Principles of Autonomy (Spring 25)
Lecture 3: Checking Safety

Professor: Sayan Mitra

Jan 28, 2025

mitras@Illinois.edu

@Mitrasayn

Outline

- Safety requirements
- Reachability
- Inductive invariants
- Data structures

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 $\alpha = q_0, q_1, q_2, \ldots$ such that

- $q_0 \in Q_0$
- For all i in α, $(q_i, q_{i+1}) \in D$

A nondeterministic automata has many executions

E.g. P,D,P,D,… ; P, D, Auto; …

Requirements and Counter-examples

A requirement defines a collection of executions

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

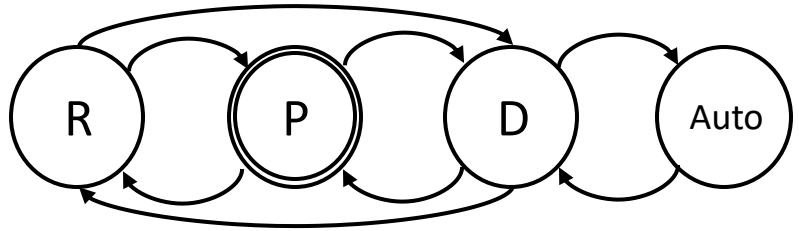
An automaton A satisfies a requirement R if ***all*** executions of A are contained in R

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

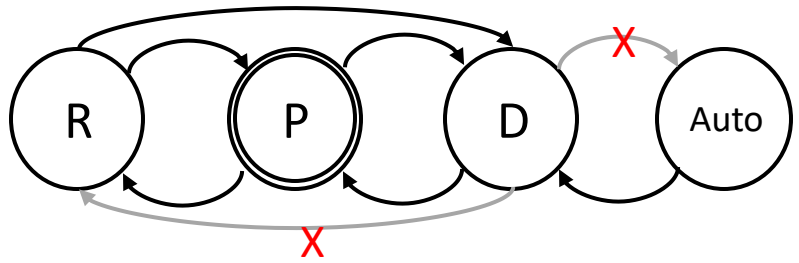
A'_{DL} satisfies both the requirements

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

A_{DL}



A'_{DL}



Safety requirements and verification

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

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

$R_{nocollision} = \{\alpha \mid \forall i\ \alpha_i.x_2 > \alpha_i.x_1\}$ safety $U = \{\boldsymbol{q} \mid \boldsymbol{q}.x_2 - \boldsymbol{q}.x_1 \leq 0\}$

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

$R_{follows} = \{\alpha \mid \exists i\ 2 > \alpha_i.x_2 - \alpha_i.x_1 > 1\}$ not a safety requirement

Safety verification: Reachable states, invariants

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

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 α of A that reaches U

Counter-examples of safety are finite executions

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

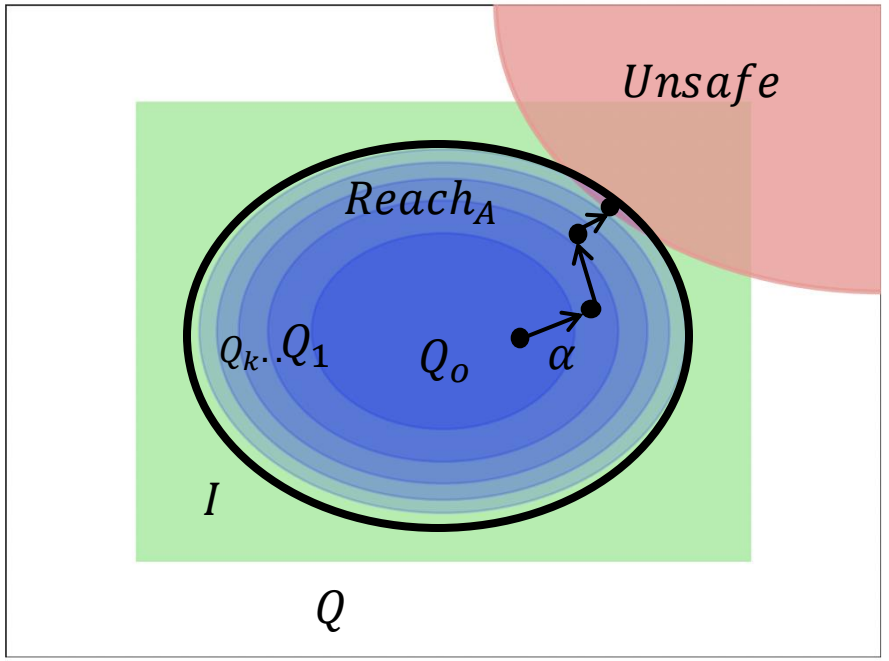
The reachable states of A is the set $Reach_A \subseteq Q$ of all the states that are reachable

