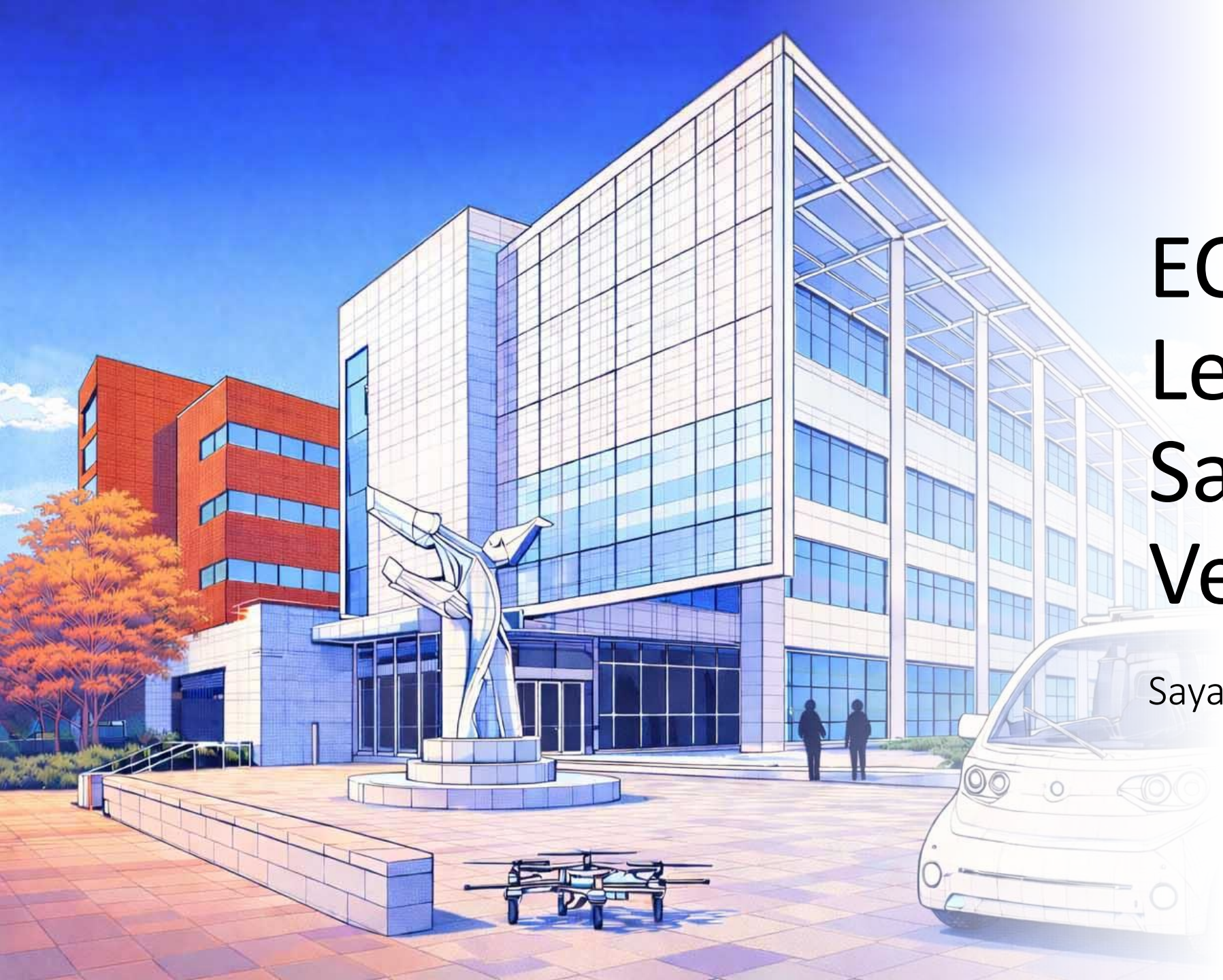


# ECE 484 Lecture 3 Safety Verification

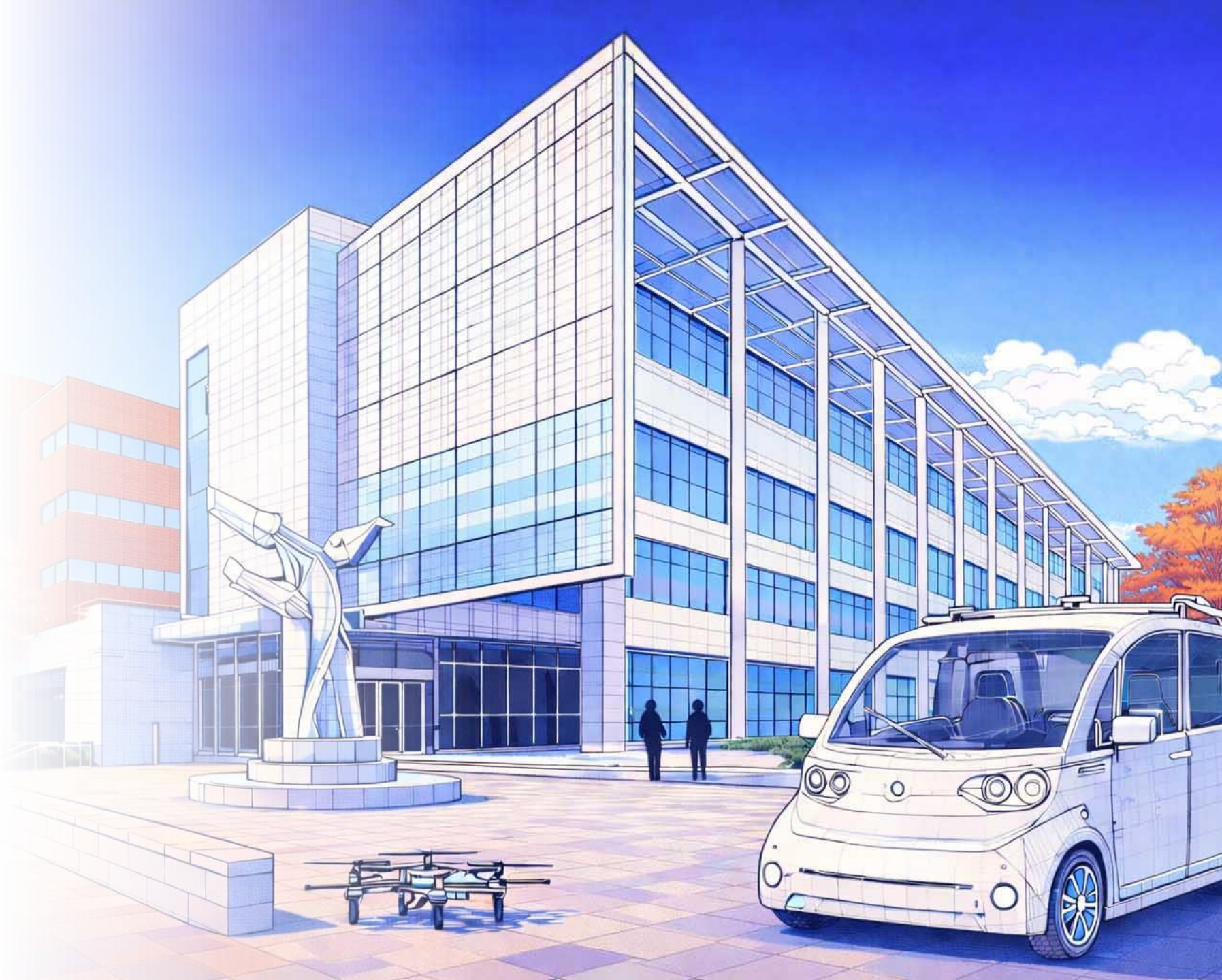
Sayan Mitra

# Outline

Safety Verification

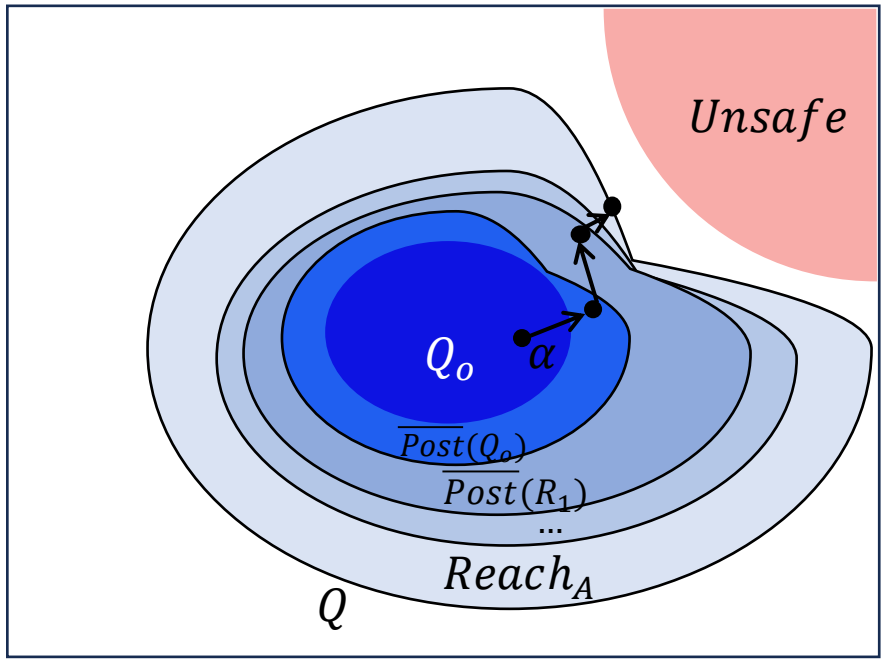
Inductive invariants

Coordinates



# Safety Verification via Reachability

