Principles of Safe Autonomy
ECE 484 SP’26
Lecture 2

Professor: Sayan Mitra

Jan 22, 2026

https://safeautonomy-illinois.github.io/ece484-site/

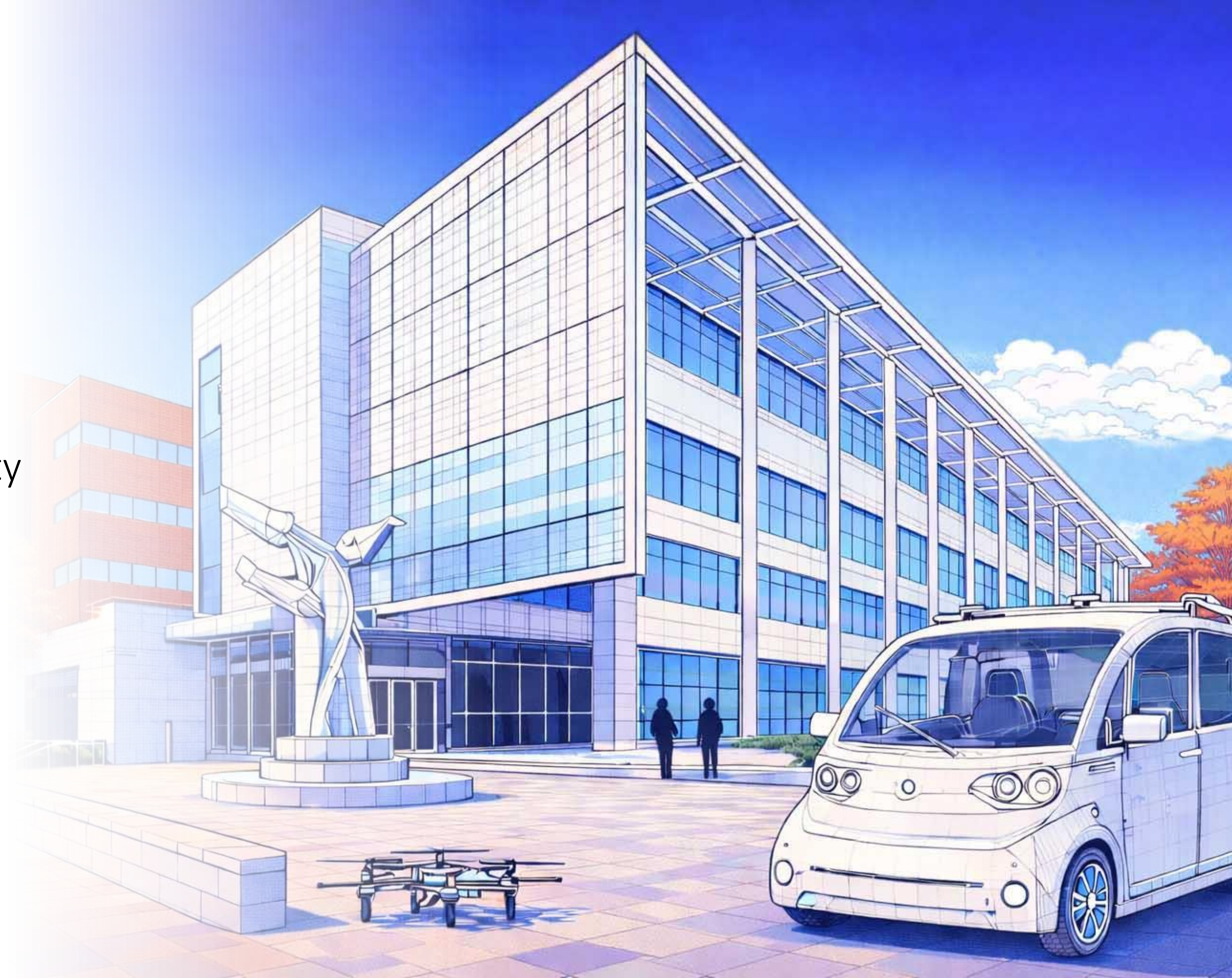
Outline

Motivation

Administrivia

Introduction to Safety

- Models
- Requirements
- Proofs



Automata or state machine models

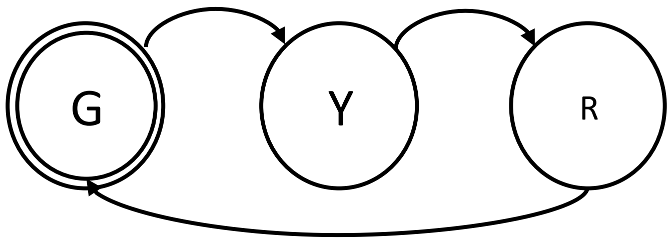
An automaton A is defined by a triple $\langle Q, Q_0, D \rangle$, where

- Q is a set of states
- $Q_0 \subseteq Q$ is a set of initial states
- $D \subseteq Q \times Q$ is a set of transitions

An execution of A is a finite or infinite sequence $q_0, q_1, \ldots$ such that $q_0 \in Q_0$ and $(q_i, q_{i+1}) \in D$

Example: Traffic light automaton

- $Q = \{G, Y, R\}$ $Q_0 = \{G\}$
- $D = \{(G, Y), (Y, R), (R, G)\}$



Execution of traffic light G, Y, R, G, Y, R … infinite even though finite state

Requirements and Counter-examples

Requirements define what the system must and must not do

Example: "Car stays within speed limit"

Autonomous car: "Ego should not collide with lead car"

Collatz: "Every number eventually ends in the 4-2-1 cycle"

A requirement defines a set ***R*** of allowed executions

An execution α that is not in the set ***R*** is a *counter-example*

$R_{\text{eventually-1}} = \{\alpha \mid \exists k\ \alpha_k = 1\}$

An automaton A satisfies a requirement R if *all* executions of A satisfies R

Whether the Collatz automaton satisfies the requirement $R_{\text{eventually-1}}$ for all initial conditions remains an open problem, although no counter-example has been found up to 2^{70}

This is an example of a verification problem

Verification problem

Verification problem: Given an automaton A and a requirement R, check whether all executions of A satisfy R or find a counter-example

Testing or checking individual executions can help find counter-examples but cannot show that there is no counter-example

Verification can be hard because

- |Q| is finite but large and testing may require visiting all the states (e.g., Collatz)
- |Q| is small but the number of executions is very large
- |Q| may be infinite and D may be nondeterministic --- typical for autonomous system

Example: Automatic Emergency Braking (AEB)

Car must brake to maintain safe gap with lead vehicle/pedestrian

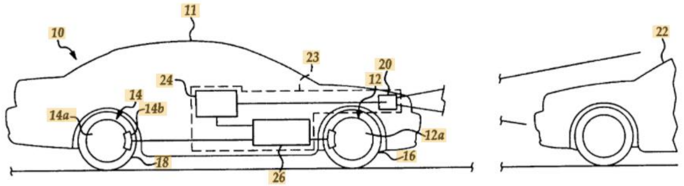
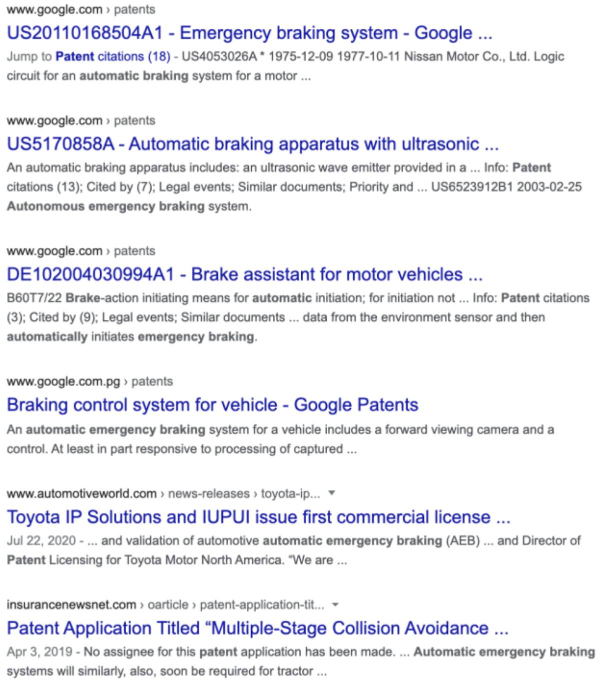