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 $\mathrm{Reach_A} \subseteq I$ such that $I \cap \mathrm{U} = \emptyset$

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



Computing reachable sets

Computing the reachable set of A allows us to verify safety

Reachability algorithm

$R_0 = Q_0$ // Sets of states

$R_1 = \emptyset$

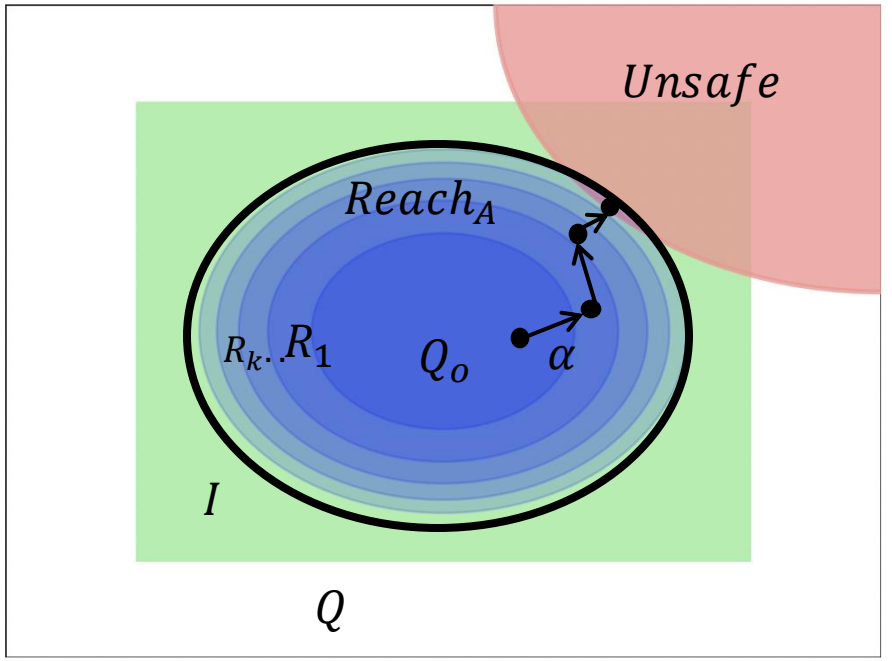
$i = 1$

While $R_i \neq R_{i-1}$

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

$i = i + 1$

Return R_i



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

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

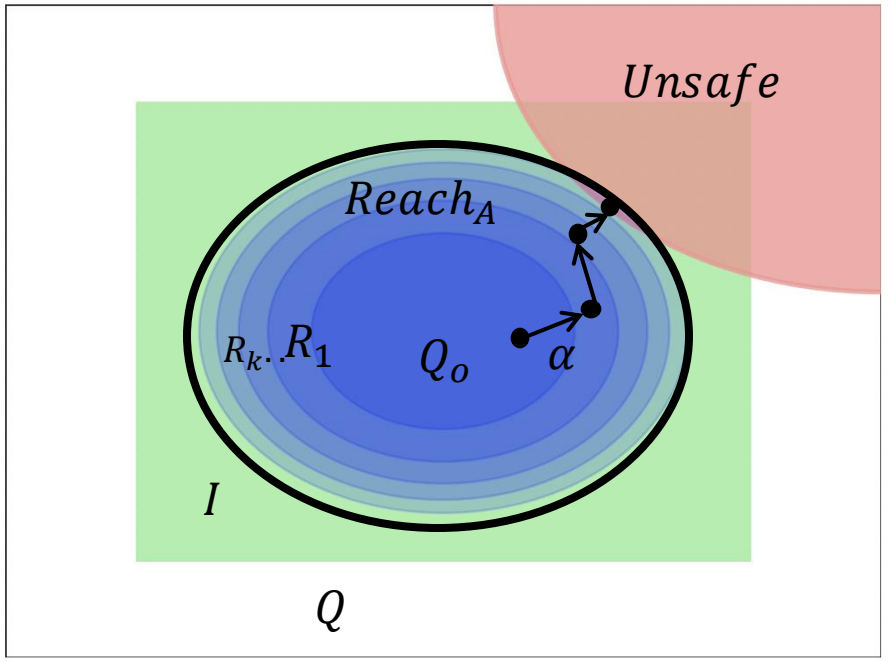
Verse: A Python library for reasoning about multi-agent hybrid system scenarios arxiv Li et al. *CAV 2023*.

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

Computing reachable sets and over-approximations

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