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 $R_i$




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.

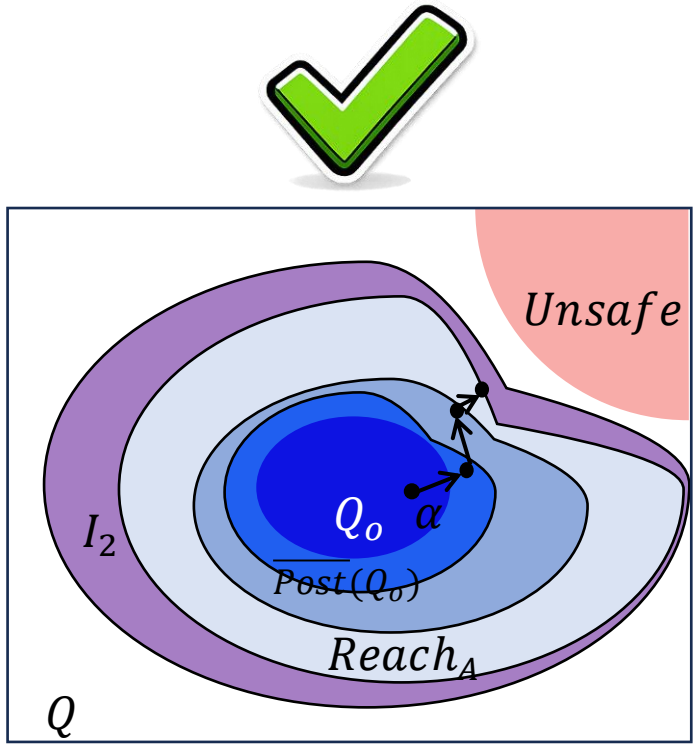
$R \cap Unsafe = \emptyset$ proves safety, but $\overline{Reach_A}(Q_0) \cap Unsafe \neq \emptyset$ does not imply that there is a real counterexample

