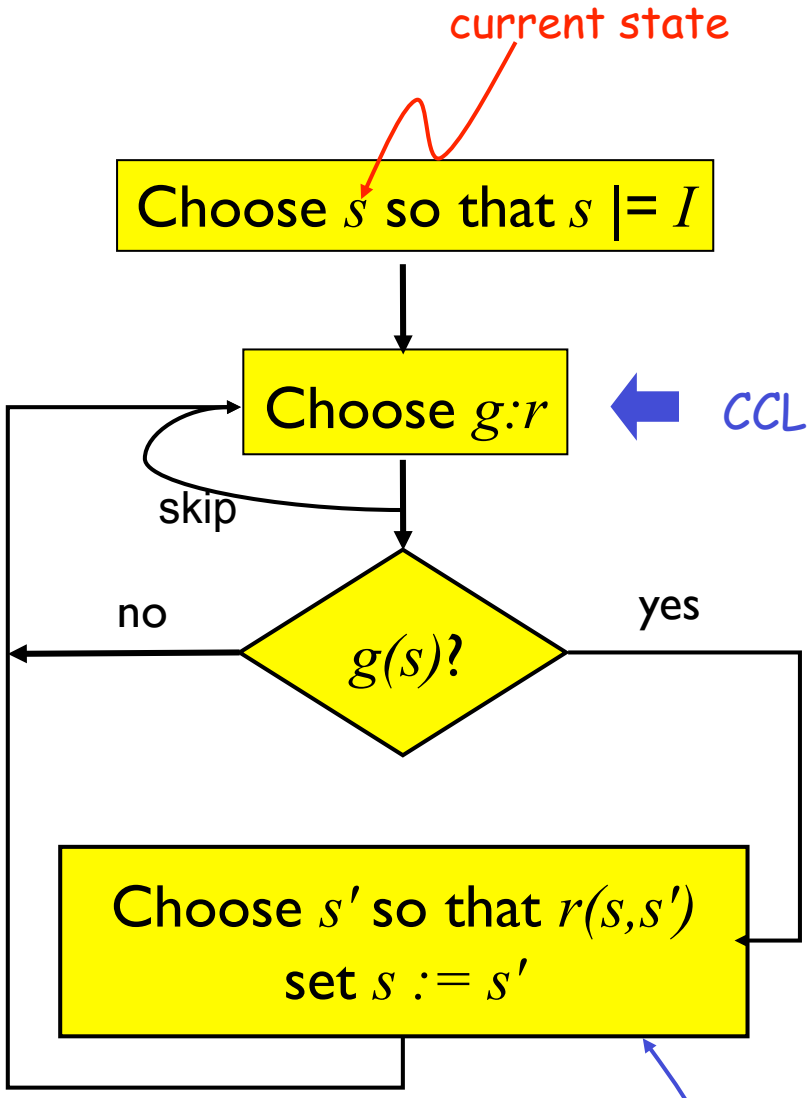
Formal semantics in transition systems and temporal logic. *RoboFlag* drill formalized and basic algorithms verified.

Automated Verification

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

Guarded Command Programs

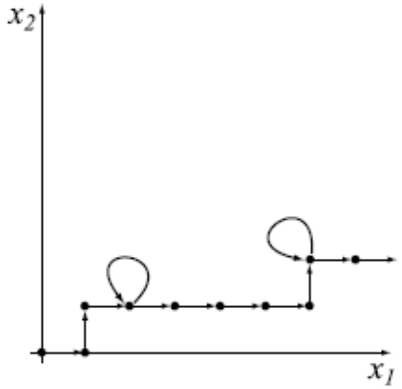




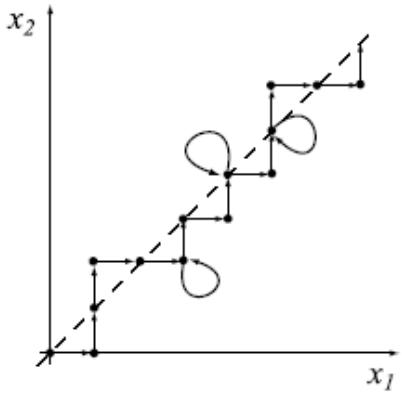
- Non-deterministic execution schedule models concurrency
- Easy to reason about programs
- Guarded commands = update functions

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

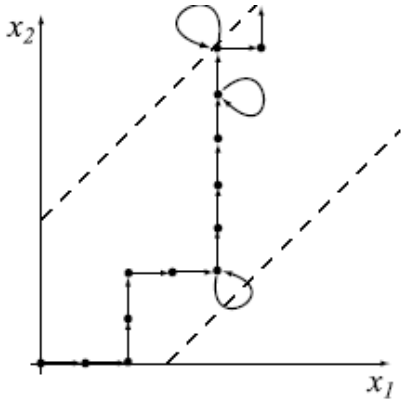
Scheduling and Composition



UNITY
Each command must be executed infinitely often.