Reachability algorithm
$R_0 = Q_0$
$R_1 = \emptyset$
$i = 1$
While $R_i \neq R_{i-1}$
  $R_{i+1} = Post(R_i) \cup R_i$
  $i = i + 1$
Return R_i



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

All states that are reachable in exactly k steps $Reach_A(k, k) = Post^k(Q_0)$

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

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

Reachable states and invariants

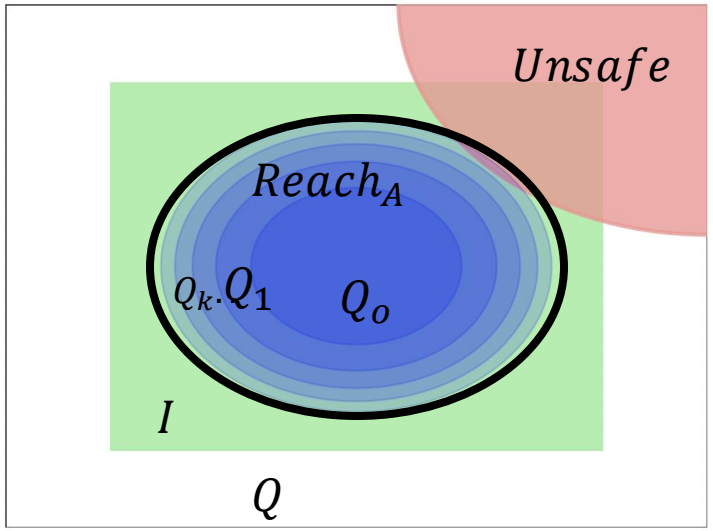
An invariant of A is any set of states $I \subseteq Q$ such that $\mathrm{Reach}_A \subseteq I$

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

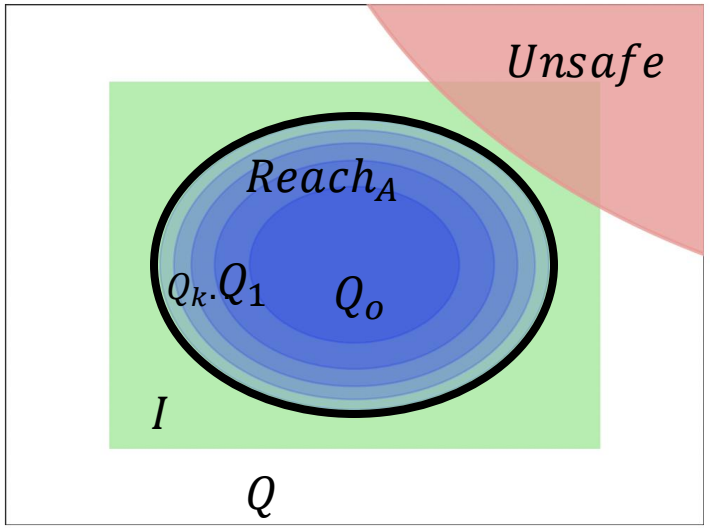
In general computing or finding invariants is also hard; it is easier to check whether a given set is an invariant

If $\mathrm{Q}_0 \subseteq I$ and $Post(I) \subseteq I$ then I is an invariant, i.e., $\mathrm{Reach}_A \subseteq I$.