# Invariants and safety verification

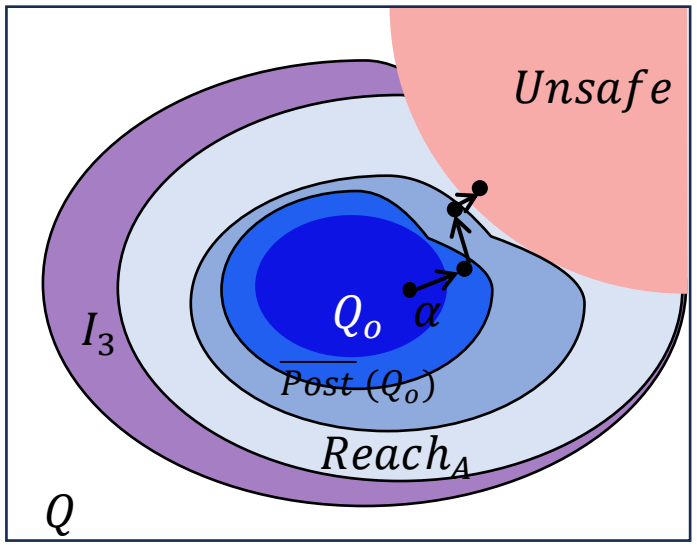
A set $I \subseteq Q$ is an invariant if $Reach_A(Q_0) \subseteq I$

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

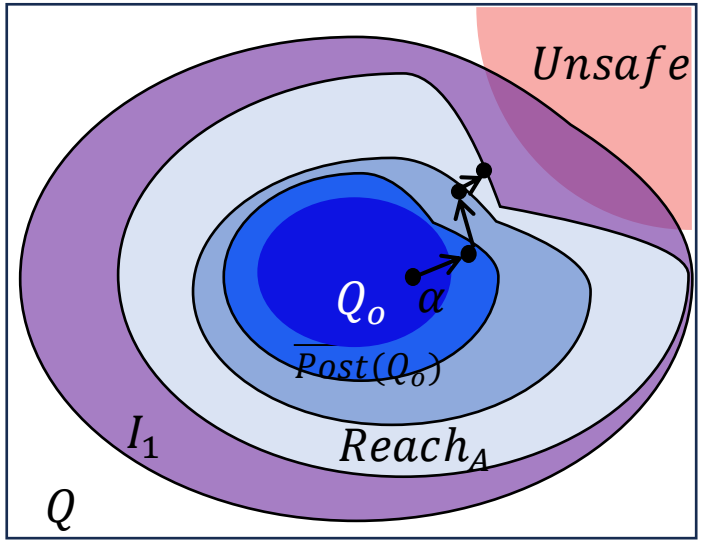
If the algorithm terminates, it returns *an* invariant which may or may not prove safety