Figure 1


www.google.com › patents
US20110168504A1 - Emergency braking system - Google ...
Jump to **Patent** citations (18) - US4053026A * 1975-12-09 1977-10-11 Nissan Motor Co., Ltd. Logic circuit for an **automatic braking** system for a motor ...

www.google.com › patents
US5170858A - Automatic braking apparatus with ultrasonic ...
An automatic braking apparatus includes: an ultrasonic wave emitter provided in a ... Info: **Patent** citations (13); Cited by (7); Legal events; Similar documents; Priority and ... US6523912B1 2003-02-25 **Autonomous emergency braking** system.

www.google.com › patents
DE102004030994A1 - Brake assistant for motor vehicles ...
B60T7/22 **Brake**-action initiating means for **automatic** initiation; for initiation not ... Info: **Patent** citations (3); Cited by (9); Legal events; Similar documents ... data from the environment sensor and then **automatically** initiates **emergency braking**.

www.google.com.pg › patents
Braking control system for vehicle - Google Patents
An **automatic emergency braking** system for a vehicle includes a forward viewing camera and a control. At least in part responsive to processing of captured ...

www.automotiveworld.com › news-releases › toyota-ip...
Toyota IP Solutions and IUPUI issue first commercial license ...
Jul 22, 2020 - ... and validation of automotive **automatic emergency braking** (AEB) ... and Director of **Patent** Licensing for Toyota Motor North America. "We are ...

insurancenewsnet.com › oarticle › patent-application-tit...
Patent Application Titled "Multiple-Stage Collision Avoidance ...
Apr 3, 2019 - No assignee for this **patent** application has been made. ... **Automatic emergency braking** systems will similarly, also, soon be required for tractor ...

There is no standard for checking correctness of AEB systems

Future: Every code commit in github from an AEB engineer, **proves a theorem** establishing A satisfies R_{gap}

Automaton model of AEB

Automaton $A = \langle Q, Q_0, D \rangle$

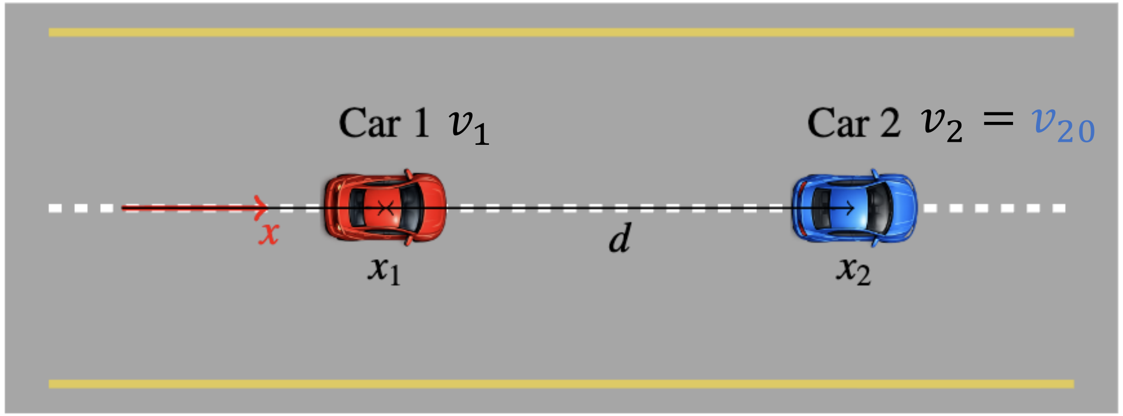
- $Q: [x_1, x_2, v_1] \in \mathbb{R}^3$
- $Q_0 = \{[x_1 = x_{10}, x_2 = x_{20}, v_1 = v_{10}]\}$
- $D \subseteq Q \times Q$ written as a program