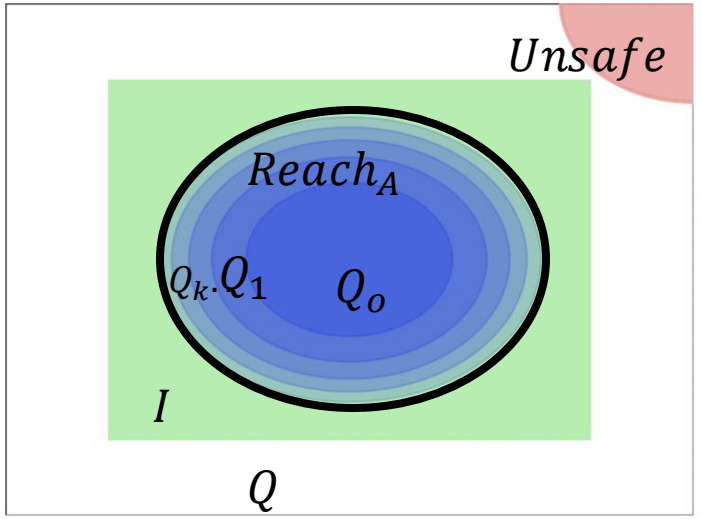
Such invariants are called inductive invariants



Unsafe with counter-example



Safe but not proved by invariant I



Safety proved by computing invariant I

Inductive invariants

Theorem. If $\mathrm{Q}_0 \subseteq I$ and $Post(I) \subseteq I$ 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 inductive inv]

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

$(\alpha_k, \boldsymbol{q}) \in D$ implies $\boldsymbol{q} \in Post(\alpha_k)$ which implies [by monotonicity of Post and $\alpha_k \in I$] $\boldsymbol{q} \in Post(I) \subseteq I$

Take away: Guess a candidate inductive invariant $I \cap Unsafe = \emptyset$ and check $\mathrm{Q}_0 \subseteq I$ and $Post(I) \subseteq I$. If successful then safety is verified.