System is safe but and verified by invariant $I_2$



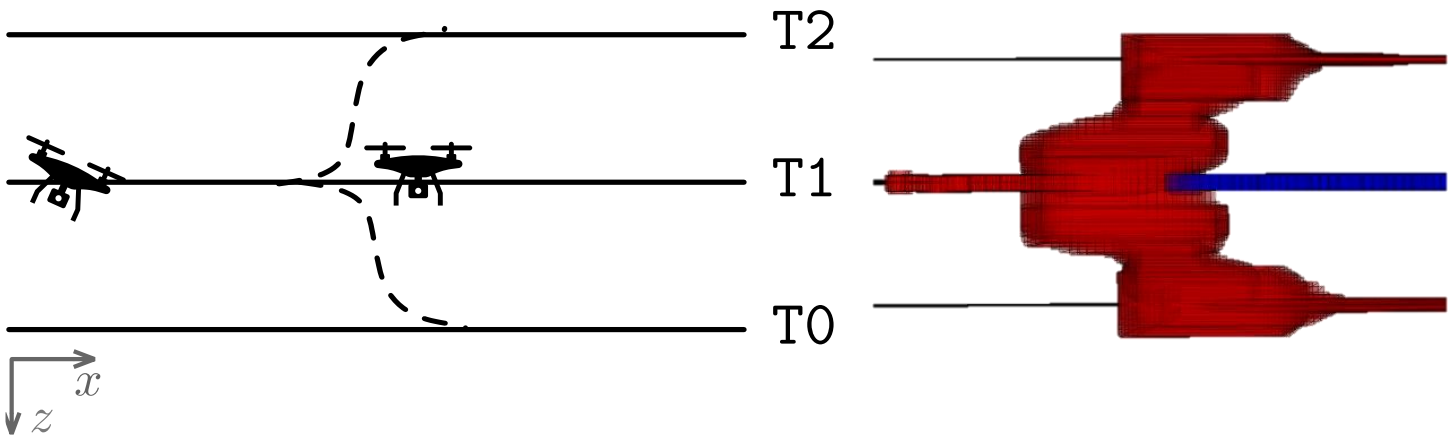

$I_3 \cap Unsafe \neq \emptyset$ and system is unsafe

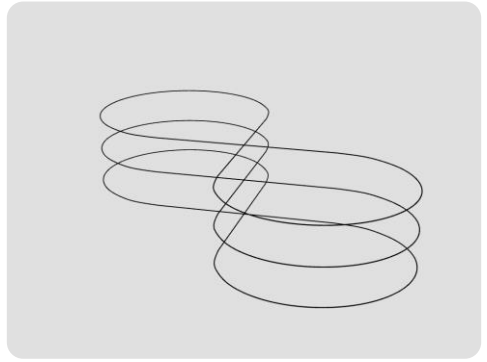



$I_1 \cap Unsafe \neq \emptyset$ but system is safe

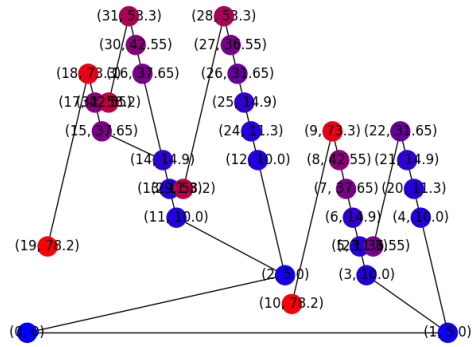
# Verse performs reachability analysis (MP0)

```
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"
```



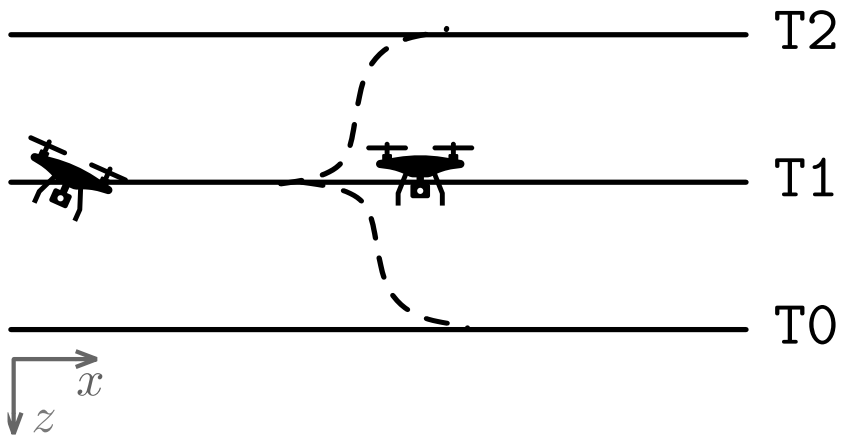



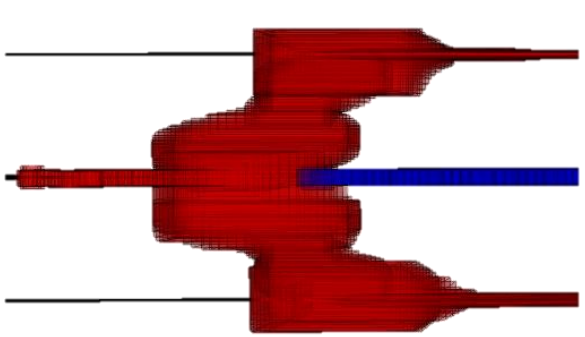
```
q1 = QuadrotorAgent(”q1", …) // Defines the dynamics
q1.set_initial([…], (Mode.Normal, Track.T1))
scenario.add_agent(q1)
q2 = …
scenario.set_map(M5())
scenario.simulate(…)
scenario.verify(…)
```

