

# Principles of Autonomy

## Fall 2025 Lecture 3: Checking Safety

Professor: Huan Zhang

Aug 26, 2025

<https://publish.illinois.edu/safe-autonomy/>

<https://huan-zhang.com>

huanz@illinois.edu



# More about labs and MPs

- ▶ More time for forming your team: **MP0 has been changed to individual**
- ▶ Deadline for finalizing all teams: Friday, **Sep 5**
- ▶ **All team members must be in the same lab session**
- ▶ You can switch your lab section through the regular course registration portal (if the section you want to switch to is not full)
- ▶ **MP1 and HW1** will be released this **Friday 9 am** - check Gradescope



# More about labs and MPs

- ▶ If you cannot make the switch yourself, because the section you want to switch to is always full, send a request to the **lead TA Hanna Chen** (hannac4@illinois.edu) **on Tuesday, Sep 2 (TODAY!)**
  - ▶ cc your email to the **corresponding lab TAs**: the one you are moving out from and the one you are moving to
  - ▶ **early/late requests will not be handled**
- ▶ You will be notified whether your lab switch is approved by the end of Wednesday, Sep 3
  - ▶ Not guaranteed - we need to balance the enrollments of each section
- ▶ All teams must be formed by **Friday, Sep 5**
  - ▶ You will need to submit the team member names - pay attention to announcements on **Campuswire**
  - ▶ **All team members must be in the same lab section**



# Outline

- ▶ Safety requirements
- ▶ Reachability
  - ▶ Data structures
- ▶ Inductive invariants
  - ▶ Example



# Recall: Automata

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

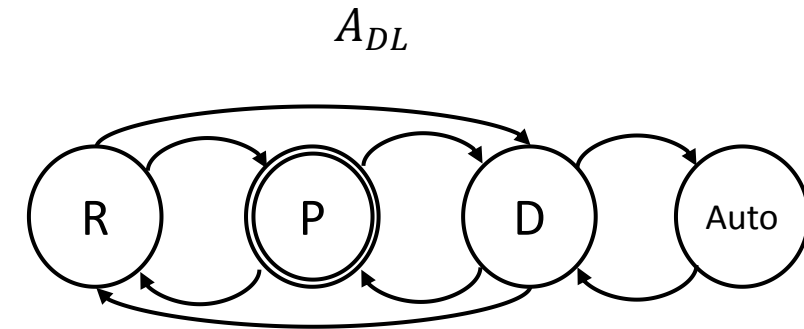
- ▶  $Q$  is a set of **states**
- ▶  $Q_0 \subseteq Q$  is a set of **initial states**
- ▶  $D \subseteq Q \times Q$  is a set of **transitions**

An **execution** of an automaton  $A$  is a finite or infinite sequence of states  $\alpha = q_0, q_1, q_2, \dots$  such that

- ▶  $q_0 \in Q_0$
- ▶ For all  $i$  in  $\alpha$ ,  $(q_i, q_{i+1}) \in D$
- ▶ We write  $q_{i+1} = D(q_i)$

A nondeterministic automata has many executions

E.g.  $P, D, P, D, \dots$  ;  $P, D, \text{Auto}; \dots$



# Requirements and Counter-examples

A **requirement** defines a collection of executions

$$R_{noAuto} = \{\alpha \mid \forall i \alpha_i \neq Auto\};$$

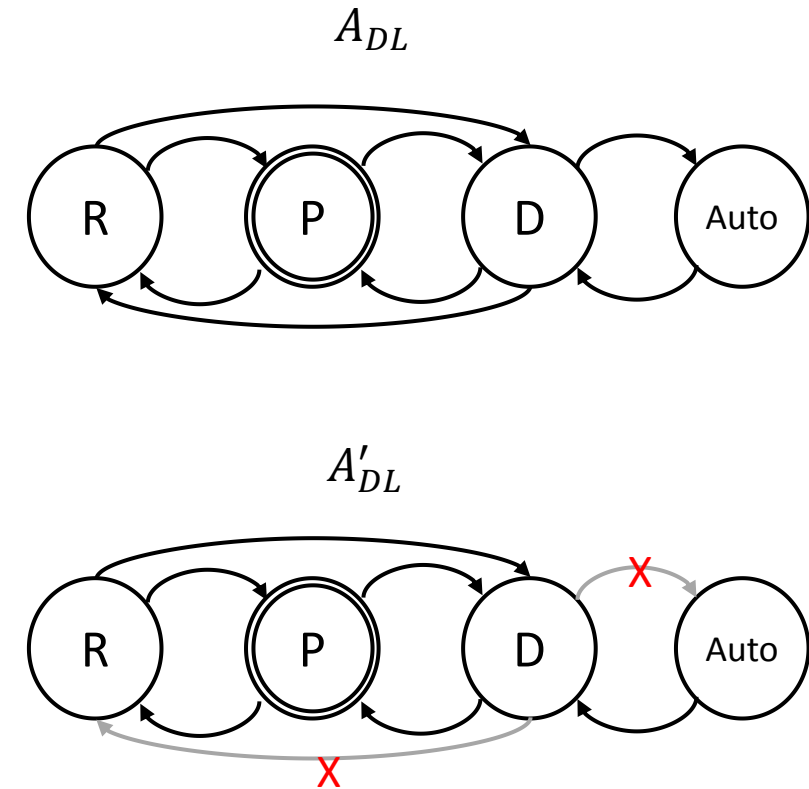
$$R_{noD2R} = \{\alpha \mid \forall i \text{ if } \alpha_i = D, \alpha_{i+1} \neq R\}$$