Automaton model a bouncing ball

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

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

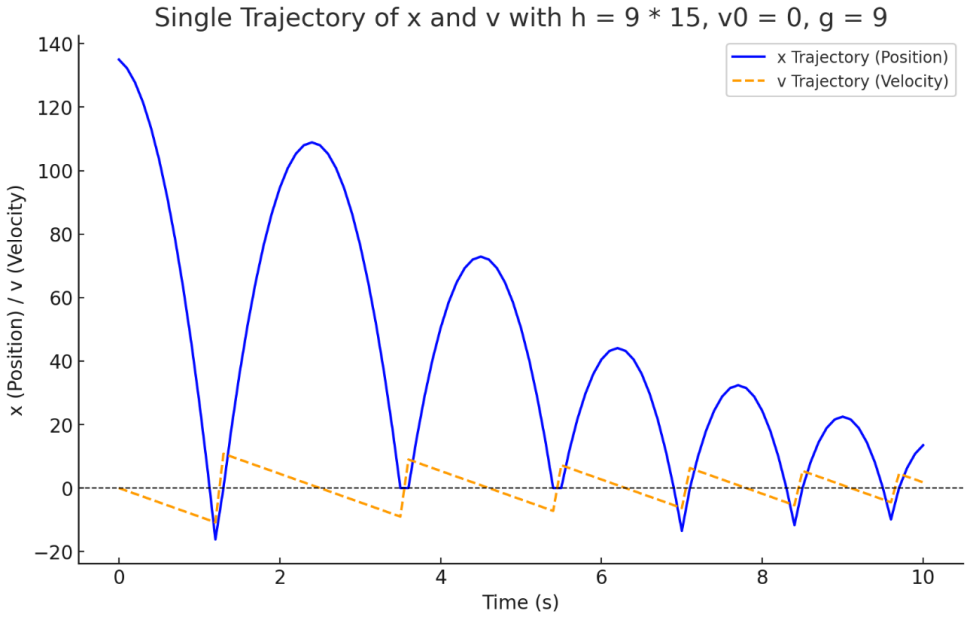
If $x \leq 0$ and $v < 0$

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

else

$$v = v - g$$

$$x = x + 2v - g$$



Note $q'.x = q.x + 2q.v - g$

Automaton model a bouncing ball

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

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

We write $q' = D(q)$

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

$v = -v; x = 0$

else

$v = v - g$

$x = x + 2v - g$

Note $q'.x = q.x + 2q.v - g$

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., $\mathbf{Reach_A} \subseteq I_1$.

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 $q \in I_1$, $\mathrm{D}(q) \in I_1$

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 $q'.x \leq 0$, and therefore $q' \in I_1$

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

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

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

Automaton model a bouncing ball

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

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

We write $q' = D(q)$

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

$v = -v$; x=0

else

$v = v - g$

$x = x + 2v + g$

Note $x' = x + 2v - g$

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

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., $\mathbf{Reach_A} \subseteq I_2$.

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

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

Inductive step: We have to show $Post(I_2) \subseteq Post(I_2) \subseteq I_2$, that is, for any $q \in I_2$, $\mathrm{D}(q) \in I_2$

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

Two cases co consider based on if-else

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

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

$= g(h - q.x - 2\ q.v) + g^2 = g(h - q.x - 2\ q.v + g) = g(h - q.x')$

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

Data structures for representing sets

- Intervals: $R_i: [a, b] \in \mathbb{R}$
  - $x \coloneqq x + 5 \rightarrow R_{i+1} = [a + 5, b + 5]$
  - Similarly for other operations +,-,x any monotonic operation
- Hyperrectangles: $R_i: [\boldsymbol{a}, \boldsymbol{b}] \in \mathbb{R}^n$
- Polytopes: $[\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_n}]$ represented by the vertices
- Ellipsoids: represented by a center and a shape matrix
- Star sets: represented by a center, basis vectors, and a predicate

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

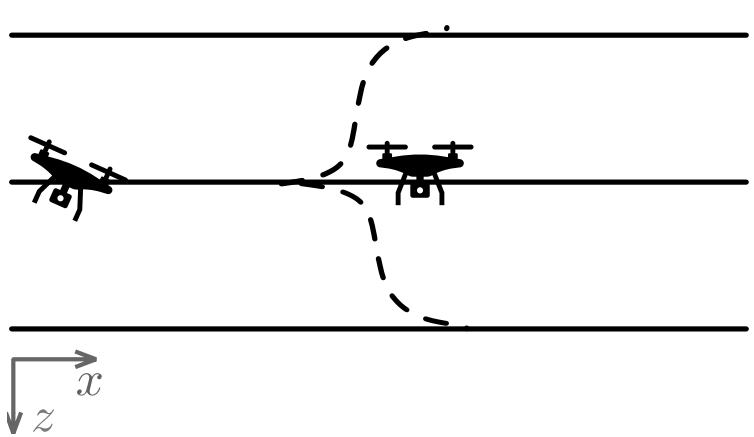
In building a reachability algorithm this data structure is an important choice