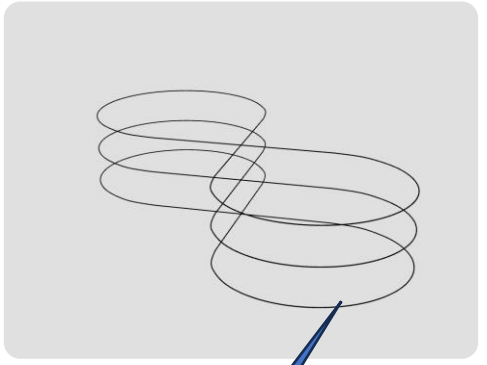


# Verse: Python library for reachability analysis (MP0)

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





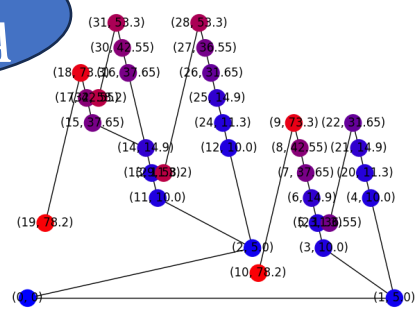
$Q$

$D$

```
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
        …
```

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

$Reach_A$



$Unsafe$

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

# Reachability Analysis

## Benefits

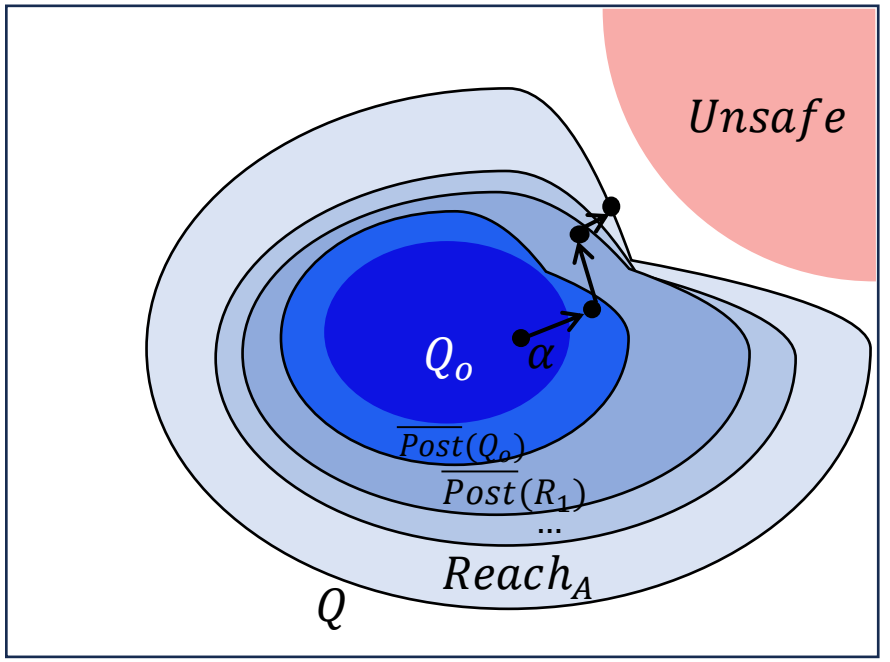
Fully automatic

## Limitations

Termination

Scalability

Conservativeness




Alternative approach: Guess and check an invariant that is adequate for safety

# Inductive invariants

**Proposition 1.** If (i) $\mathrm{Q_0} \subseteq I$ and (ii) $Post(I) \subseteq I$ then $I$ is an invariant, i.e., $\mathrm{Reach_A} \subseteq I$.

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 $\alpha$ 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 $\boldsymbol{q} = \alpha_{k+1} \in I$.

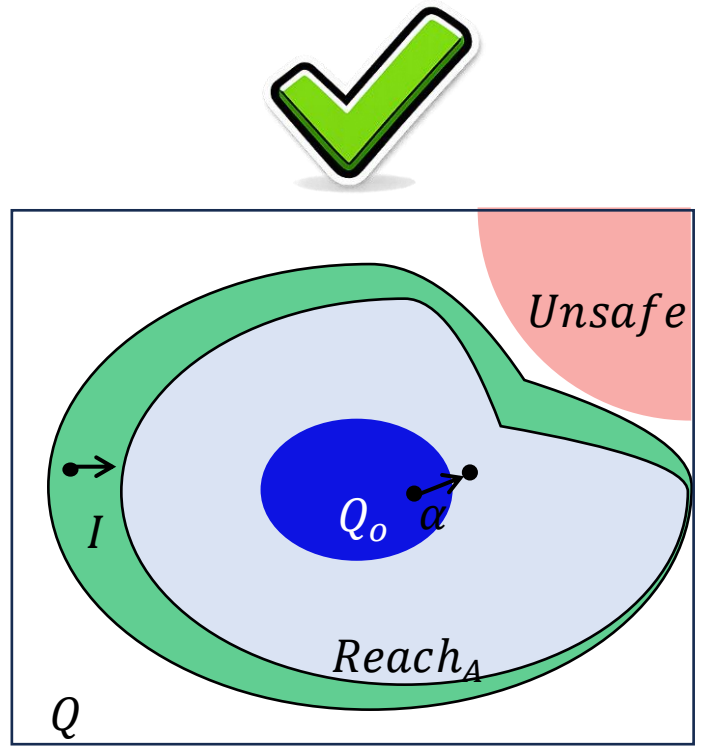
$\boldsymbol{q} \in Post(\alpha_k)$ [Definition of Post, $(\alpha_k, \boldsymbol{q}) \in D$]

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

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

# Inductive invariants and Safety

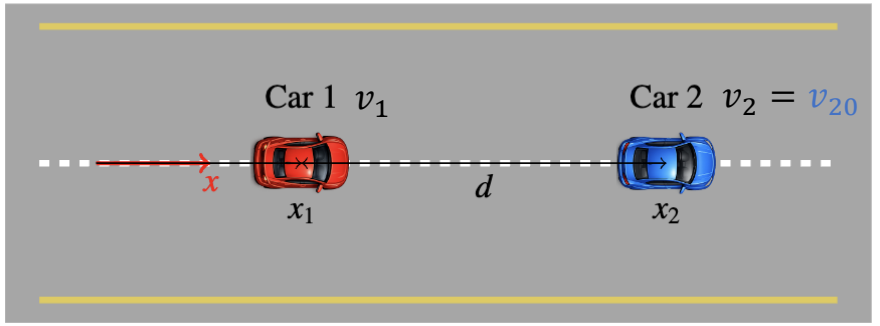
- Guess a candidate inductive invariant $I$
- If $\mathrm{I} \cap Unsafe = \emptyset$ and $\mathrm{Q_0} \subseteq I$ and $Post(I) \subseteq I$ then by the Proposition 1 $\mathbf{Reach}_A \subseteq I$ and we have verified safety
- If the start and transition conditions fail, that does *not* imply that $I$ is not an invariant
- It only implies that $I$ cannot be checked inductively by Proposition 1.