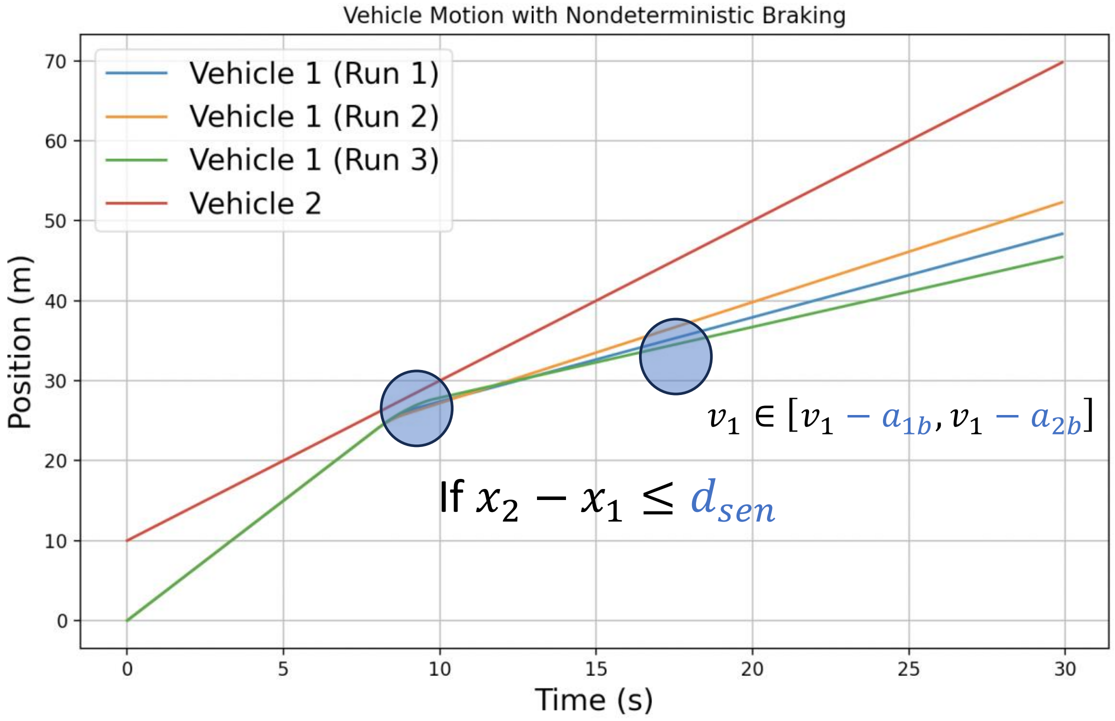
If $x_2 - x_1 \leq d_{sen}$

$v_1 \in [v_1 - a_{1b}, v_1 - a_{2b}]$

$x_2 = x_2 + v_2$

$x_1 = x_1 + v_1$





Automaton model of AEB

Automaton $A = \langle Q, Q_0, D \rangle$

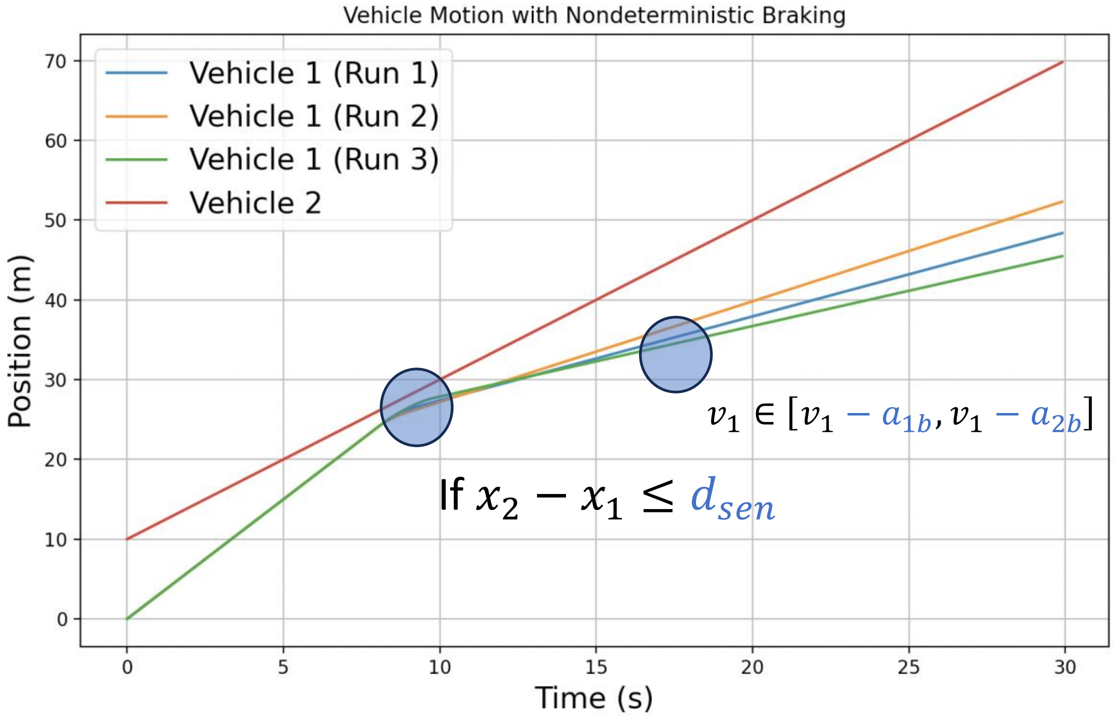
- $Q: \mathbb{R}^3$; $\boldsymbol{q} \in Q$ $\boldsymbol{q}.x_1, \boldsymbol{q}.x_2 \in \mathbb{R}$
- $Q_0 = \{\boldsymbol{q} \mid [\boldsymbol{q}.x_1 = x_{10}, \boldsymbol{q}.x_2 = x_{20}, \boldsymbol{q}.v_1 = v_{10}]\}$
- $(\boldsymbol{q}, \boldsymbol{q}') \in D$ iff

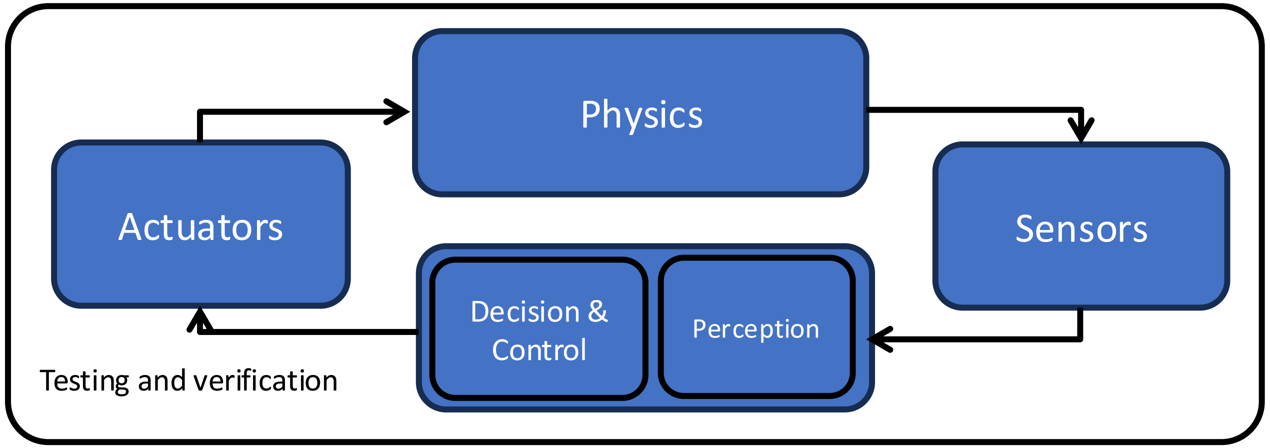
  If $\boldsymbol{q}.x_2 - \boldsymbol{q}.x_1 \leq d_{sen}$

  $\boldsymbol{q}'.v_1 \in [\boldsymbol{q}.v_1 - a_{1b}, \boldsymbol{q}.v_1 - a_{2b}]$

  $\boldsymbol{q}'.x_2 = \boldsymbol{q}.x_2 + \boldsymbol{q}.v_2$

  $\boldsymbol{q}'.x_1 = \boldsymbol{q}.x_1 + \boldsymbol{q}.v_1$





What is missing in the AEB model?

If $x_2 - x_1 \leq 2.0$

$v_1 \in [v_1 - a_{1b}, v_1 - a_{2b}]$

else $v_1 = v_1$

$x_2 = x_2 + v_2$

$x_1 = x_1 + v_1$

- Acceleration, friction in dynamics
- Uncertainty in sensing
- Uncertainty in lead vehicle behavior
- Timing of execution of control loop

“All models are wrong, but some are useful.”

Safety and liveness requirements

$R_{gap} = \{\alpha \mid \forall i\ \alpha_i.x_2 > \alpha_i.x_1\}$ non-zero gap $U_{gap} = \{\boldsymbol{q} | \boldsymbol{q}.x_2 - \boldsymbol{q}.x_1 \leq 0\}$

$R_{sp-lim} = \{\alpha \mid \forall i\ \alpha_i.v_1 \leq 70\}$ speed limit $U_{sp-lim} = \{\boldsymbol{q} | \boldsymbol{q}.x_1 \geq 70\}$

$R_{catch-up} = \{\alpha \mid \exists i\ 2 > \alpha_i.x_2 - \alpha_i.x_1 > 1\}$ catch eventually

A safety requirement says that *every* state along *every* execution should stay in safe states

Equivalently, no execution of A ever reaches any unsafe states

R_{gap} and R_{sp-lim} are safety requirements with U_{gap} and U_{sp-lim} as the unsafe sets

$R_{catch-up}$ is not a safety requirement; it is an example of a liveness / progress requirement

A liveness requirement says that along every execution eventually some good state is reached