An automaton  $A$  **satisfies** a requirement  $R$  if **all** executions of  $A$  are contained in  $R$

$A_{DL}$  does not **satisfy** either requirement  $R_{noAuto}$   $R_{noD2R}$  because there are **counter-example** executions  $\alpha^{(1)} = P, D, Auto$  and  $\alpha^{(2)} = P, D, R$

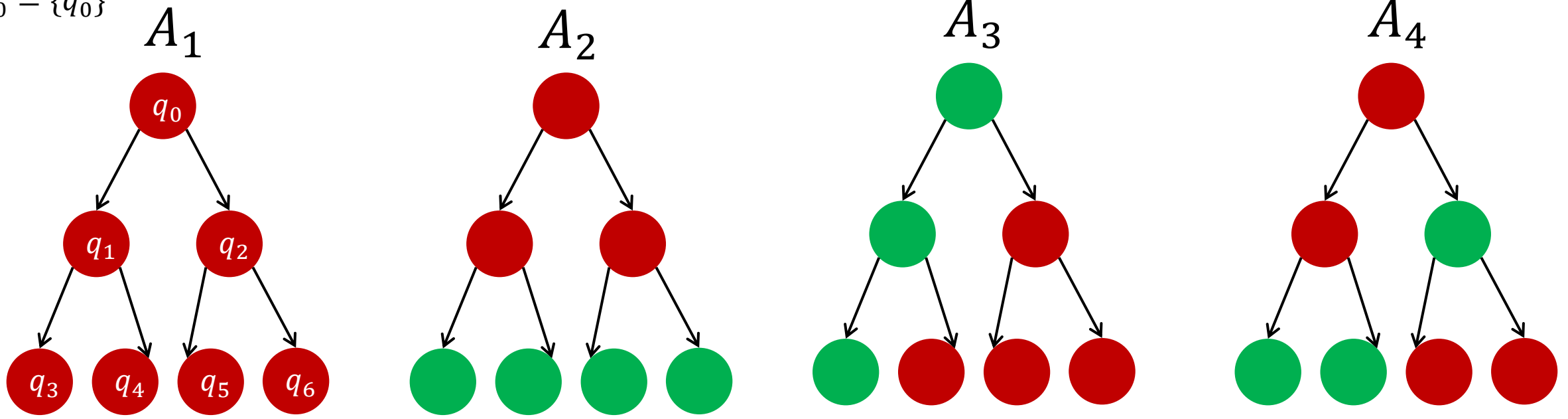
$A'_{DL}$  satisfies both the requirements

**Verification problem:** Given an automaton  $A$  and a requirement  $R$ , **decide** whether all executions of  $A$  are in  $R$  or find a counter-example that is outside  $R$



# Automata and requirements

$$Q_0 = \{q_0\}$$



Which automata satisfy the following requirements

1.  $R_{\text{nevergreen}} = \{\alpha \mid \forall i, \alpha_i.\text{color} \neq \text{green}\}$

2.  $\text{Unsafe} = \{q \mid q.\text{color} = \text{green}\}$

3.  $R_{\text{eventuallygreen}} = \{\alpha \mid \exists i, \alpha_i.\text{color} = \text{green}\}$

4.  $R_{\text{neverred}} = \{\alpha \mid \exists i, \alpha_i.\text{color} \neq \text{red}\}$



# Safety requirements and verification

A **safety requirement** is a requirement that states that no execution should reach a certain **set of bad (or unsafe) states**  $U \subseteq Q$

$$R_{noAuto} = \{\alpha \mid \forall i, \alpha_i \neq Auto\}$$

$$\text{safety } U = \{Auto\}$$

$$R_{nocollision} = \{\alpha \mid \forall i, \alpha_i.x_2 > \alpha_i.x_1\}$$

$$\text{safety } U = \{q \mid q.x_2 - q.x_1 \leq 0\}$$

$$R_{noD2R} = \{\alpha \mid \forall i, \text{ if } \alpha_i = D \text{ then } \alpha_{i+1} \neq R\} \quad \text{not a safety requirement}$$

$$R_{follows} = \{\alpha \mid \exists i, 2 \geq \alpha_i.x_2 - \alpha_i.x_1 \geq 1\} \quad \text{not a safety requirement (depending on the entire execution, not a set of states)}$$





# Safety verification: Reachable states, invariants

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

Counter-examples of safety are finite executions  $\alpha = q_0 \dots q_k, q_k \in U$