EPOCH
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.

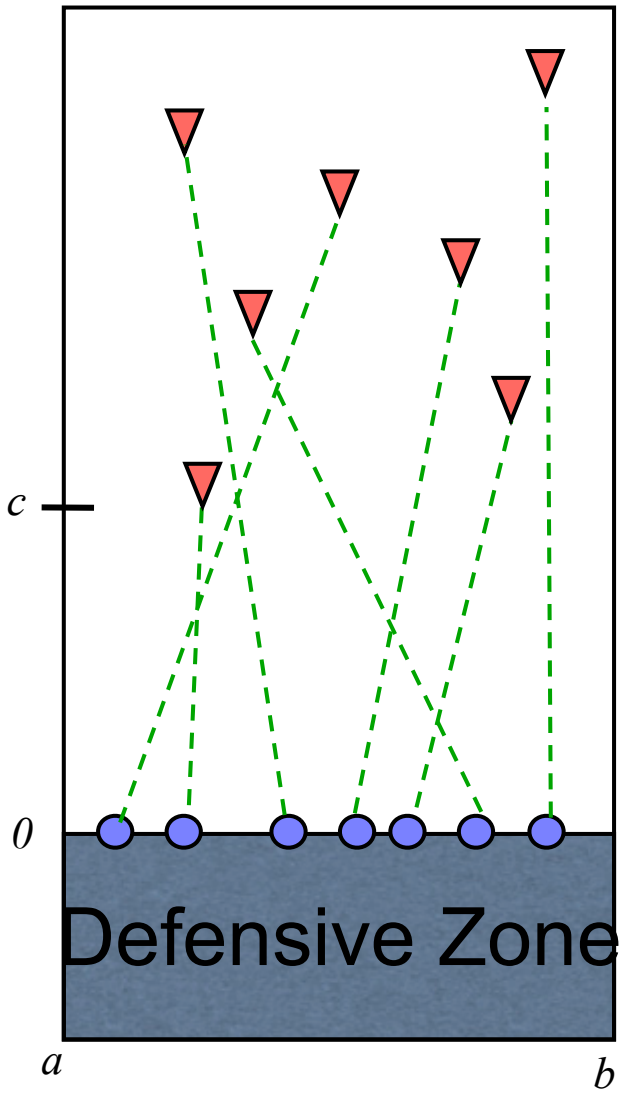
$P(i)$	
Initial	$x_i = 0$
Commands	$true \ : \ x_i' = x_i + 1$

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

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

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

Example: RoboFlag Drill



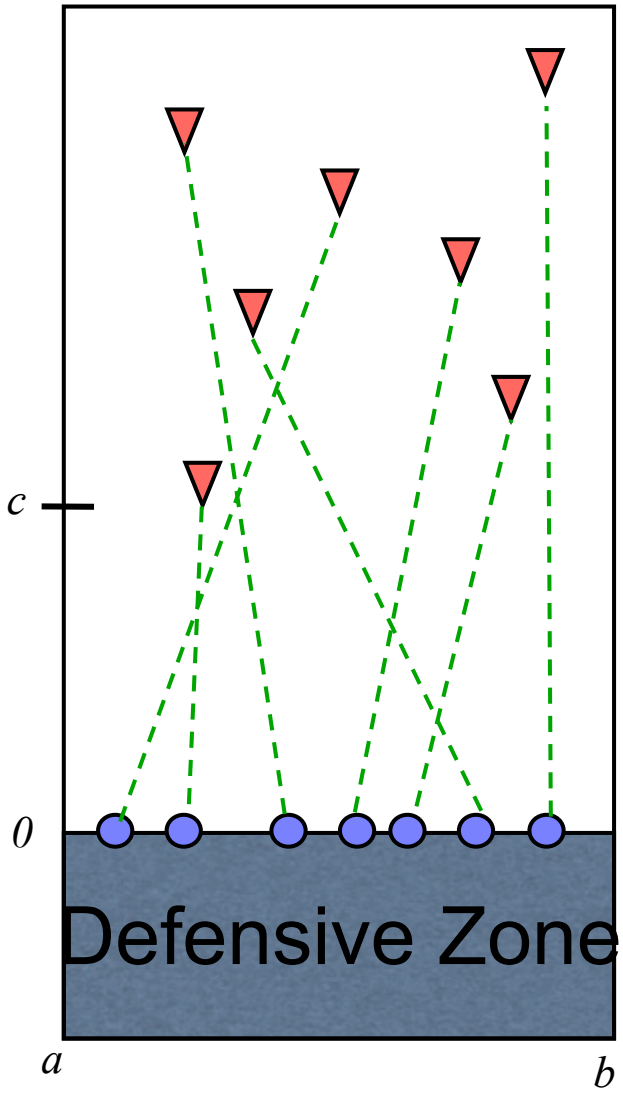
$Red(i)$	
Initial	$x_i \in [a,b] \wedge y_i > c$
Commands	$y_i > \delta \;:\; y_i' = y_i - \delta$
	$y_i \leq \delta \;:\; x_i' \in [a,b] \wedge y_i > c$

