Provably Safe Controller Synthesis Using Safety Proofs as Building Blocks

Yanni Kouskoulas[1] Aurora Schmidt[1]
Jean-Baptiste Jeannin[2] Daniel Genin[1] Jessica Lopez[1]

International Conference in Software Engineering Research and Innovation 2019

This work was supported by the Federal Aviation Administration (FAA)
Traffic-Alert & Collision Avoidance System (TCAS) Program Office (PO)
AJM-233: Volpe National Transportation Systems Center

Contract Nos. DTFAWA11C00074 and DTRT5715D30011

[1]Johns Hopkins University Applied Physics Laboratory
[2]University of Michigan

Outline

- Introduction
- Approach
- Controller Synthesis
- Implementation and Results
- Conclusions

Outline

- **Introduction**
- Approach
- Controller Synthesis
- Implementation and Results
- Conclusions

Problem

- Control software for complex, safety-critical robotic machinery needs safety guarantees
  - Primary controller based on machine learning approaches or neural networks may not have desired safety guarantees

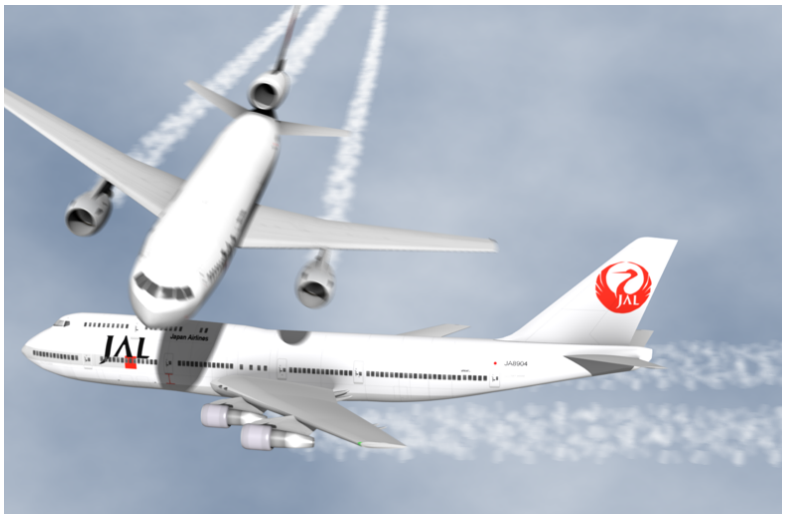


Image from airlive.net

Image from Intuitive Surgical

Motivating Question

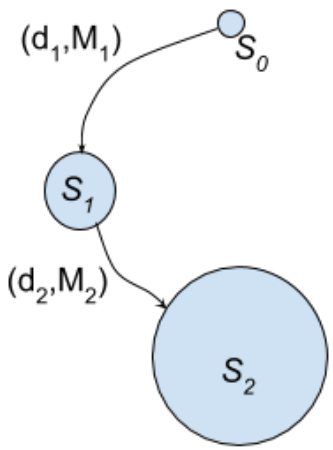
- How can we impose safety guarantees on cyber-physical systems based on machine learning techniques, or whose algorithms learn and evolve over time?

Outline

- Introduction
- **Approach**
- Controller Synthesis
- Implementation and Results
- Conclusions

Modeling System Dynamics

- Develop hybrid system models that represent sequences of discrete actions and continuous physics of the system
  - Differential equations with uncertain parameters
- Predictive, non-deterministic, continuous model represents ranges of future possibilities that encompass realistic but bounded behavior
- Schematic illustration of uncertainty growth in future projection of action plan.
  - Set S_0 corresponds to the initial known state of the actor. Sets S_1 and S_2 correspond to reachable sets at the end of actions (d_1, M_1) and (d_2, M_2), respectively.