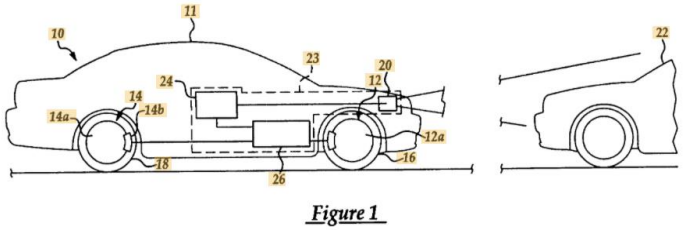



System is safe and verified by the inductive invariant

# Automatic Emergency Braking (AEB)

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

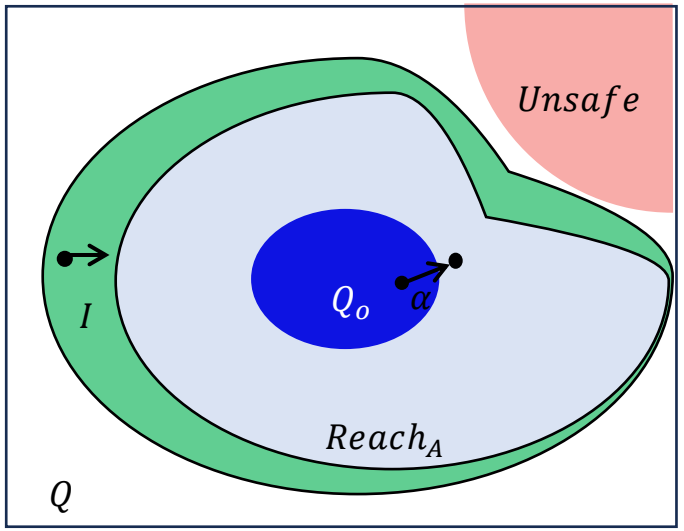



Safety requirement $x_2 > x_1$

$Q: [x_1, x_2, v_1] \in \mathbb{R}^3$

$Unsafe \subseteq \mathbb{R}^3 := \{\boldsymbol{q} \mid \boldsymbol{q}.x_2 \leq \boldsymbol{q}.x_1\}$




System is safe and verified by the inductive invariant

# Automaton model of AEB

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

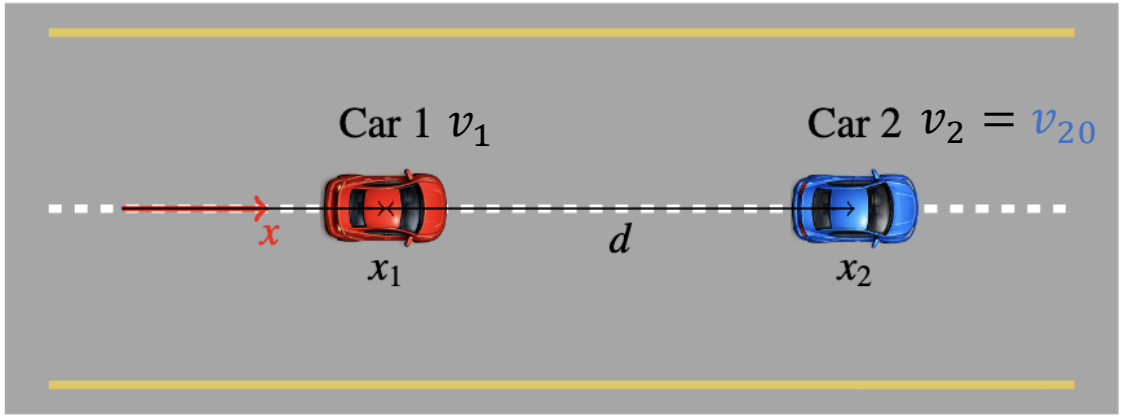
- $Q$: $[x_1, x_2, v_1] \in \mathbb{R}^3$
- $Q_0 = \{[x_1 = x_{10}, x_2 = x_{20}, v_1 = v_{10}]\}$
- $D \subseteq Q \times Q$ written as a program