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

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

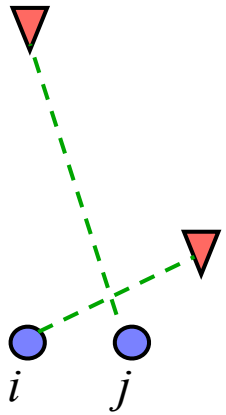
$$P_{Blue}(n) = +_{i=1}^{n} Blue(i)$$

RoboFlag Control Protocol



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

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



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

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

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

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)

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

Robots are "far enough" apart.

- skip $\forall v \ . \ v' = v$ state remains unchanged
- p **co** q $\Box(p \rightarrow [(\bigcirc q \vee \text{skip}) \wedge \Diamond \bigcirc q])$
  "if p is true, then next time state changes, q will be true"

"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:

$z_i \in [min, r$

Clauses:

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

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

Specificatio

Initial:

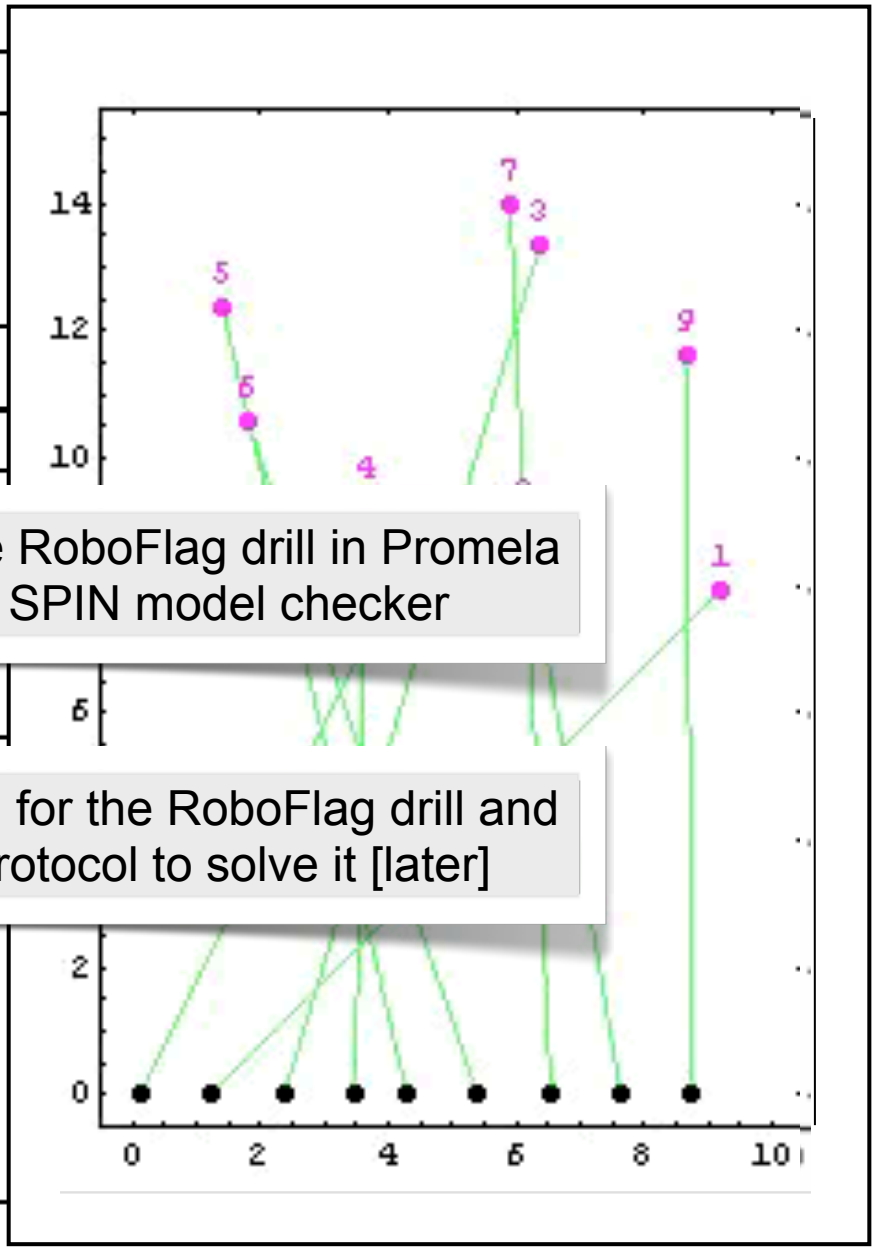
α is a bijection from $\{1, ..., n\}$ to $\{1, ..., n\}$.