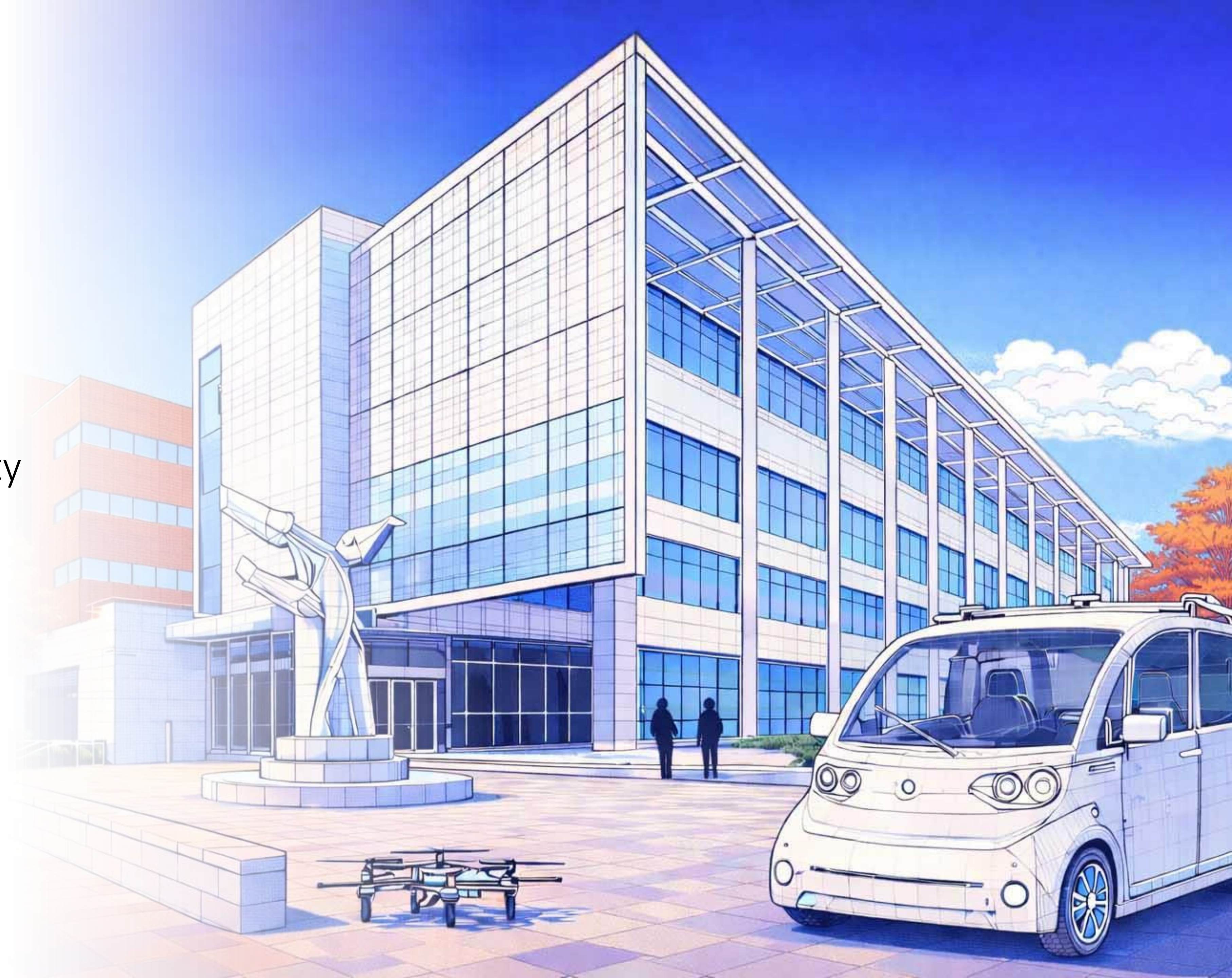
Outline

Motivation

Administrivia

Introduction to Safety

- Models
- Requirements
- Verification



Safety verification: Finite State Automata

Safety verification problem: Given an automaton A and an unsafe set U, check whether there exists any execution α of A that reaches U

Counter-examples of safety are finite executions ending in U

For finite automata, safety verification can be solved using depth first search from Q_0

- Consider $\langle Q, Q_0, D \rangle$ as a directed graph with $\langle Q, D \rangle$
- DFS computes all paths or executions from Q_0
- If none of these executions hit U there is no counter-example
- Absence of a counter-example **proves** that the automaton is safe

In practice, explicit enumeration of all paths may not scale to large graphs

Safety verification and Reachability: Infinite State Spaces

A state $q \in Q$ is reachable if there exists an execution α such that $\alpha_i = q$.

$Reach_A(Q_0) \subseteq Q$ the set of reachable states of A from Q_0

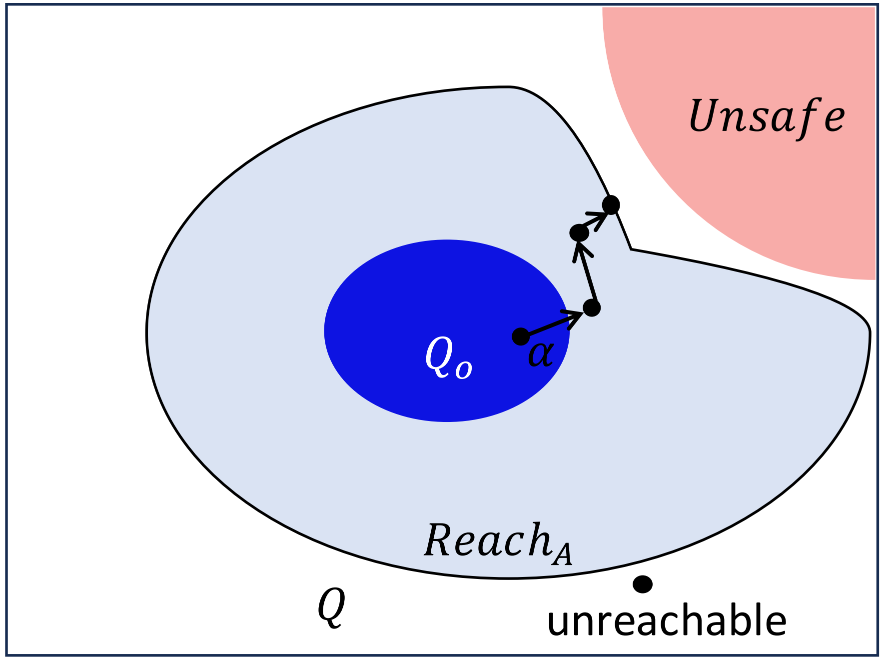
Safety verification problem is equivalent to as checking $Reach_A \cap U = \emptyset$?

That is, if we can compute $Reach_A$ then we can verify safety

Finite state systems DFS computes $Reach_A(Q_0)$

For infinite state systems, we need:

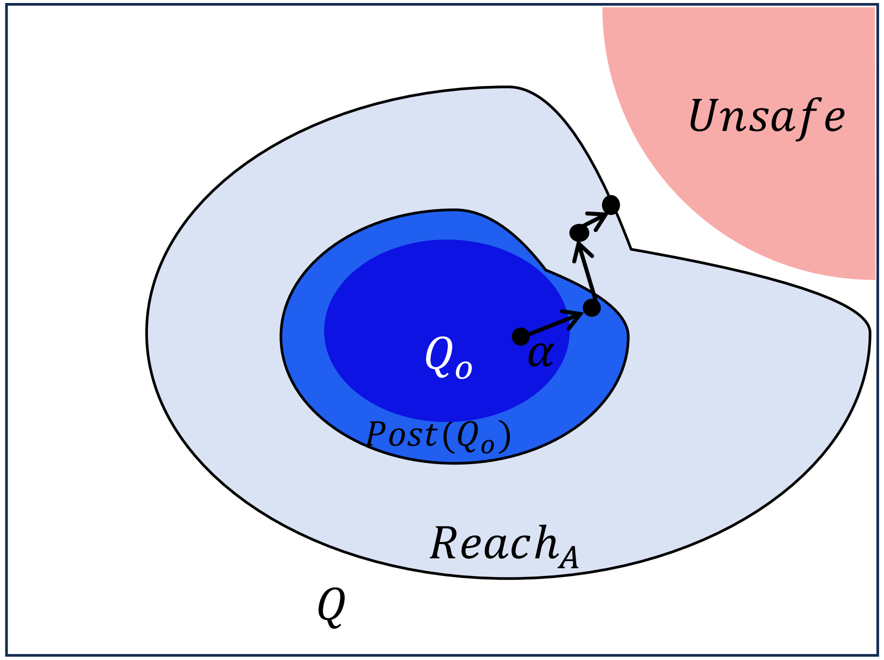
- Representation of infinite sets of states
- Iteratively computing $Reach_A$



Computing reachable sets and over-approximations

Define $Post(R) = \{q' \mid \exists\, q \in R, (q, q') \in D\}$ that gives all the states that can be reached in one step from the set of states R

- For a deterministic system $Post(\{q\}) = q'$ for $(q, q') \in D$
- For finite R, $Post(R) = \cup_{q \in R} Post(\{q\})$
- $Post(Q_0) = \{q \mid \exists\, q_0 \in Q_0, (q_0, q) \in D\}$ states reachable in 1 step from Q_0
- $Post(Post(Q_0)) = ?$



Infinite sets & nondeterministic $Post(R)$ requires some representation of sets

Example:

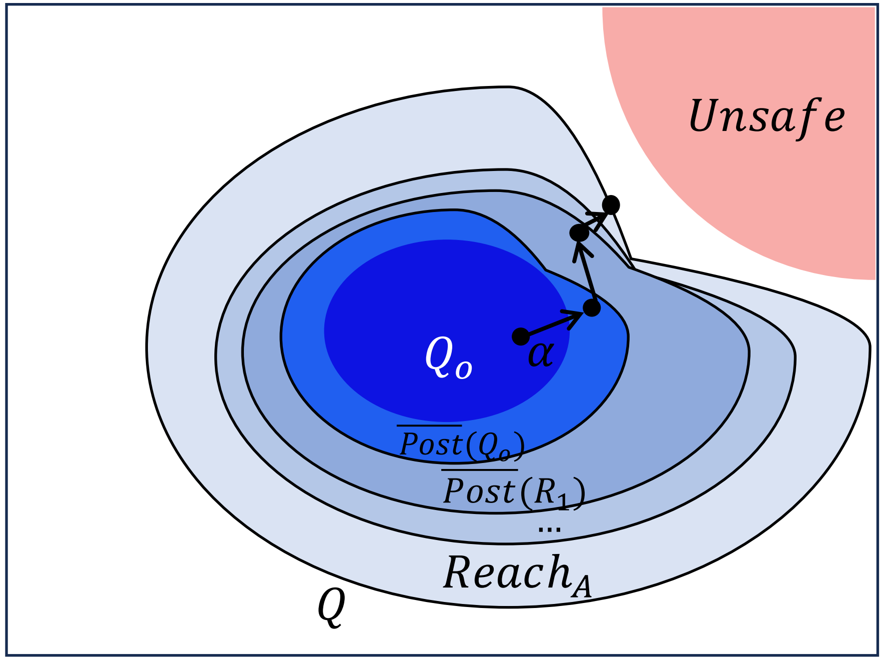
- $Q = [x_1 : \mathbb{R}]$ $D: x_1 = x_1 + v_1$ then $R = [a, b]$ $Post(R) = [a + v_1, b + v_1]$
- $Q = \mathbb{R}^4$ then $R = [\boldsymbol{a}, \boldsymbol{b}]$ then $Post(R)$ is a hyperrectangle

Generally, for nonconvex R nonlinear D exact $Post(R)$ may be infeasible

We use over-approximation $\overline{Post}(R)$ such that $Post(R) \subseteq \overline{Post}(R)$

Reachable sets and over-approximations

Reachability($A = \langle Q, Q_0, D \rangle$)
$R_0 = Q_0$
$R_1 = \emptyset$
$i = 0$
do
  $R_{i+1} = \overline{Post}\,(R_i) \cup R_i$
  $i = i + 1$
Until $R_i \neq R_{i-1}$
Return R_i



Exercise. Show that Post and $\overline{Post}$ is monotonic, i.e., If $\boldsymbol{S_1} \subseteq \boldsymbol{S_2}$ then $Post(S_1) \subseteq Post(S_2)$.

Exercise. Show that all states that are reachable in exactly k steps is $Post^k(Q_0)$.

Exercise. If this algorithm terminates and returns R then $Reach_A(Q_0) \subseteq R$, i.e., it computes an over-approximation of the reachable sets of A.

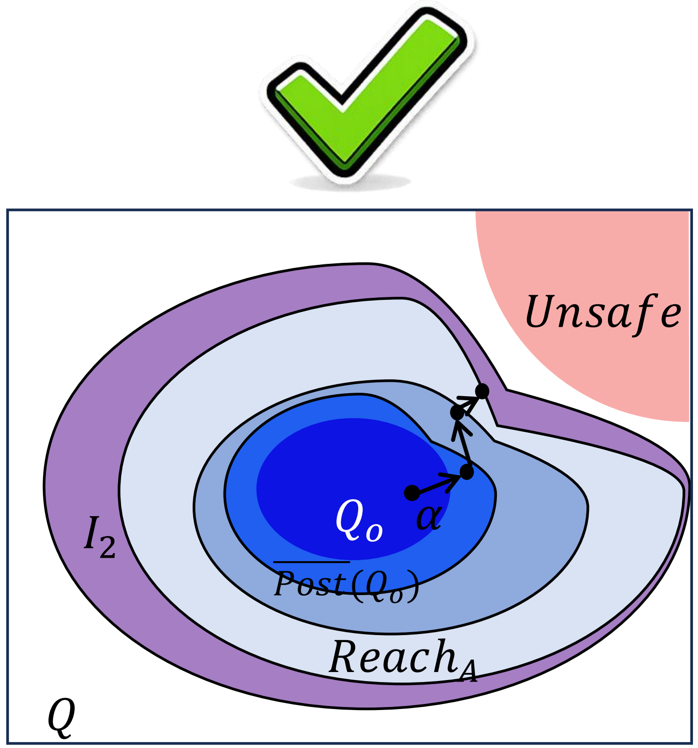
$R \cap Unsafe = \emptyset$ proves safety, but $\overline{Reach_A}(Q_0) \cap Unsafe \neq \emptyset$ does not imply that there is a real counterexample

Invariants and safety verification

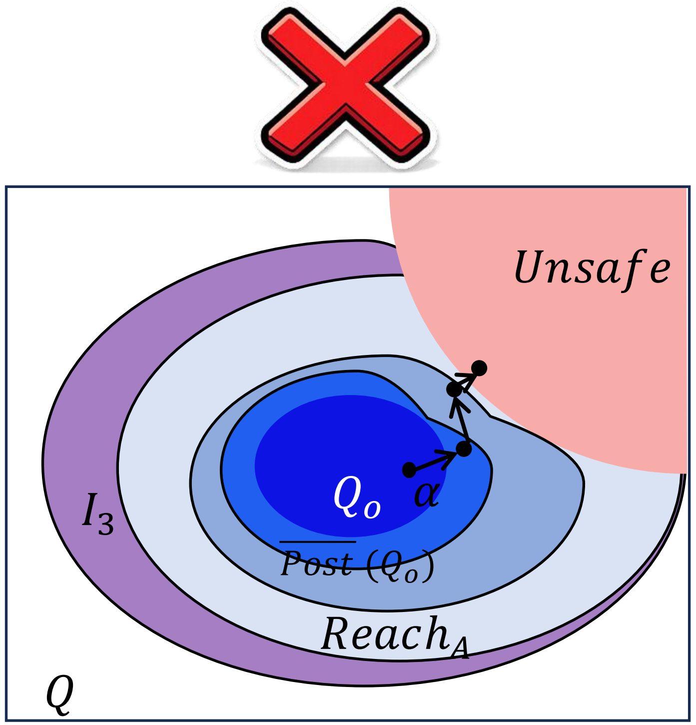
A set $I \subseteq Q$ is an invariant if $Reach_A(Q_0) \subseteq I$

Over-approximates the reachable states, not unique, and define everything that *can* happen

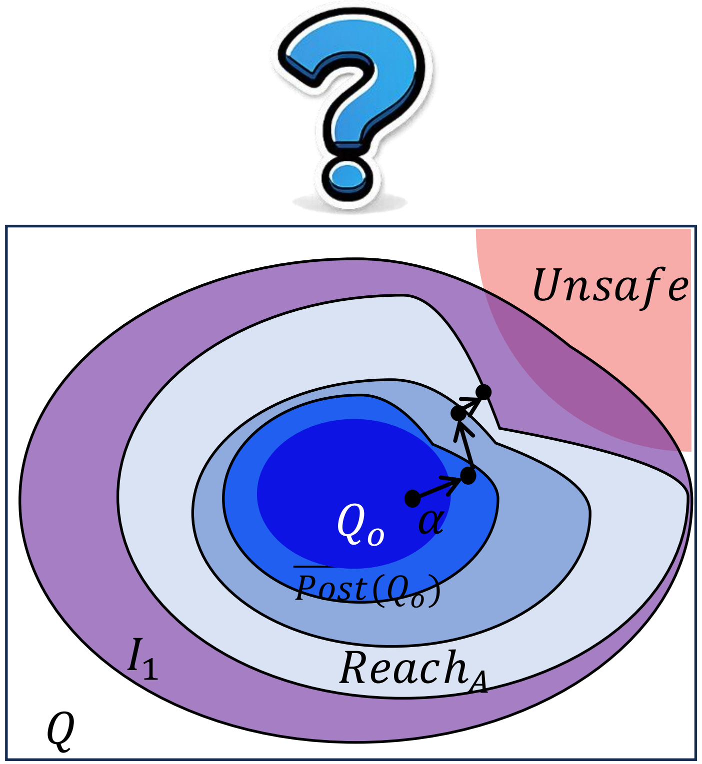
If the algorithm terminates, it returns *an* invariant which may or may not prove safety