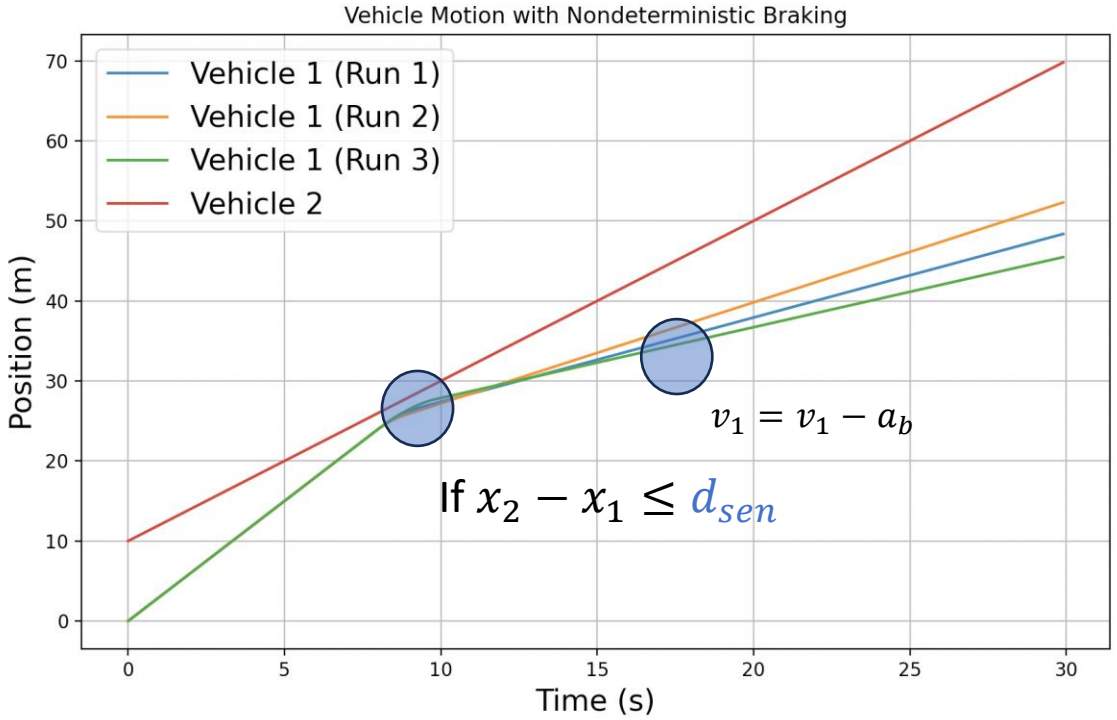
  If $x_2 - x_1 \leq d_{sen}$

  $v_1 = v_1 - a_b$

  $x_2 = x_2 + v_2$

  $x_1 = x_1 + v_1$



# Automaton model of AEB

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

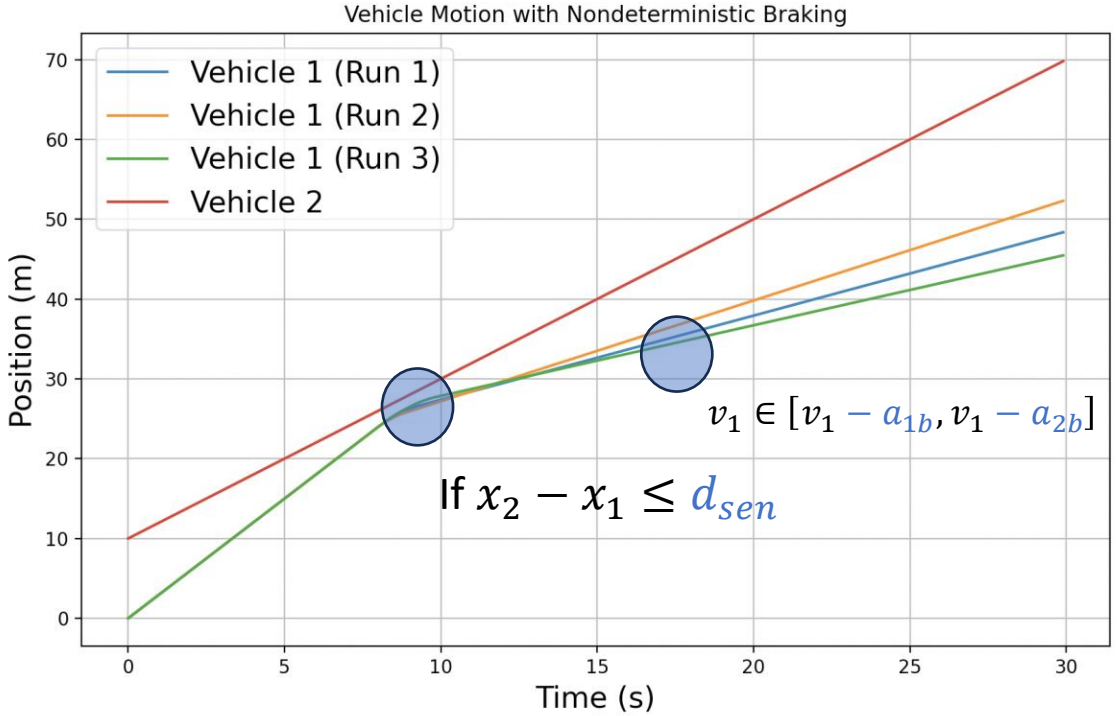
- $Q: \mathbb{R}^3; \boldsymbol{q} \in Q\ \boldsymbol{q}.x_1, \boldsymbol{q}.x_2 \in \mathbb{R}$
- $Q_0 = \{\boldsymbol{q} \mid [\boldsymbol{q}.x_1 = x_{10}, \boldsymbol{q}.x_2 = x_{20}, \boldsymbol{q}.v_1 = v_{10}]\}$
- $(\boldsymbol{q}, \boldsymbol{q}') \in D$ iff

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

  $\boldsymbol{q}'.v_1 = \boldsymbol{q}.v_1 - a_b$

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

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



# AEB

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

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

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

The invariance proof will relate *total time of braking* with the initial separation. We need a timer

# Checking Inductive Invariant for AEB

timer = 0
If $x_2 - x_1 \leq d_s$
  If $v_1 \geq a_b$
    $v_1 = v_1 - a_b$
    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. $I_1$: $\text{timer} + \frac{v_1}{a_b} \leq \frac{v_{10}}{a_b}$.

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 $q \in I_1$.