Clauses:

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

...

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



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]

Planner Stack

Mission Planner performs high level decision-making

Burdick et al, 2007

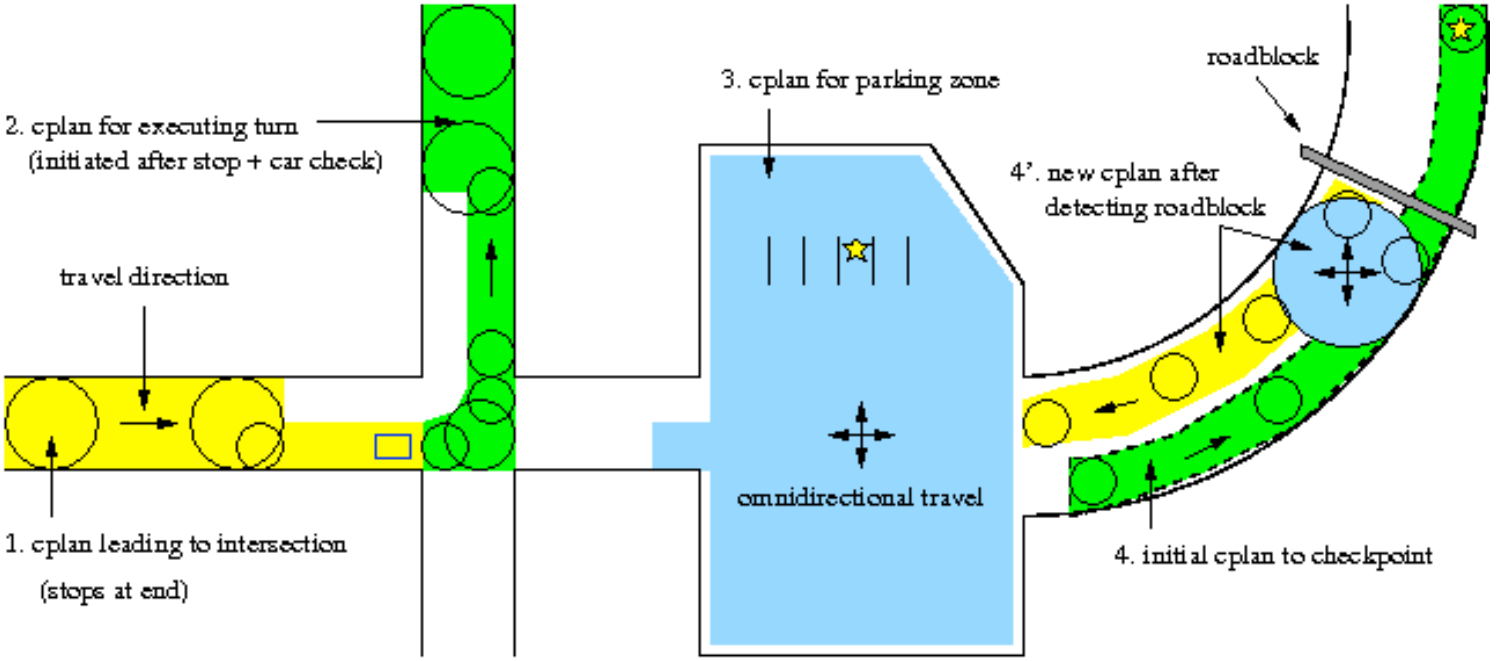
- Graph search for best routes; replan if routes are blocked