System is safe but and verified by invariant I_2



$I_3 \cap Unsafe \neq \emptyset$ and system is unsafe

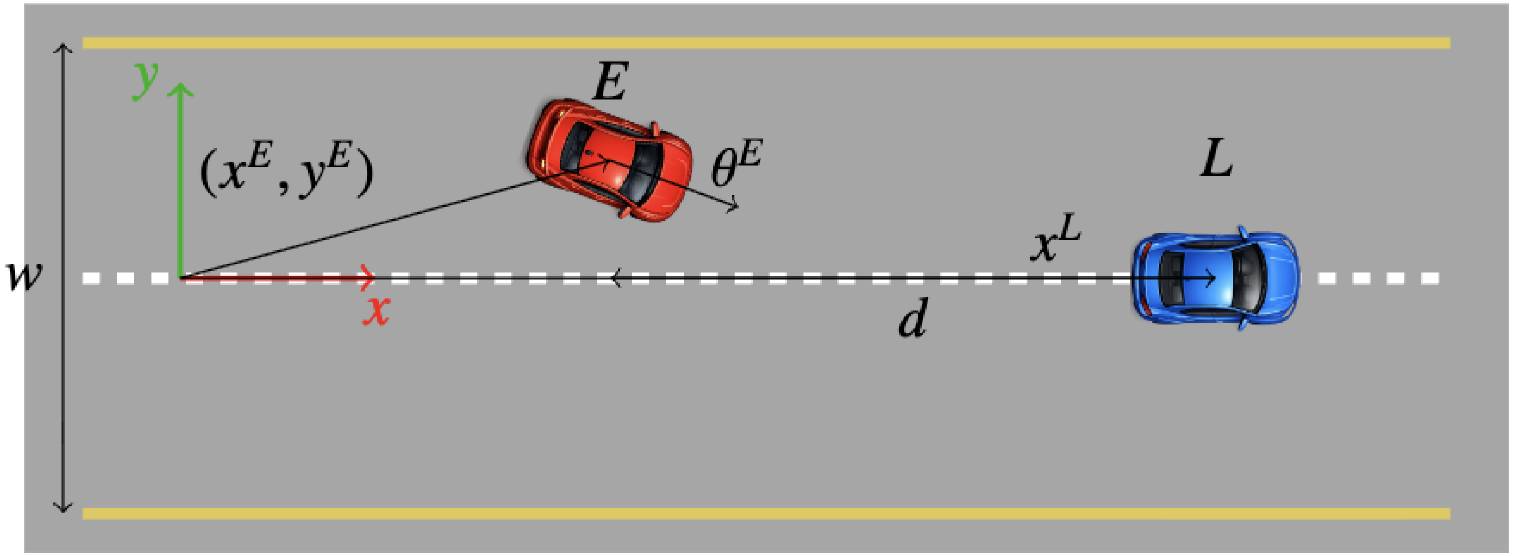


$I_1 \cap Unsafe \neq \emptyset$ but system is safe

Example: Lane-keeping

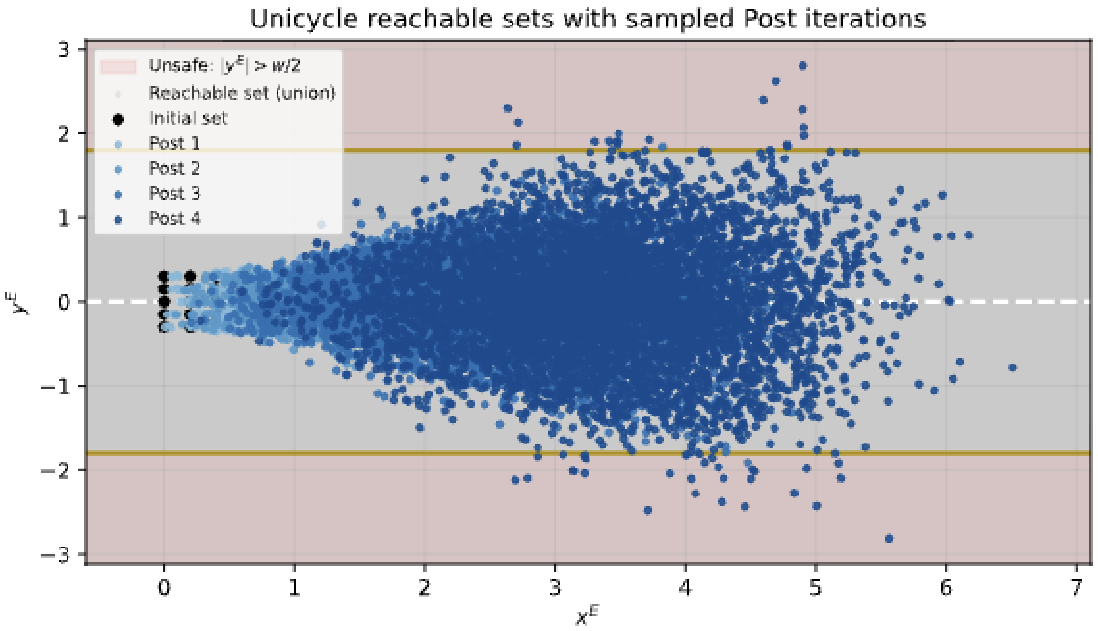
Example. Vehicle (E) with braking and lane-keeping controller $Q: [x^E, y^E, \theta^E, x^L] \in \mathbb{R}^4$ and velocities are chosen in each step