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

Absence of a counter-example **proves** that the automaton is safe



# Safety verification: Reachable states, invariants

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

Counter-examples of safety are finite executions

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

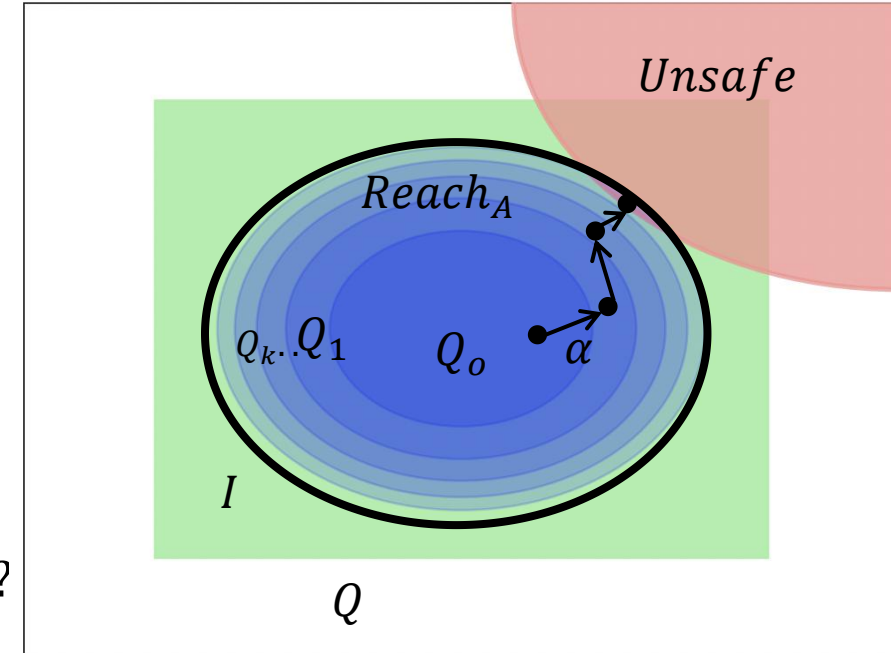
The **reachable states** of  $A$  is the set  $Reach_A \subseteq Q$  of all the states that are reachable

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

Further, sufficient to compute an over-approximation  $Reach_A \subseteq I$  such that  $I \cap U = \emptyset$

An **invariant** of  $A$  is any set of state  $I \subseteq Q$  such that  $Reach_A \subseteq I$



# Computing reachable sets

Computing the **reachable set** of  $A$  allows us to verify safety

Reachability algorithm

$R_0 = Q_0$  // Sets of states

$i = 0$

**do**

$i = i + 1$

$R_i = \{q' \mid (q, q') \in D, q \in R_{i-1}\} \cup R_{i-1}$

**Until**  $R_i = R_{i-1}$

**Return**  $R_i$

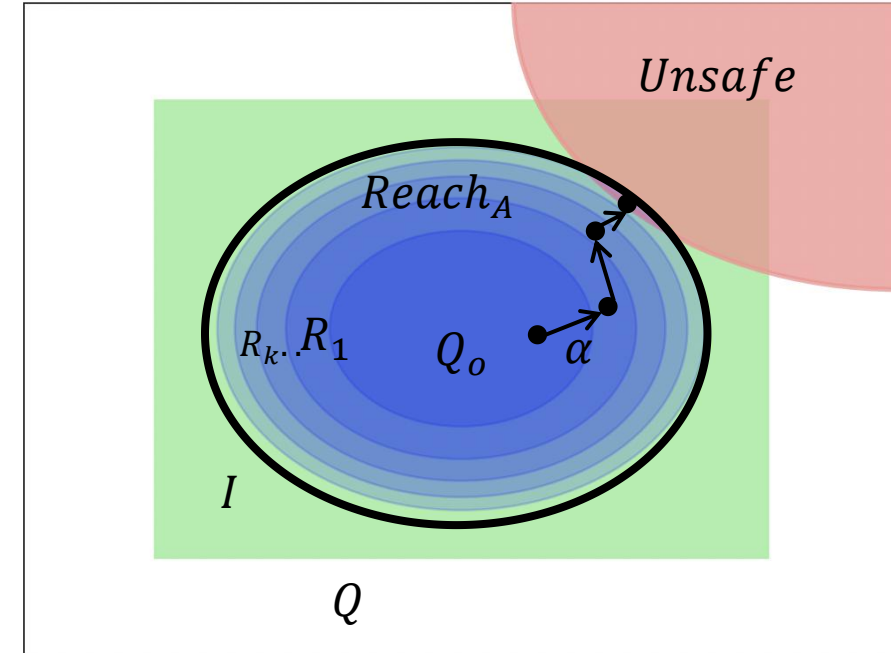
All states that are reachable within  $k$  steps  $Reach_A(0, k) = R_k$

If the algorithm terminates then  $Reach_A$  is the final output of the algorithm

If  $Reach_A \cap Unsafe = \emptyset$  then we have a proof of safety

Algorithm may not terminate and computing  $R_{i+1}$  for one-step can also be challenging for uncountable  $R_i$

Verse tool you will learn to use in MP0 performs reachability analysis



[Verse: A Python library for reasoning about multi-agent hybrid system scenarios](https://github.com/AutoVerse-ai/Verse-library) [arxiv](#) Li et al. CAV 2023.