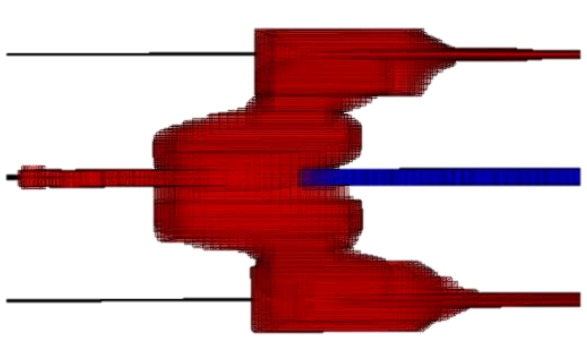
Reachability algorithm needs to compute

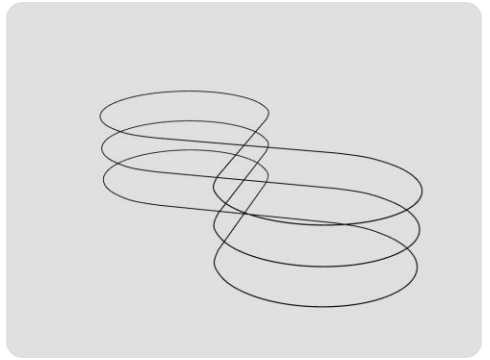
$$R_{i+1} = D(R_i) \cup R_i$$

Verse: Python library for reachability analysis (MP0)

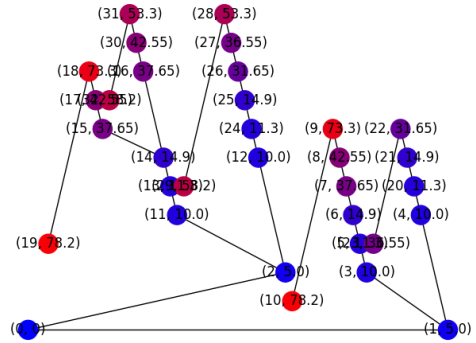
```
class Mode(Enum):
  Normal = auto()
  Up = auto()
  …
class Track(Enum):
  T0 = auto()
  T1 = auto()
  …
class State:
  x: float
  y: float
  …
  mode: Mode
  track: Track

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"
```





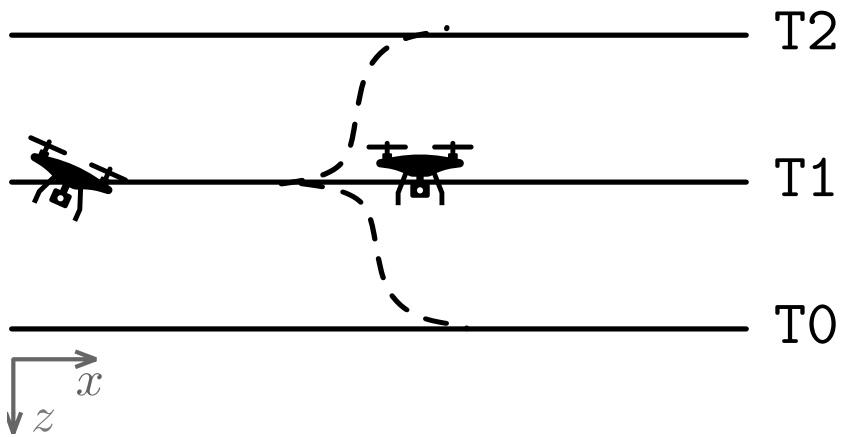


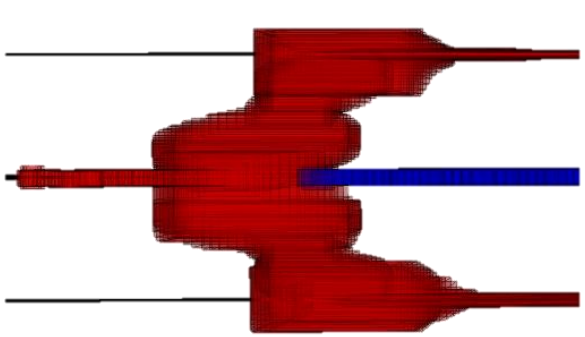
```
q1 = QuadrotorAgent(”q1", …) // Defines the dynamics
q1.set_initial([…], (Mode.Normal, Track.T1))
scenario.add_agent(q1)
q2 = …
scenario.set_map(M5())
scenario.simulate(…)
scenario.verify(…)
```