See course reader for definition of D

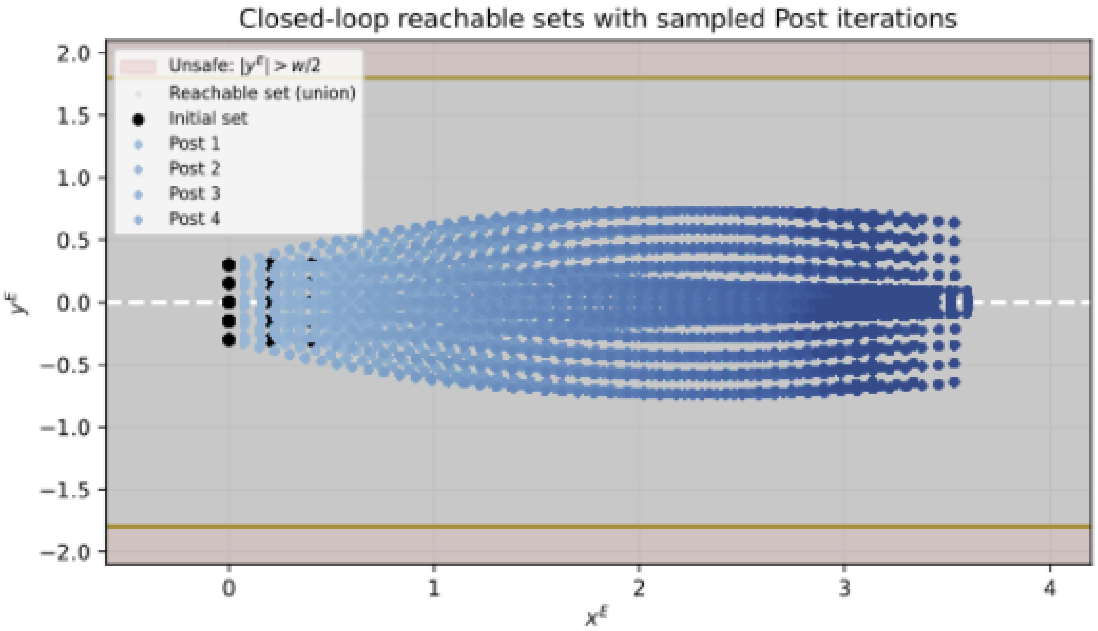


Reachable sets

Open-loop



Closed-loop

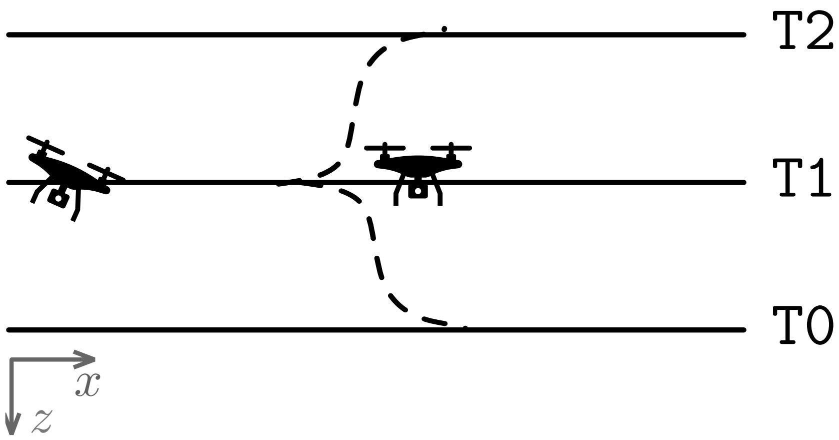


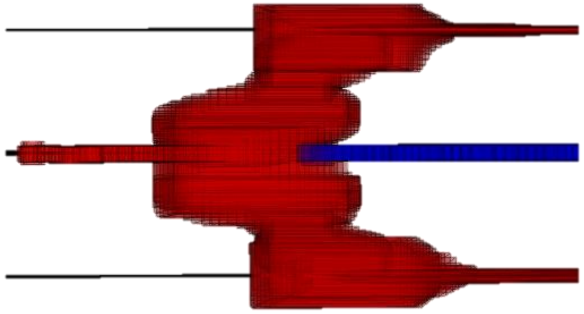
Summary

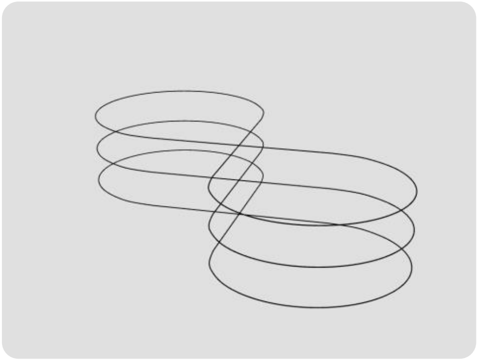
- Canvas quiz: https://canvas.illinois.edu/courses/67113/assignments/1563205?display=full_width_with_nav
- Verification is the problem of proving/disproving requirements
- Safety requirements state Unsafe things never happen OR
  - All reachable states are disjoint from unsafe sets $\mathbf{Reach}_A \cap Unsafe = \emptyset$
- For finite state systems explicit reachability possible via DFS
- In general, reachability and verification are hard (state space explosion, undecidability)
- We can over-approximate $\mathbf{Reach}_A \subseteq \overline{Post}^k(Q_0)$

Verse: Python library for reachability analysis (MP0)

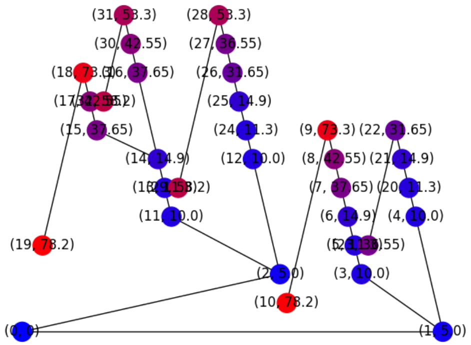
```
class Mode(Enum):
  Normal = auto()
  Up = auto()
  …
class Track(Enum):
  T0 = auto()
  T1 = auto()
  …
class State:
  x: float
  y: float
  …
  mode: Mode
  track: Track

def decisionLogic(ego: State, others: List[State], map):
  if ego.mode == Normal:
    if any(isClose(ego, other) for other in others):
      if map.exist(ego.track, ego.mode, Up):
        next.mode = Up
        next.track = map.h(ego.track, ego.mode, Up)
      if map.exist(ego.track, ego.mode, Down):
        next.mode = Down
        …

assert not any(isVeryClose(ego, other) for other in others), "Seperation"
```





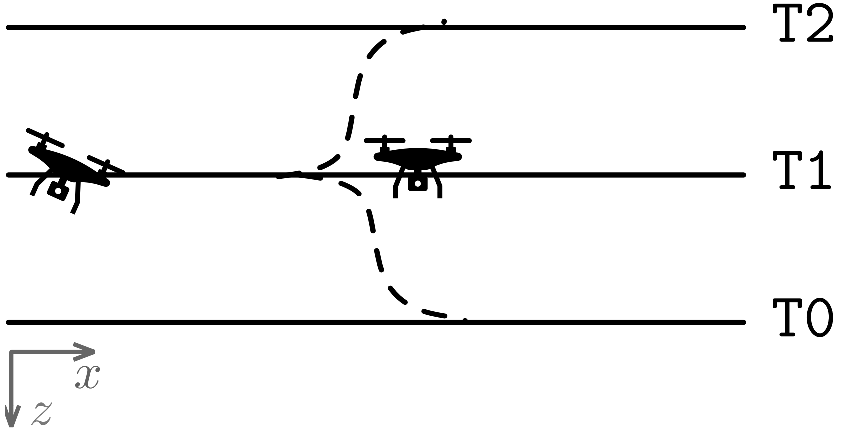


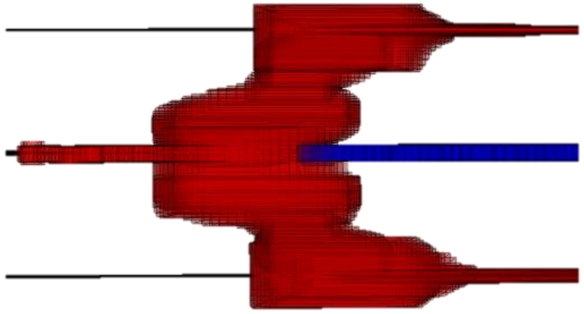
```
q1 = QuadrotorAgent(”q1", …) // Defines the dynamics
q1.set_initial([…], (Mode.Normal, Track.T1))
scenario.add_agent(q1)
q2 = …
scenario.set_map(M5())
scenario.simulate(…)
scenario.verify(…)
```

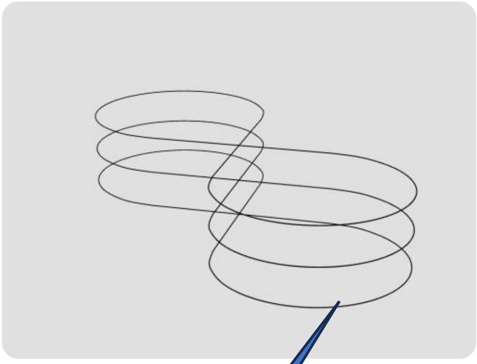


Verse: Python library for reachability analysis (MP0)

```
class Mode(Enum):
  Normal = auto()
  Up = auto()
  …
class Track(Enum):
  T0 = auto()
  T1 = auto()
  …
class State:
  x: float
  y: float
  …
  mode: Mode
  track: Track
```