<https://github.com/AutoVerse-ai/Verse-library>



# Computing reachable sets and over-approximations

$$R' = \text{Post}(R) = D(R) = \{q' \mid (q, q') \in D, q \in R\}$$

Reachability algorithm

$R_0 = Q_0$

$i = 0$

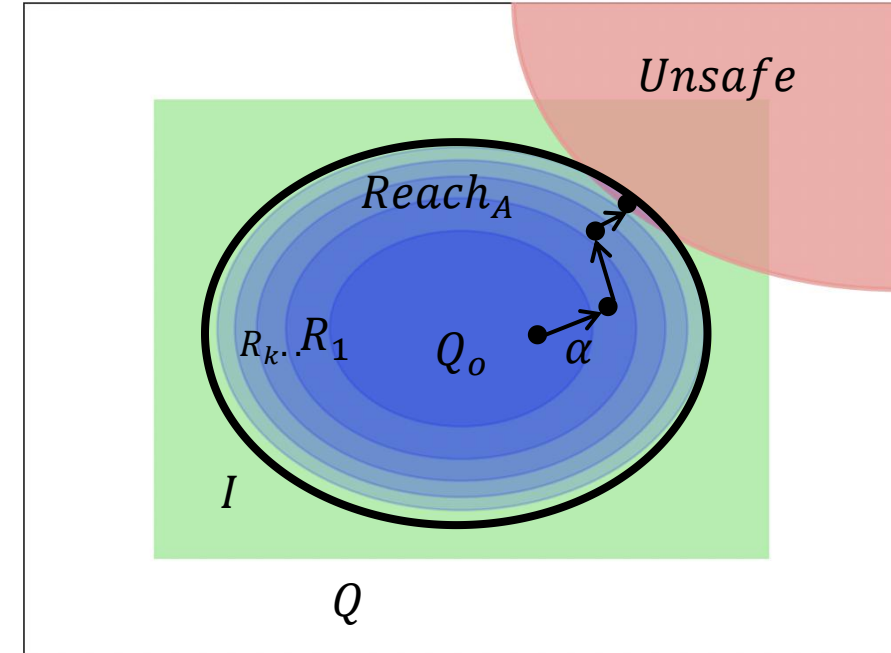
**do**

$i = i + 1$

$R_i = \text{Post}(R_{i-1}) \cup R_{i-1}$

**Until**  $R_i = R_{i-1}$

**Return**  $R_i$



Here the **Post**(R) defines the set of states that can reach in one-step, starting from a set of states R

**Exercise:** Show that **Post** is monotonic, i.e., If  $S_1 \subseteq S_2$  then  $\text{Post}(S_1) \subseteq \text{Post}(S_2)$ .



# Computing reachable sets and over-approximations

$$R' = \text{Post}(R) = D(R) = \{q' \mid (q, q') \in D, q \in R\}$$

Reachability algorithm

$$R_0 = Q_0$$

$$i = 0$$

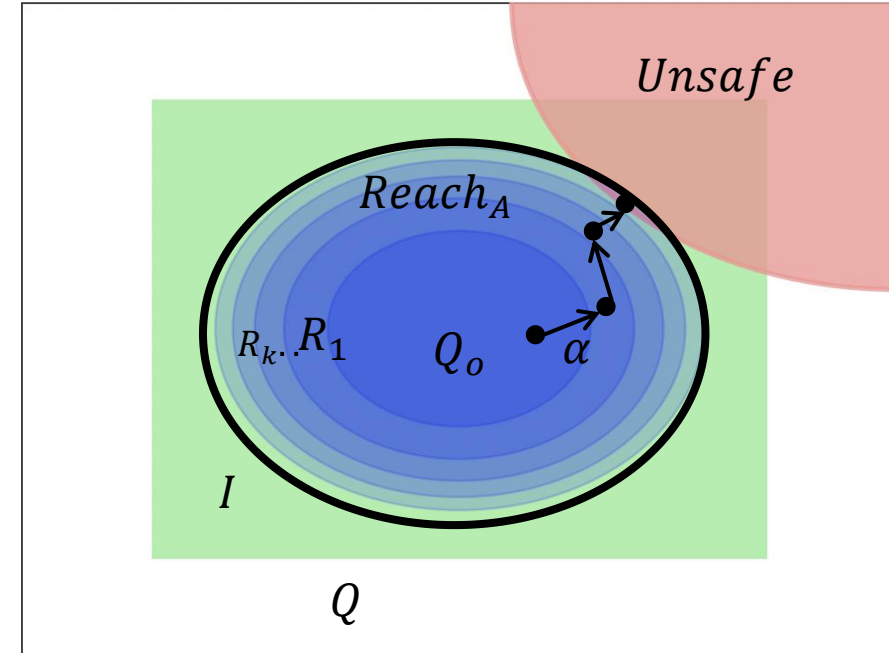
**do**

$$i = i + 1$$

$$R_i = \text{Post}(R_{i-1}) \cup R_{i-1}$$

**Until**  $R_i = R_{i-1}$

**Return**  $R_i$



All states that are reachable in exactly k steps  $\text{Reach}_A(k, k) = \text{Post}^k(Q_0)$

If  $\overline{\text{Post}}(S_1)$  computes an **over-approximation** of  $\text{Post}(S_1)$  then the above algorithm computes an over-approximation of  $\text{Reach}_A \subseteq \overline{\text{Reach}_A}$