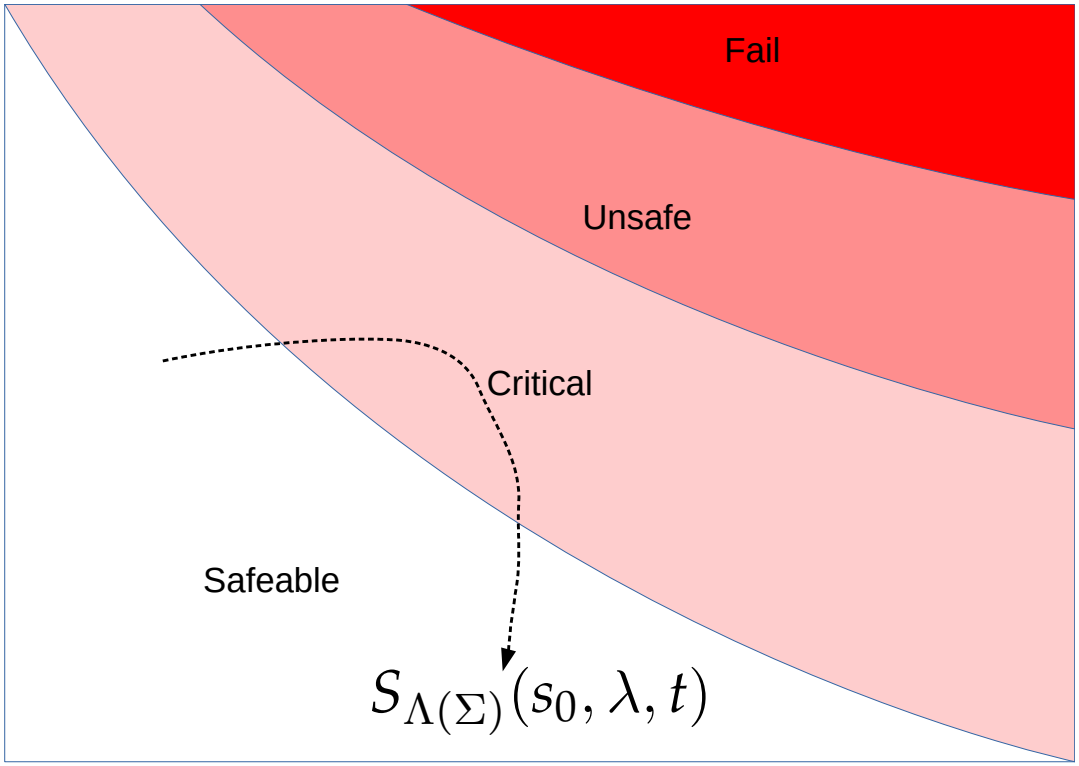
Main Idea

- Improve the safety of computer-controlled robotic machinery by using safety proofs as building material for runtime safety controller
  - Construct system component from safety proofs
  - Invoke formally verified safety calculations at runtime
  - Confer safety guarantees provided by proofs on primary controller
  - What are minimal requirements for easy translation?

Approach

- Work with a dictionary of action sequences
- Identify where we are in the state space according to the avaialability of safety-preserving action sequences (SPAS)
  - A (SPAS) is one that is *guaranteed* to preserve a safety property in *unbounded time*
- Use safety proofs to characterize state space:
  - *Safeable* A SPAS will be available one control action in the future, no matter what action we take at present
  - *Critical* At least one SPAS may be initiated in the present, but there may not be one available in the future
  - *Unsafe* No SPAS is available – some sequences may result in safe operation but outcome is uncertain
  - *Fail* No SPAS available – any action sequence results in guaranteed violation of safety property

State space



$S_{\Lambda(\Sigma)}(s_0, \lambda, t)$ is the system's state at time t during its operation. $\lambda \in \Lambda(\Sigma)$ is a realization from the probability space of non-deterministic trajectories available when the system satisfies the constraints of scenario Σ.

Safety Predicates

- A set of safety objectives $\{P_1(s, \Pi_1), P_2(s, \Pi_2), \ldots\}$ along with a corresponding set of safety predicates for each, $\{\Psi_1(s_0, \Sigma, \Pi_1), \Psi_2(s_0, \Sigma, \Pi_2), \ldots\}$
  - For objective i, $\Psi_i(s_0, \Sigma, \Pi_i)$ evaluates safety from the present state s_0 into the future in unbounded time
  - Account for a range of future possibilities associated with a sequence of future actions and uncertain system dynamics
- Safety predicates Ψ_i are quantifier-free, ensure safety for scenario Σ
- Scenario Σ represents future uncertainty and action sequences for multiple actors

Proof Requirement: guaranteeing safety property in unbounded time

Theorem *In-model safety in unbounded time*

$$\forall s_0, \Sigma, \Pi_i, \Psi_i(s_0, \Sigma, \Pi_i) \rightarrow \\ \forall t, \lambda \in \Lambda(\Sigma), t > 0 \rightarrow P_i(S_{\Lambda(\Sigma)}(s_0, \lambda, t), \Pi_i) \wedge \\ \Psi_i(S_{\Lambda(\Sigma)}(s_0, \lambda, t), \Sigma \triangleright t, \Pi_i) \quad (1)$$

parameterized by initial state s_0, scenario Σ, and safety parameters Π_i.
This says each Ψ_i evaluates safety property for scenario Σ in unbounded time under model assumptions.