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

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

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

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

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

# 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]

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

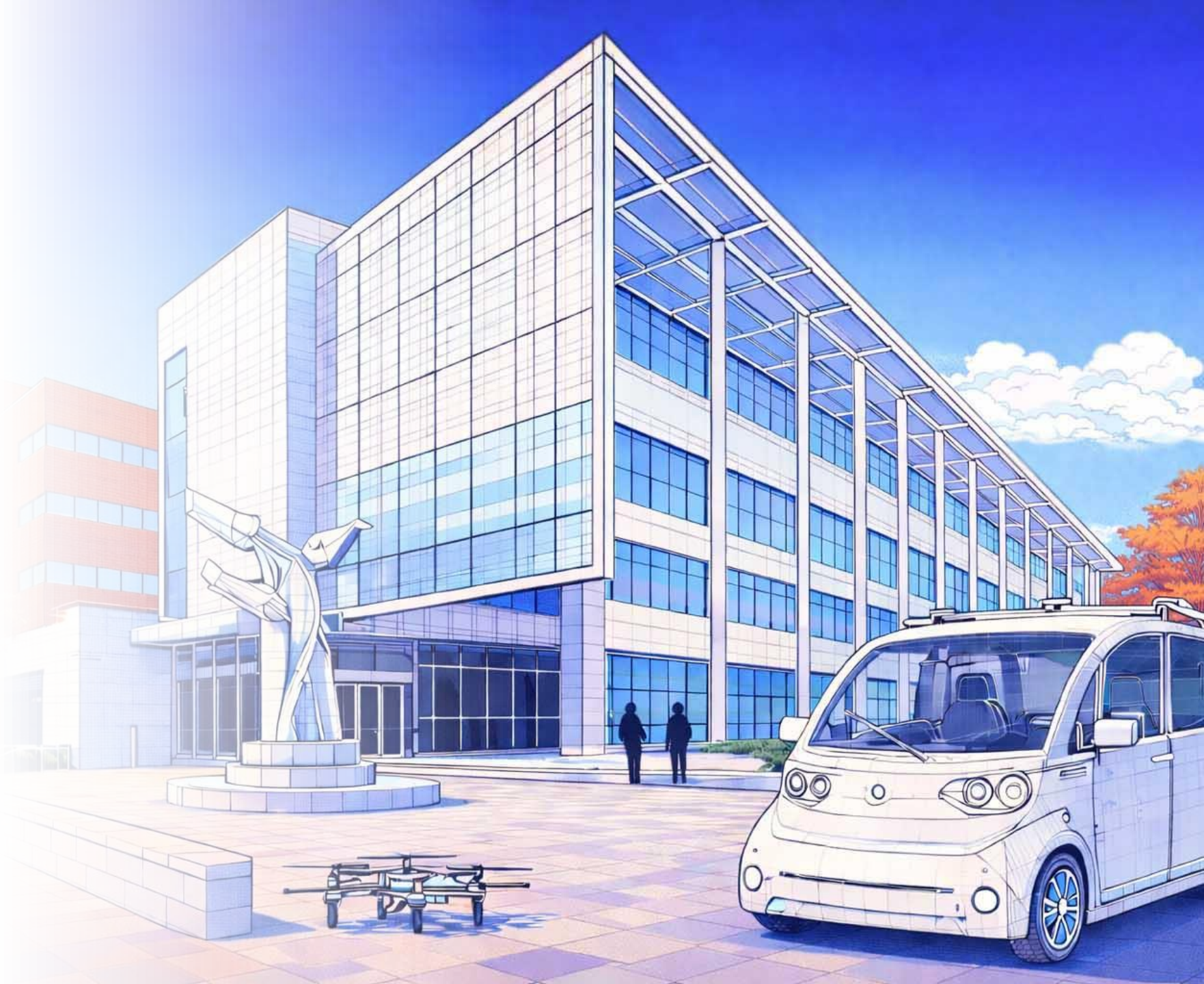
> 0 [By Assumption]

# 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 is a special class of requirements
- Reachability analysis can prove safety automatically*
- Inductive invariants over-approximating reachable states give another method for proving safety

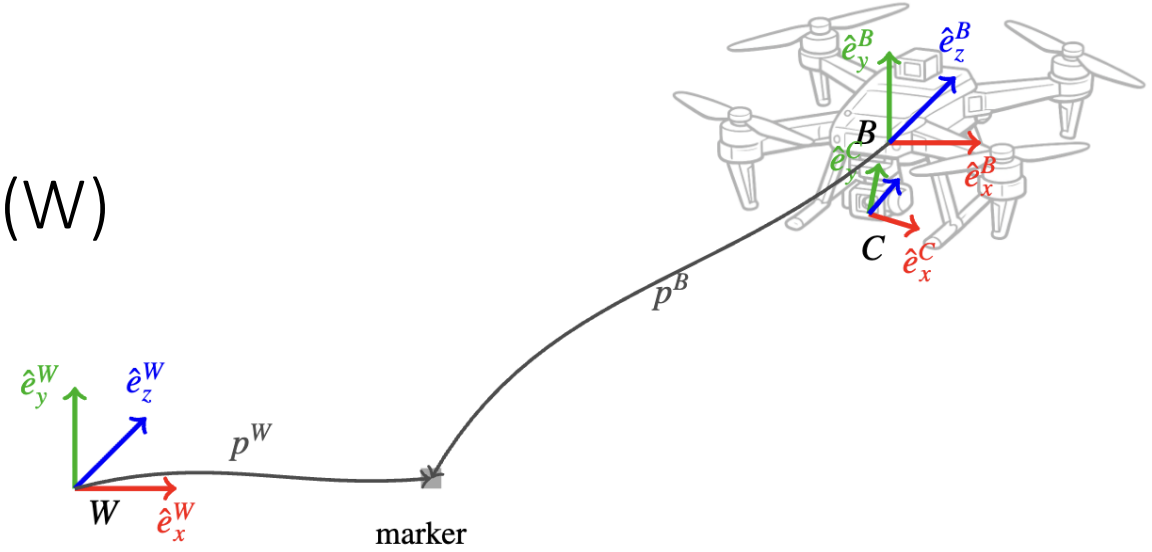
# Outline

- Safety Verification
- Coordinate transforms



# Coordinate frames

- Different coordinate frames are used for describing different aspects of an autonomous system, E.g.,
- position from GPS in world coordinates (W)
- position of a marker as seen by camera: camera frame (C)
- torque: body frame (B)
- What is a coordinate frame?
- How are different coordinates related?



# Coordinate frame

- A coordinate frame (W) is defined by its origin $O_W$ and 3 axes $\hat{e}_x^W, \hat{e}_y^W, \hat{e}_z^W$
- Let $p^W = [x^W, y^W, z^W]$ and $p^B = [x^B, y^B, z^B]$ be the coordinates of a point $p$ in the $W$ frame and $B$ frame
- We would like to convert $p^W$ to $p^B$ and vice versa
- If the coordinate frames were just translations of each other:
- Let the origin of B in the frame of W be $p_{O_B}^W$ ; this is the displacement of $O_B$ from $O_W$ in the $W$ frame
- Then, $p^W = p_{O_B}^W + p^B \qquad p^B = p^W - p_{O_B}^W$
- What if the frames are rotated? Next time.