$\overline{\text{Reach}_A} \cap \text{Unsafe} = \emptyset$  proves safety, but  $\overline{\text{Reach}_A} \cap \text{Unsafe} \neq \emptyset$  does not imply that there is a real counterexample



# Data structures for representing sets

- ▶ Intervals:  $R_i: [a, b] \in \mathbb{R}$ 
  - ▶  $x := x + 5 \rightarrow R_{i+1} = [a + 5, b + 5]$
  - ▶ Similarly for other operations  $+, -, \times$  any monotonic operation
- ▶ Hyperrectangles:  $R_i: [\mathbf{a}, \mathbf{b}] \in \mathbb{R}^n$
- ▶ Polytopes:  $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]$  represented by the vertices
- ▶ Ellipsoids: represented by a center and a shape matrix
- ▶ Star sets: represented by a center, basis vectors, and a predicate

Reachability algorithm  
needs to compute

$$R_{i+1} = D(R_i) \cup R_i$$

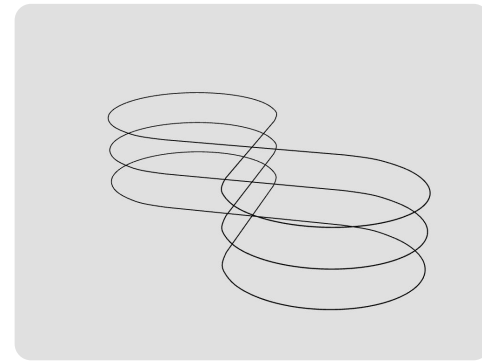
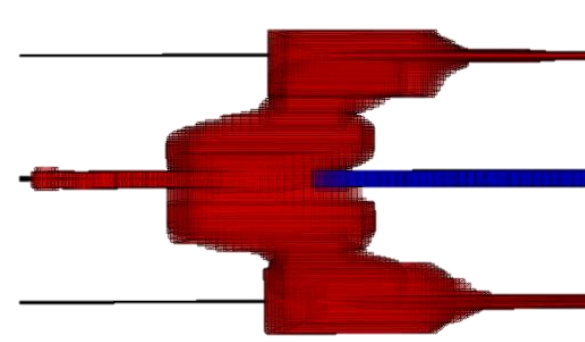
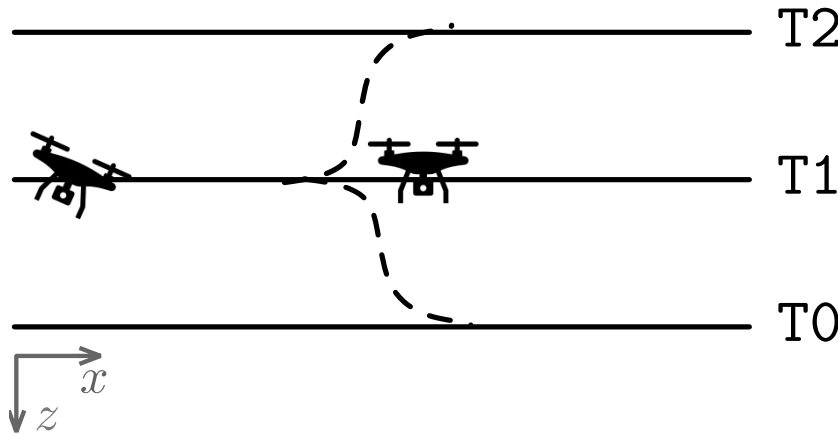
Different representations allow different operations to be performed on the sets efficiently

In building a reachability algorithm this data structure is an important choice



# Verse: Python library for reachability analysis (MP0)

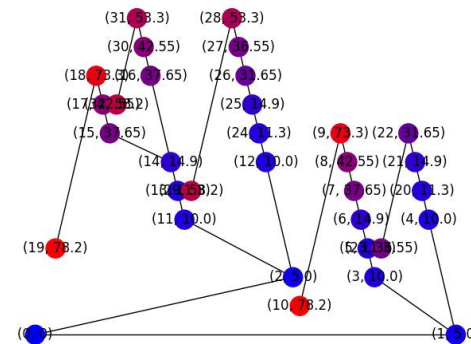
```
class Mode(Enum):
    Normal = auto()
    Up = auto()
    ...
class Track(Enum):
    T0 = auto()
    T1 = auto()
    ...
class State:
    x: float
    y: float
    ...
    mode: Mode
    track: Track
```



```
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
            ...
```