How to handle off-model dynamics

- What happens when the system dynamics violate our proof assumptions?
  - We can call this off-model dynamics
- Safety properties may not hold
- Safety controller must not "give up"
- Still give safest advice it can under the circumstances
- We need to quantify a spectrum of weaker safety guarantees when the strong property fails

Proof Requirement: Ranking action sequences for safety

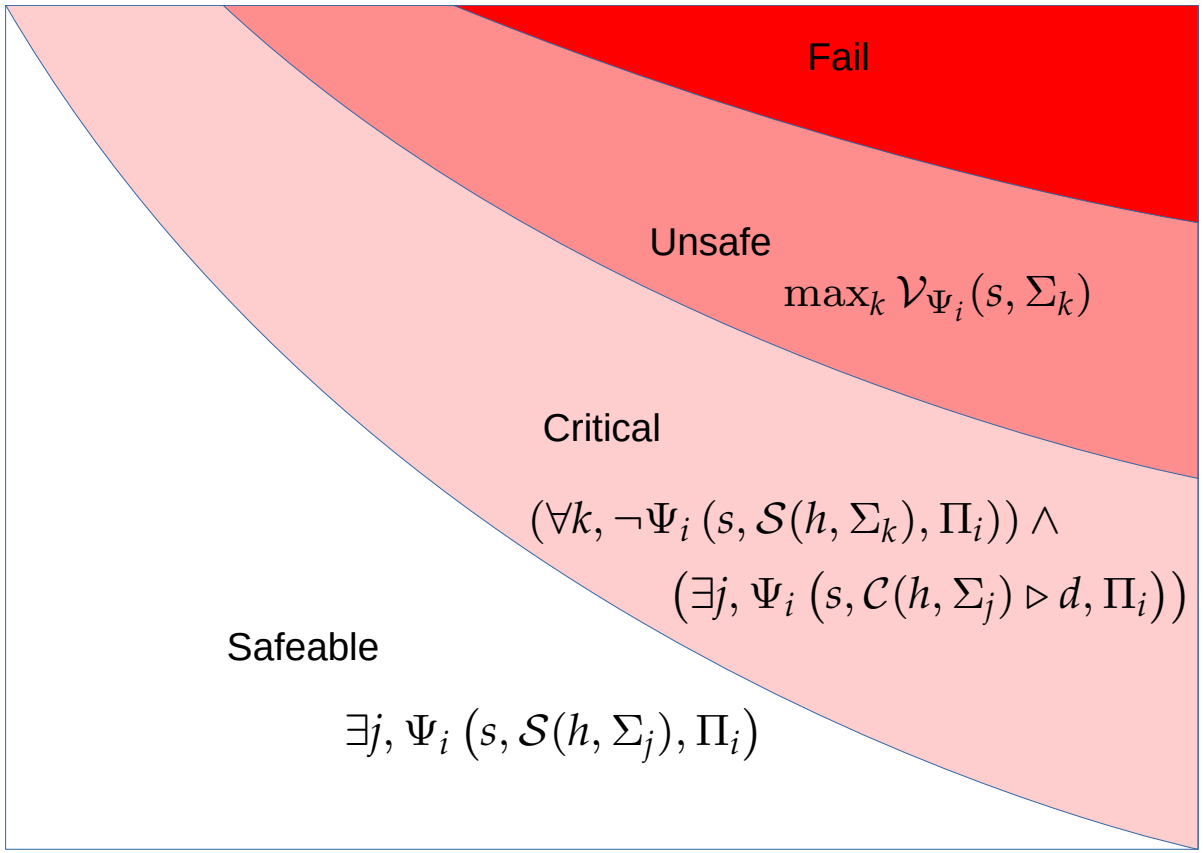
To handle off-model – unplanned, unmodeled – system dynamics, we define the set of safe parameter possibilities $\mathcal{R}_{\Psi_i}(s, \Sigma) = \{\alpha \,|\Psi_i(s, \Sigma, \alpha)\}$, and require a monotonic function in parameter space $\mathcal{V}_{\Psi_i}(s, \Sigma)$.

Theorem *Off-model monotonic safety ranking*

$$\forall\ s, \Sigma, \Xi,\ \mathcal{V}_{\Psi_i}(s, \Sigma) < \mathcal{V}_{\Psi_i}(s, \Xi) \rightarrow \mathcal{R}_{\Psi_i}(s, \Sigma) \subset \mathcal{R}_{\Psi_i}(s, \Xi) \quad (2)$$

We can use $\mathcal{V}_{\Psi_i}(s, \Sigma)$ as a metric that helps us quantify an action plan's relative "closeness" to preserving the safety property.