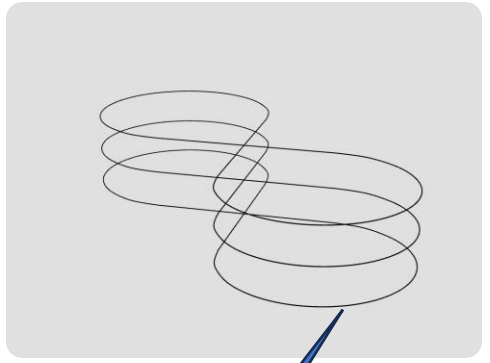


Verse: Python library for reachability analysis (MP0)

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







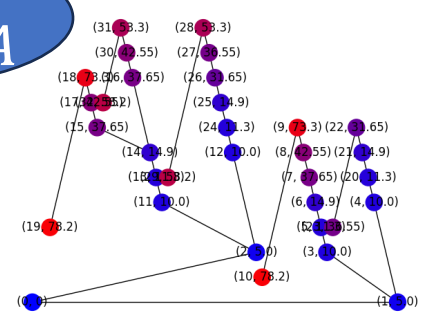
Q

D

```
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(…)
```

$Reach_A$



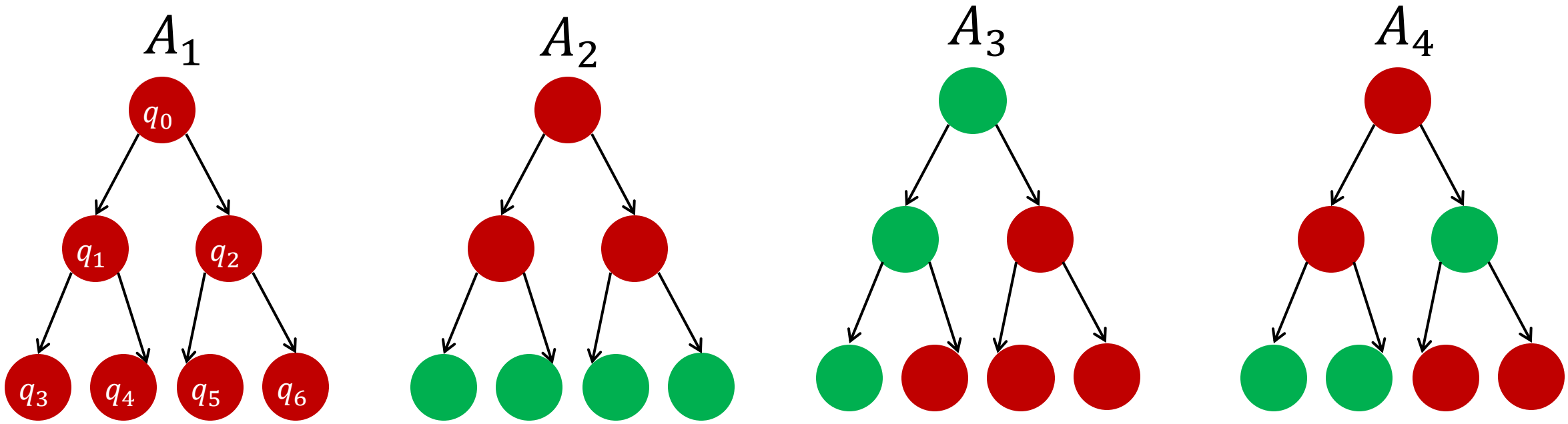
$Unsafe$

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

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

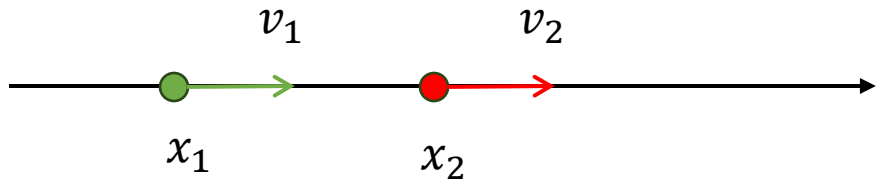
Automata and requirements



Which automata satisfy the following requirements

1. $R_{nevergreen} = \{\alpha \mid \forall i\ \alpha_i.color \neq green\}$
2. Unsafe = $\{q \mid q.color = green\}$
3. $R_{eventuallygreen} = \{\alpha \mid \exists i\ \alpha_i.color = green\}$
4. $R_{neverred} = \{\alpha \mid \exists i\ \alpha_i.color \neq red\}$