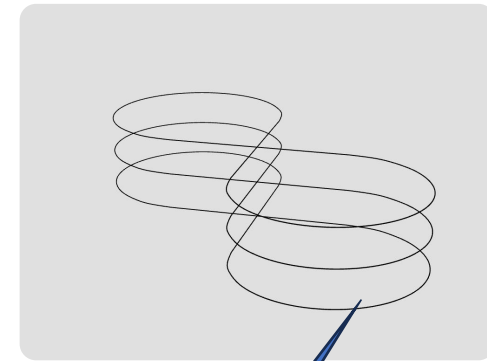
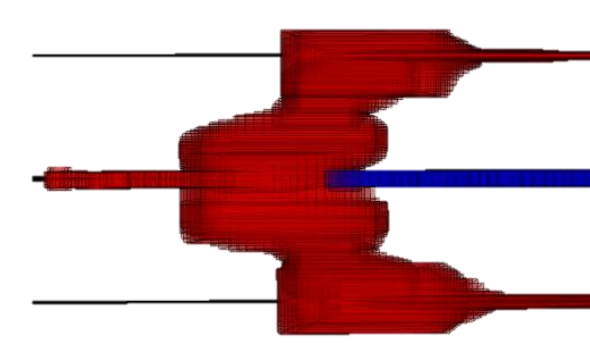
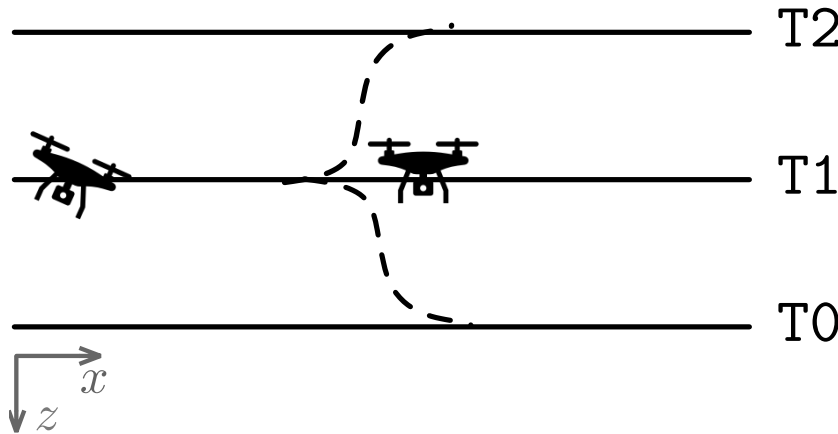
```
q1 = QuadrotorAgent("q1", ...) // Defines the dynamics
q1.set_initial([...], (Mode.Normal, Track.T1))
scenario.add_agent(q1)
q2 = ...
scenario.set_map(M5())
scenario.simulate(...)
scenario.verify(...)
```

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



# 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
```



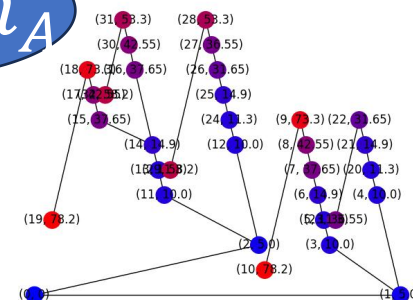
```
def decisionLogic(ego: State, others: List[State], map):
    if ego.mode == Normal:
        if any(isClose(ego, other) for other in others):
            if map.exist(ego.track, ego.mode, Up):
                next.mode = Up
                next.track = map.h(ego.track, ego.mode, Up)
            if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

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

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

*Unsafe*

*Reach<sub>A</sub>*





# Reachable states and invariants

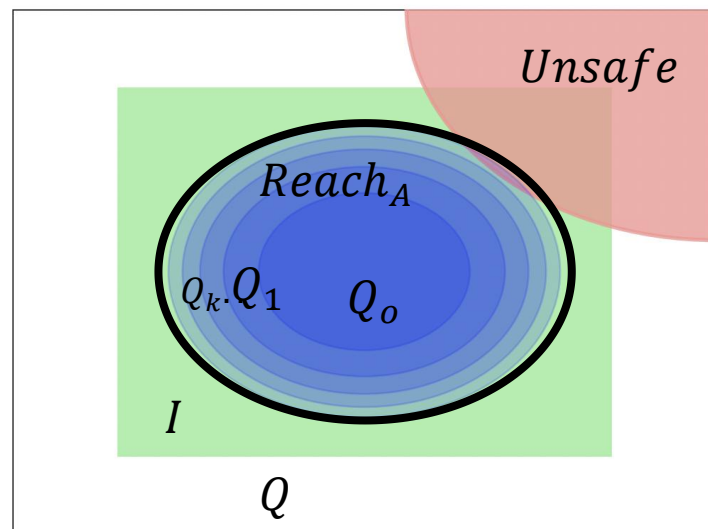
An **invariant** of  $A$  is any set of states  $I \subseteq Q$  such that  $\text{Reach}_A \subseteq I$

An invariant proves safety if  $I \cap \text{Unsafe} = \emptyset$

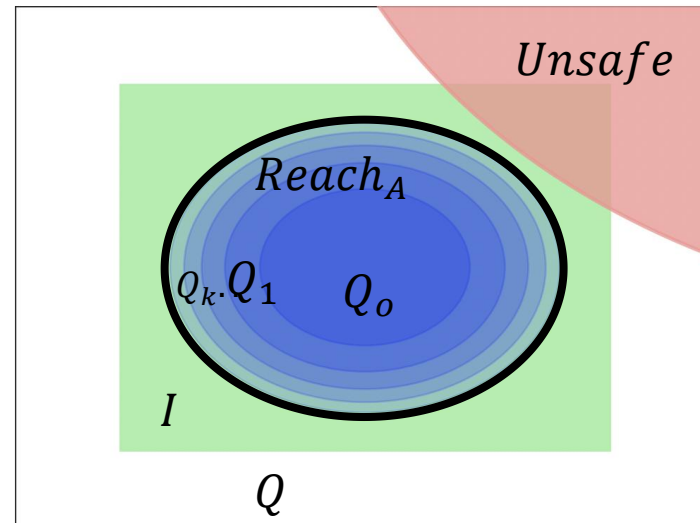
In general computing or finding invariants is also hard; it is easier to check whether a given set is an invariant

If  $Q_0 \subseteq I$  and  $\text{Post}(I) \subseteq I$  then  $I$  is an invariant, i.e.,  $\text{Reach}_A \subseteq I$ .