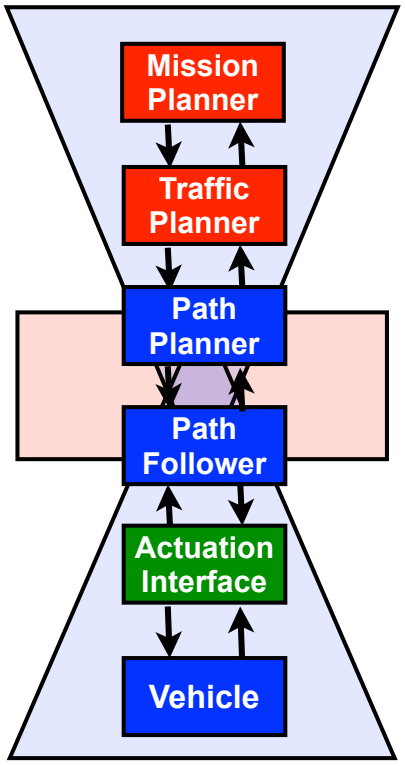
Traffic Planner handles rules of the road

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

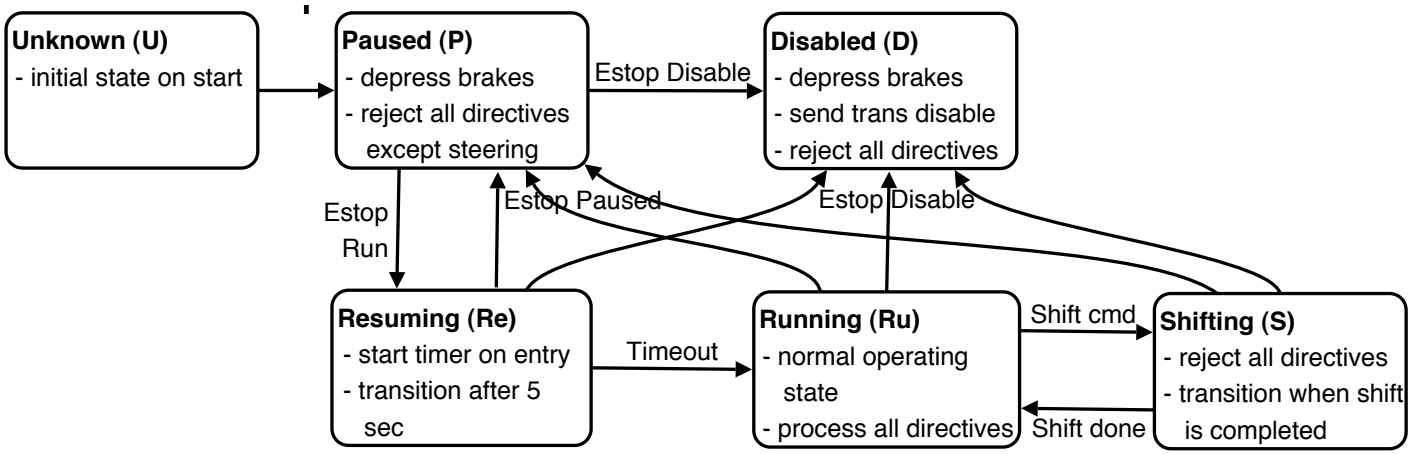
Path Planner/Path Follower generate trajectories and track them

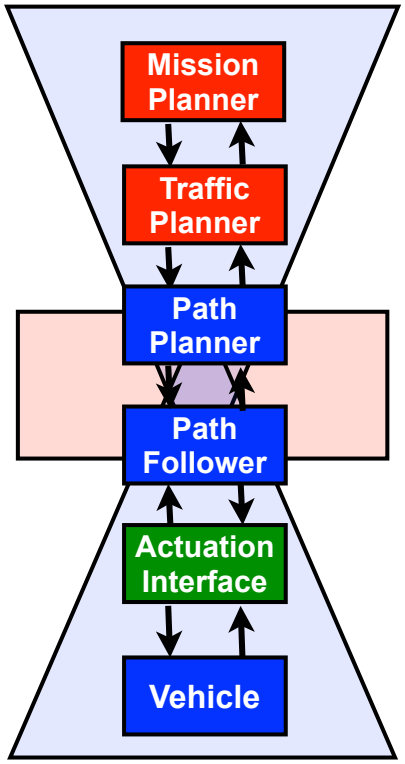
- Optimized trajectory generation + PID control (w/ anti-windup)
- Substantial control logic to handle failures, command interface, etc