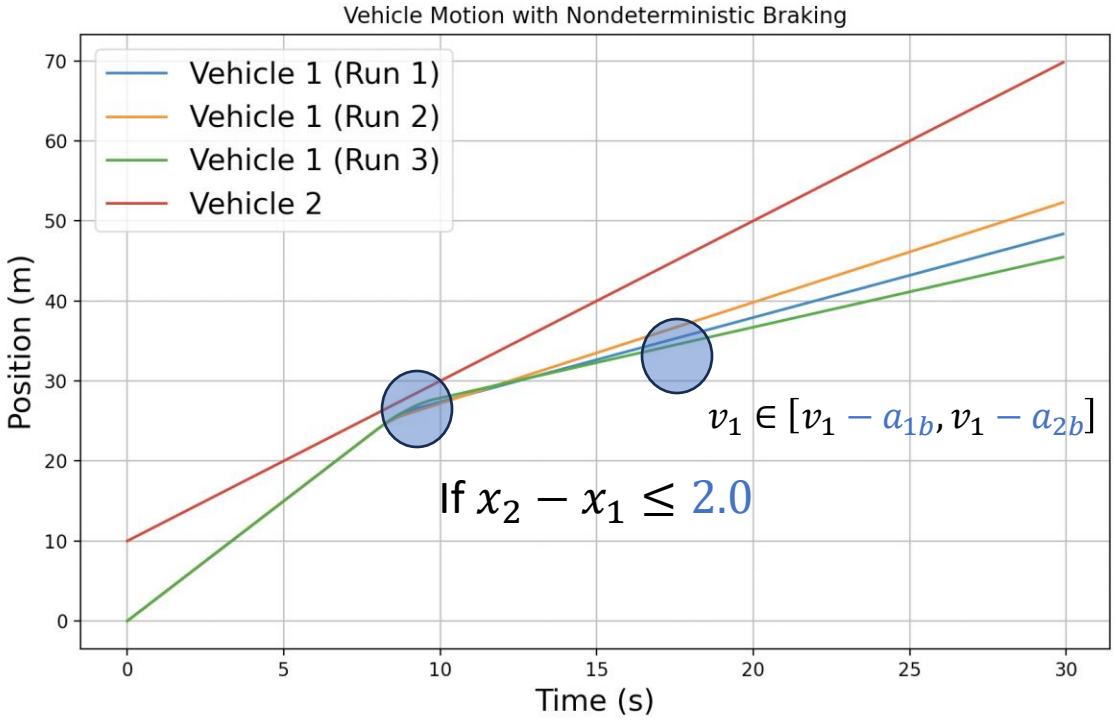
Automaton model of AEB (MP0)



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

- $Q = \mathbb{R}^4$
  - $\boldsymbol{q} \in Q$ $\boldsymbol{q}.x_1, \boldsymbol{q}.v_2 \ldots \in \mathbb{R}$
- $Q_0 = \{\boldsymbol{q} \mid \boldsymbol{q}.x_1 = x_{10}, \boldsymbol{q}.x_1 = x_{20}, \ldots\}$
- $D \subseteq \mathbb{R}^4 \times \mathbb{R}^4$ written as a program

If $x_2 - x_1 \leq 2.0$
  $v_1 \in [v_1 - a_{1b}, v_1 - a_{2b}]$
else $v_1 = v_1$
$x_2 = x_2 + v_2$
$x_1 = x_1 + v_1$



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

Modified Example

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]