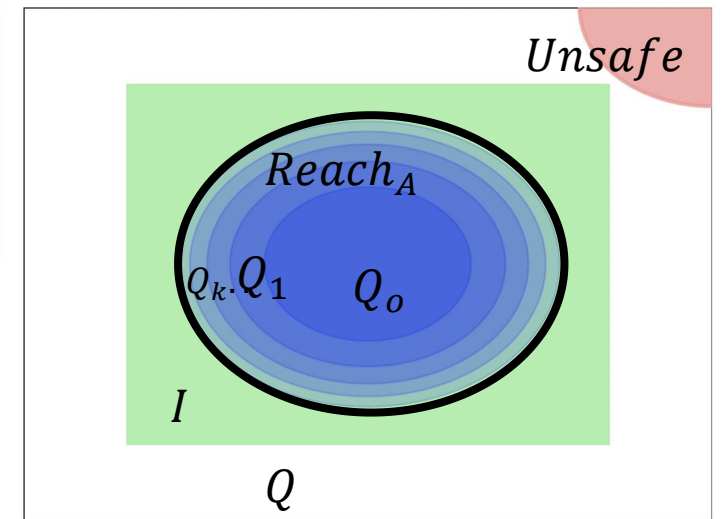
Such invariants are called **inductive invariants**



Unsafe with counter-example



Safe but not proved by  
invariant  $I$



Safety proved by computing invariant  $I$



# Inductive invariants

Theorem. If  $Q_0 \subseteq I$  and  $Post(I) \subseteq I$  then  $I$  is an invariant, i.e.,  $Reach_A \subseteq I$ .

Such invariants are called **inductive invariants**

**Proof.** Consider any reachable state  $q \in Reach_A \subseteq Q$

By definition of reachable state, there is an execution  $\alpha$  with  $\alpha_k = q$

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

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

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

$(\alpha_k, q) \in D$  implies  $q \in Post(\alpha_k)$  which implies [by monotonicity of  $Post$  and  $\alpha_k \in I$ ]:

$$q \in Post(\alpha_k), Post(\alpha_k) \subseteq Post(I) \Rightarrow q \in Post(I) \subseteq I$$

Take away: Guess a candidate inductive invariant  $I \cap Unsafe = \emptyset$  and check  $Q_0 \subseteq I$  and  $Post(I) \subseteq I$ . If successful then safety is verified.



# Automaton model a bouncing ball

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

- ▶  $Q = \mathbb{R}^2$ 
  - ▶  $\mathbf{q} \in Q \ \mathbf{q}.x, \mathbf{q}.v \in \mathbb{R}$
- ▶  $Q_0 = \{\mathbf{q} \mid \mathbf{q}.x = h, \mathbf{q}.v = 0\}$
- ▶  $D: \mathbb{R}^2 \rightarrow \mathbb{R}^2$  written as a program

If  $x \leq 0$  and  $v < 0$

$$v = -v; x = 0$$

else

Assume each transition is one “unit time step”:

$$v = v - g$$

$$x = x + v - \frac{1}{2}g$$

$$q'.x = q.x + q.v - \frac{1}{2}g$$

$$q'.v = q.v - g$$



# Automaton model a bouncing ball

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

- $Q = \mathbb{R}^2$
- $Q_0 = \{q \mid q.x = h, q.v = 0\}$
- $D: \mathbb{R}^2 \rightarrow \mathbb{R}^2$  written as a program

We write  $q' = D(q)$

If  $x \leq 0$  and  $v < 0$

$$v = -v; x=0$$

else

$$v = v - g$$

$$x = x + v - \frac{1}{2}g$$

$$q.x = q.x + q.v - \frac{1}{2}g$$

$$q'.v = q.v - g$$

Candidate invariant  $I_1: x \leq h$

Can we prove this inductively?

Use If  $Q_0 \subseteq I_1$  and  $Post(I_1) \subseteq I_1$  then  $I_1$  is an invariant, i.e.,  $Reach_A \subseteq I_1$ .

Base: We have to show  $Q_0 \subseteq I_1$  that is for any  $q \in Q_0$ ,  $q \in I_1$

Consider  $q \in Q_0$ ,  $q.x = h$

Inductive step: We have to show  $Post(I_1) \subseteq Post(I_1) \subseteq I_1$ , that is, for any  $q \in I_1$ ,  $D(q) \in I_1$

What do we know about  $q \in I_1$ ?  $q.x \leq h$ .