Alice Actuation Interface (adrive) Logic





Desired properties

- If *Estop Disable* is received, gcdrive state will be *Disabled* and acceleration will be 'full brake' forever
- *Estop Paused*: if not disabled, gcdrive will eventually enter *Paused* state and acceleration will be 'full brake' (not forever)
- *Estop Run*: if not *Disabled*, gcdrive will eventually be *Running* or *Resuming* (or receive another pause or disable command)
- If *Resuming*, eventually *Running* (or receive another pause or disable)
- If current mode is *Disabled, Paused, Resuming* or *Shifting*, full brake is commanded
- After receiving an *Estop Pause*, vehicle may resume operation 5 seconds after run is received (suffices to show that we transition from *Resuming* to *Running* via *Timeout*)
- …

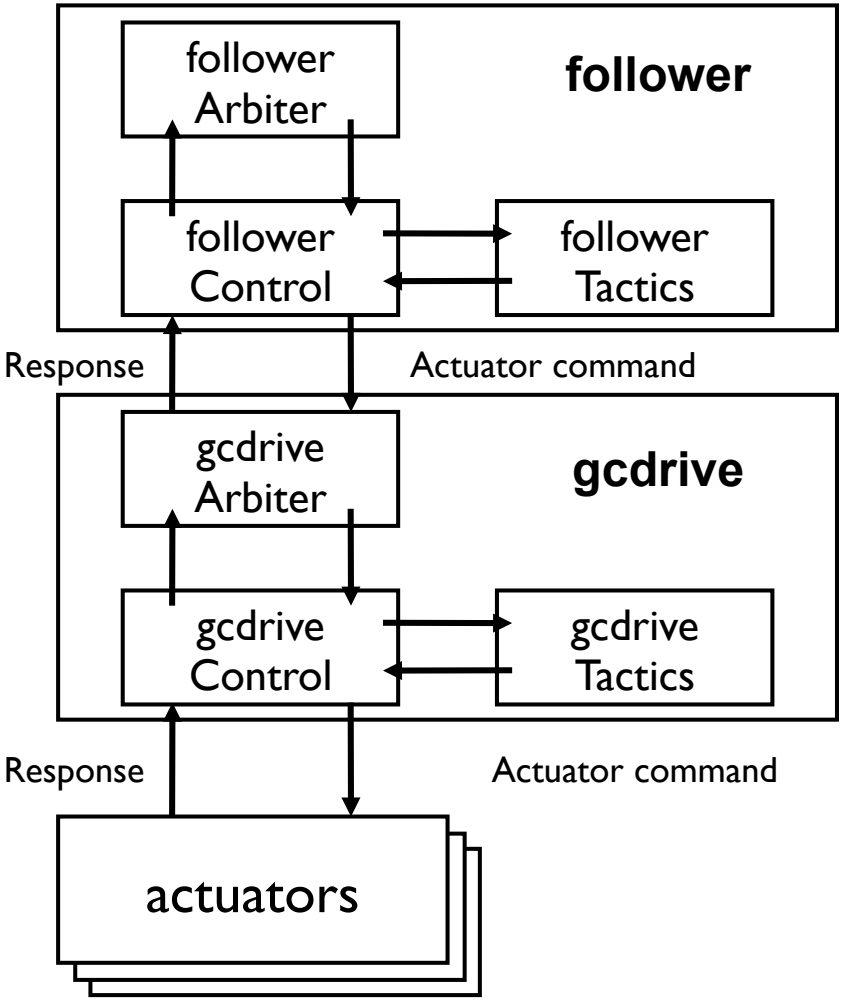
Exercise 1: verify correctness using SPIN model checker