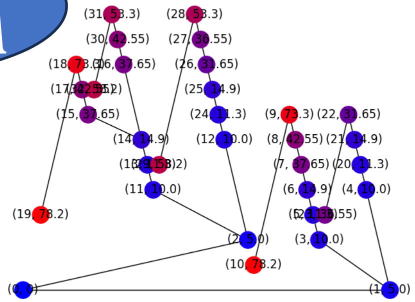


Q

D

```
q1 = QuadrotorAgent("q1", …)
q1.set_initial([…], (Mode.Normal, Track.T1))
scenario.add_agent(q1)
q2 = …
scenario.set_map(M5())
scenario.simulate(…)
scenario.verify(…)
```

$Reach_A$



```
def decisionLogic(ego: State, others: List[State], map):
  if ego.mode == Normal:
    if any(isClose(ego, other) for other in others):
      if map.exist(ego.track, ego.mode, Up):
        next.mode = Up
        next.track = map.h(ego.track, ego.mode, Up)
      if map.exist(ego.track, ego.mode, Down):
        next.mode = Down
        …

assert not any(isVeryClose(ego, other) for other in others), "Seperation"
```

$Unsafe$

Inductive invariants

Proposition 1. If (i) $\mathrm{Q}_0 \subseteq I$ and (ii) $Post(I) \subseteq I$ then I is an invariant, i.e., $\mathrm{Reach}_{\mathrm{A}} \subseteq I$.

Such invariants are called inductive invariants

Proof. Consider any reachable state $\boldsymbol{q} \in Reach_A \subseteq Q$

By definition of reachable state, there is an execution α with $\alpha_k = \boldsymbol{q}$

By induction on k we will show that $\boldsymbol{q} \in I$

Base case, for k=0, $\alpha_0 = q_0 \in Q_0 \subseteq I$ [using definition of execution and (i)]

Induction. By inductive hypothesis, suppose $\alpha_k \in I$. We have to show $\boldsymbol{q} = \alpha_{k+1} \in I$.

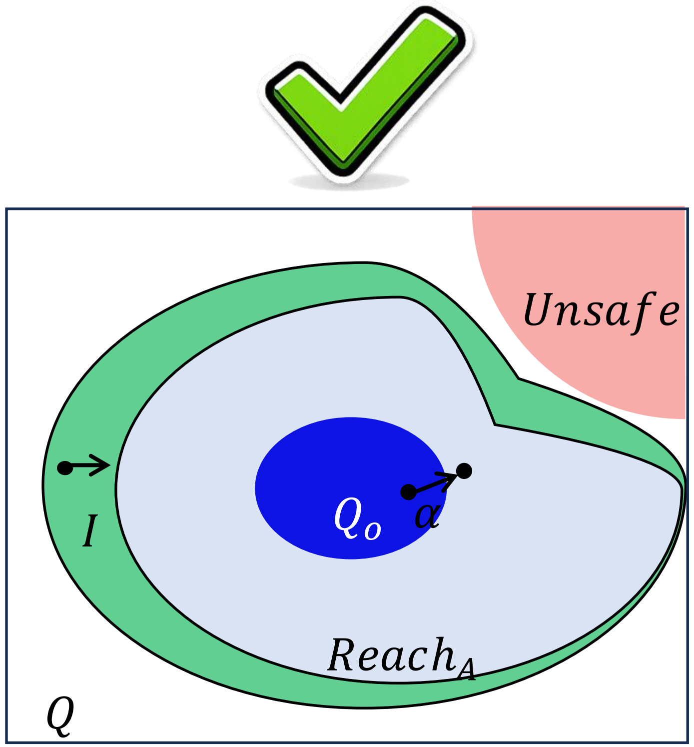
$\boldsymbol{q} \in Post(\alpha_k)$ [Definition of Post, $(\alpha_k, \boldsymbol{q}) \in D$]

$\boldsymbol{q} \in Post(I)$ [Monotonicity of Post. $\alpha_k \in I \Rightarrow Post(\alpha_k) \subseteq Post(I)$]

$\boldsymbol{q} \subseteq I$ [By (ii)]

Inductive invariants and Safety

- Guess a candidate inductive invariant I
- If $\mathrm{I} \cap Unsafe = \emptyset$ and $\mathrm{Q}_0 \subseteq I$ and $Post(I) \subseteq I$ then by the Proposition 1 $\mathbf{Reach}_A \subseteq I$ and we have verified safety
- If the start and transition conditions fail, that does *not* imply that I is not an invariant
- It only implies that I cannot be checked inductively by Proposition 1.



System is safe and verified by the inductive invariant

Revisiting AEB

To prove no crash $x_2 > x_1$ in all reachable states, we will need assumptions about initial conditions $(x_{10}, x_{20}, v_{10}, v_{20})$, sensing distance (d_s), and braking acceleration (a_b)

Discovering these assumptions (for system correctness) is a valuable side-effect of verification

Assumption: $x_{20} - x_{10} > d_s > \frac{v_{10}^2}{ab}$

The proof of correctness (as expected) will relate total time of braking with the initial separation. We need a timer

Checking Inductive Invariant for AEB

timer = 0
If $x_2 - x_1 \le d_s$
  If $v_1 \ge a_b$
    $v_1 = v_1 - a_b$
    timer := timer+1
  else
    $v_1 = 0$
else
  $v_1 = v_1$
$x_2 = x_2 + v_2$
$x_1 = x_1 + v_1$

Invariant. I_1: $\text{timer} + \frac{v_1}{a_b} \le \frac{v_{10}}{a_b}$.

Bound on total braking time in terms of velocity and deceleration

Proof. We need to check two conditions for this to be an inductive invariant: (i) $Q_0 \in I_1$ and (ii) $Post(I_1) \subseteq I_1$.

(i) Consider any $q \in Q_0$. We need to show $q \in I_1$.

$$q.timer + \frac{q.v_1}{a_b} = 0 + \frac{v_{10}}{a_b} \le \frac{v_{10}}{a_b}.$$

(ii) Consider any $(q, q') \in D$ with $q \in I_1$. We need to show $q' \in I_1$.

As there are three branches in D, there are 3 cases.

$$(a)\ q'.timer + \frac{q'.v_1}{a_b} = q.timer + 1 + \frac{q.v_1 - a_b}{a_b} = q.timer + \frac{q.v_1}{a_b} \le \frac{v_{10}}{a_b}$$

$$(b)\ q'.timer + \frac{q'.v_1}{a_b} = q.timer + 0 \le \frac{v_{10}}{a_b}$$

$$(c)\ q'.timer + \frac{q'.v_1}{a_b} = q.timer + \frac{q.v_1}{a_b} \le \frac{v_{10}}{a_b}$$

I_2: $\text{timer} \le \frac{v_{10}}{a_b}$

Invariants and assumptions give correctness proof

Consider any two reachable states:

q_1 is where $x_2 - x_1 \leq d_s$ became true first, and

q_2 is reached from q_1 with $\mathrm{q}_2.x_2 - q_2.x_1 \leq d_s$ (other reachable states are safe)

$q_2.x_2 - q_2.x_1$

$> q_1.x_2 - q_2.x_1$ [1, Because x_2 increased]

$> q_1.x_2 - q_1.x_1 - v_{10}.\frac{v_{10}}{a_b}$ [$\mathrm{I}_2 \Rightarrow \mathbf{timer} \leq \frac{v_{10}}{a_b}$ and $q_2.x_1 \leq q_1.x_1 + v_{10}.\frac{v_{10}}{a_b}$]

$> d_s - \frac{v_{10}^2}{a_b}$ [By def of q_1]

> 0 [By Assumption]

Summary

- Testing alone is inadequate---in theory and practice
- Automaton (state machine) models, executions, and requirements give us the language to state correctness claims precisely
- Verification is the problem of proving/disproving such claims
- Safety claims are a (prevalent) subset of correctness claims
- Reachability analysis can prove/disprove safety
- In general, reachability and verification are hard (state space explosion, undecidability)
- Inductive invariants over-approximating reachable states give a practical method for proving safety