Two cases to consider based on if-else

(a) if  $q.x \leq 0$  then  $q'.x = 0$ , and therefore  $q' \in I_1$

(b) Otherwise  $q.x \leq h$  and  $q'.x = q.x + q.v - \frac{1}{2}g \leq h + q.v - \frac{1}{2}g$

$I_1$  does not impose any constraints on  $q.v$  to be able to show that  $q.v \leq \frac{1}{2}g$

Conclusion  $I_1: x \leq h$  is not an inductive invariant

Although it is true through physical intuitions, we cannot prove it through this invariant



# Automaton model a bouncing ball

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

- $Q = \mathbb{R}^2$
- $Q_0 = \{q \mid q.x = h, q.v = 0\}$
- $D: \mathbb{R}^2 \rightarrow \mathbb{R}^2$  written as a program

We write  $q' = D(q)$

If  $x \leq 0$  and  $v < 0$

$$v = -v; x = 0$$

else

$$v = v - g$$

$$x = x + v - \frac{1}{2}g$$

$$q.x = q.x + q.v - \frac{1}{2}g$$

$$q'.v = q.v - g$$

Candidate invariant  $I_2: v^2 = 2g(h - x)$

Can we prove this inductively?

Use If  $Q_0 \subseteq I_2$  and  $Post(I_2) \subseteq I_2$  then  $I_2$  is an invariant, i.e.,  $Reach_A \subseteq I_2$ .

Base: We have to show  $Q_0 \subseteq I_2$  that is for any  $q \in Q_0$ ,  $q \in I_2$

Consider  $q \in Q_0$ ,  $q.v^2 = 0 = 2g(h - q.x)$  (since  $q.x = h$ )

Inductive step: We have to show  $Post(I_2) \subseteq I_2$ , that is, for any

$q \in I_2$ ,  $D(q) \in I_2$ , What do we know about  $q \in I_2$ ?  $q.v^2 = 2g(h - q.x)$  --- Eq. (1)

Two cases to consider based on if-else

(a) if  $q.x \leq 0$  then  $q'.v^2 = q.v^2 = 2g(h - q.x) = 2g(h - q'.x)$  [by def of D and (1)]

(b) Otherwise  $q'.v^2 = (q.v - g)^2 = q.v^2 - 2gq.v + g^2$

$$= 2g(h - q.x) - 2gq.v + g^2 = 2g\left(h - q.x - q.v + \frac{1}{2}g\right) = 2g(h - q'.x)$$

Conclusion:  $v^2 = 2g(h - x)$  is an inductive invariant of A

Can we use this inductive invariant to prove  $x \leq h$ ?



# Revisiting AEB

If  $x_2 - x_1 \leq d_s$

If  $v_1 \geq a_b$

$v_1 = v_1 - a_b$

else

$v_1 = 0$

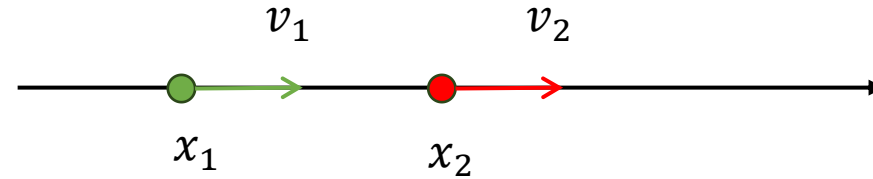
else

$v_1 = v_1$

$x_2 = x_2 + v_2$

$x_1 = x_1 + v_1$

$v_1$  never increased



initial conditions ( $x_{10}, x_{20}, v_{10}, v_{20}$ )

sensing distance ( $d_s$ )

braking acceleration ( $a_b$ )



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

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



# Invariants in AEB

Bound on total braking time in terms of velocity ( $v_{10}$ ) and deceleration ( $a_b$ )

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

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

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

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

(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.

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

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

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

Now consider the **bounds on braking time**  $I_2$ :  $\text{timer} \leq \frac{v_{10}}{a_b}$ , is it true?

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

$v_1 = v_1$

$x_2 = x_2 + v_2$

$x_1 = x_1 + v_1$





# Invariants for correctness proof

Consider any two reachable states  $q_1, q_2$ :

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

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

There could be multiple intermediate states between  $q_1$  and  $q_2$

Then, we evaluate:  $q_2.x_2 - q_2.x_1$

$$> q_1.x_2 - q_2.x_1$$

[Because the car moves forward,  $x_2$  increased]

$$> q_1.x_2 - q_1.x_1 - v_{10} \cdot \frac{v_{10}}{a_b}$$

[ $I_2 \Rightarrow \text{timer} \leq \frac{v_{10}}{a_b}$  and  $q_2.x_1 \leq q_1.x_1 + v_{10} \cdot \frac{v_{10}}{a_b}$ ]

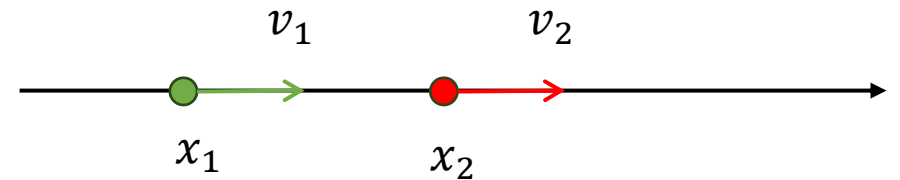
$$> d_s - \frac{v_{10}^2}{a_b}$$

[By def of  $q_1$ ]

maximum bounds on  
speed breaking time

$$> 0$$

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



# 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