DGC Example: Changing Gear

Wongpiromsarn and M
CDC 2008

Verify that we can't drive while shifting or drive in the wrong gear

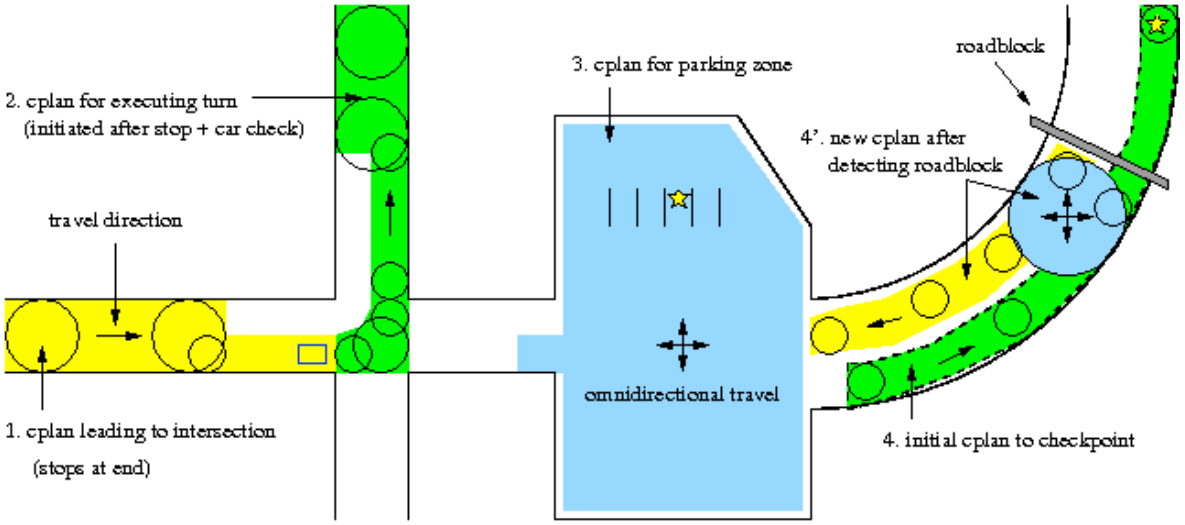
- Five component: follower Control, gcdrive Arbiter, gcdrive Control, actuators and network
- Construct temporal logic models for each component (including network)

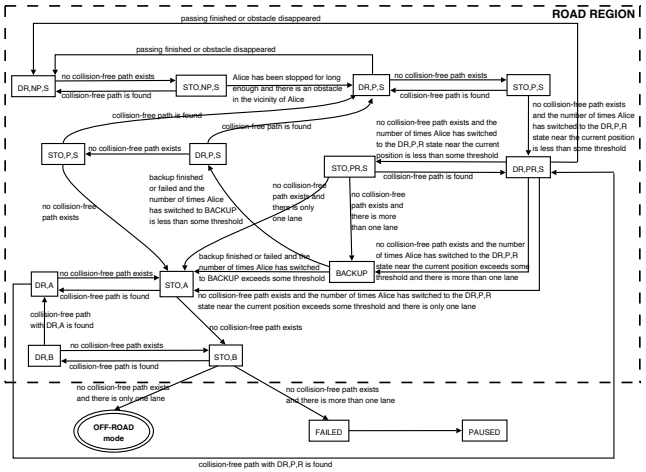


Asynchronous operation

- Notation: $Message_{mod,dir}$ - message to/from a module; Len = length of message queue
- Verify: follower has the right knowledge of the gear that we are currently in, or it commands a full brake.
  - $\Box$ $((Len(TransResp_{f,r}) = Len(Trans_{f,s}))$ $\wedge$ $TransResp_{f,r}[Len(TransResp_{f,r})]$ = COMPLETED $\Rightarrow$ $Trans_f = Trans))$
  - $\Box$ $(Trans_f = Trans \vee Acc_{f,s} = -1)$
- Verify: at infinitely many instants, follower has the right knowledge of the gear that we are currently in, or we have hardware failure.
  - $\Box\Diamond$ $(Trans_f = Trans = Trans_{f,s}[Len(Trans_{f,s})] \vee HW\ failure)$

Moving up the Planning Stack





How do we *design* control protocols that manage behavior

- Mixture of discrete and continuous decision making
- Insure proper response external events, with unknown timing
- Design input = specification + model (system + environment)
- Design output = finite state machine implementing logic

Approach: rapidly explore all trajectories satisfying specs

- Search through all possible actions and events, discarding executions that violate a set of (LTL) specifications
- Issue: state space explosion (especially due to environment)
- Good news: recent results in model checking for class of specs