State space


Fail
Unsafe
$\max_k \mathcal{V}_{\Psi_i}(s, \Sigma_k)$
Critical
$(\forall k, \neg \Psi_i\,(s, \mathcal{S}(h, \Sigma_k), \Pi_i)) \wedge$
$\left(\exists j, \Psi_i\,\left(s, \mathcal{C}(h, \Sigma_j) \triangleright d, \Pi_i\right)\right)$
Safeable
$\exists j, \Psi_i\,\left(s, \mathcal{S}(h, \Sigma_j), \Pi_i\right)$

Guarantees

- Each control action we check for safety-preserving action sequences available for future control actions
- When there is no safety-preserving action sequence in the future, one is guaranteed available in the present due to analysis from the previous timestep
- Safeable states cannot transition directly to unsafe states without first transitioning through a critical state (guaranteed by formal proof)

Outline

- Introduction
- Approach
- **Controller Synthesis**
- Implementation and Results
- Conclusions

Controller Synthesis

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

- We can synthesize a controller that is generally applicable to systems that have safety predicates as described
- The function body is invoked at each control action, with up-to-date state information and the action proposed by the primary controller at this moment

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj(s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj(s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj(s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

Compute safety predicates to identify safeable states: is there any safety-preserving action sequence that we could implement starting in the future?

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

} If so, allow primary controller to act without interference.

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

} Compute next action in sequence under a range of future decisions.

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

Compute safety predicates to identify critical states: which safety-preserving action sequence can we implement starting now? which is underway that we can continue?

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

} Compute ranking to evaluate safety characteristics of action sequences.

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

} Invoke action that implements or continues a safety-preserving action sequence.

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

}

If there is no available safety-preserving action sequence, which is most safe or maintains the most desirable combination safety properties (as determined by function U)?

CONTROLLER SYNTHESIS

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj (s, S(h, Σi), Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
             map(λj.Ψj (s, C(h, Σi ▷ e), Πj),1:n),
             1:k, d)
       m = map(λi.λe.
             map(λj.VΨj (s, C(h, Σi ▷ e)),1:n),
             1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

History tracks system action sequence that has been invoked but due to system lag not yet affected system dynamics (but will in the future)

Outline

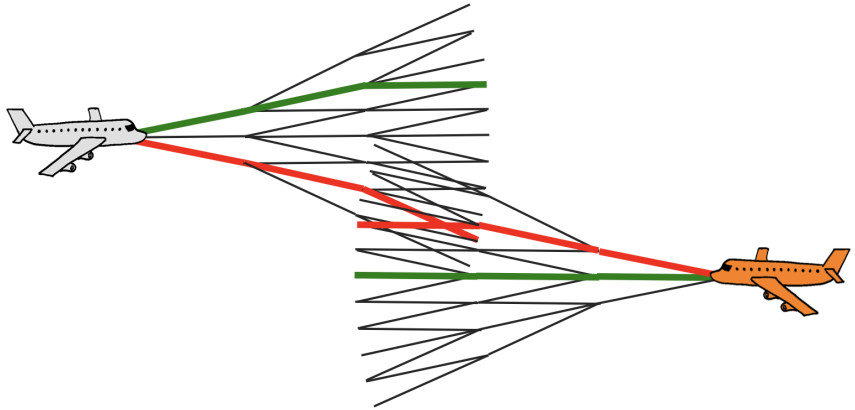
- Introduction
- Approach
- Controller Synthesis
- **Implementation and Results**
- Conclusions

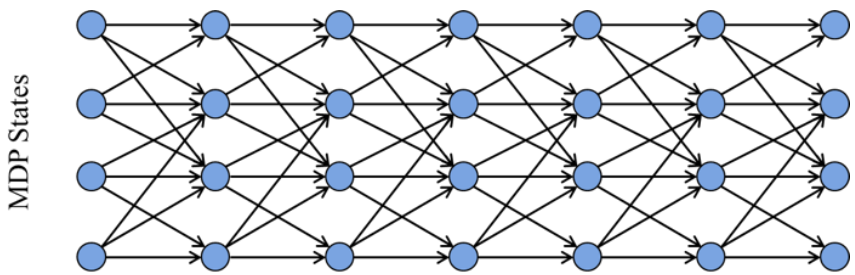
Airborne Collision Avoidance

- Industrial systems install on all airliners
  - Detects potential mid-air collisions
  - Since 1980s: TCAS
  - New system: ACAS-X
- Give vertical advisories, e.g. *Climb, Do not descend*
- System of last resort

ACAS-X Design

- Core of the system is a table driving behavior
  - Optimal policy for a Markov Decision Process (discrete dynamics)
  - Cost for each action at discrete cutpoints in the state space
  - Large size (36 MB compressed, 2GB uncompressed)
- Additional online costs computed to modify costs according to ad-hoc rules





Safety Analysis

We developed predicates for safety analysis of vertical collision avoidance maneuvers in ACAS X:

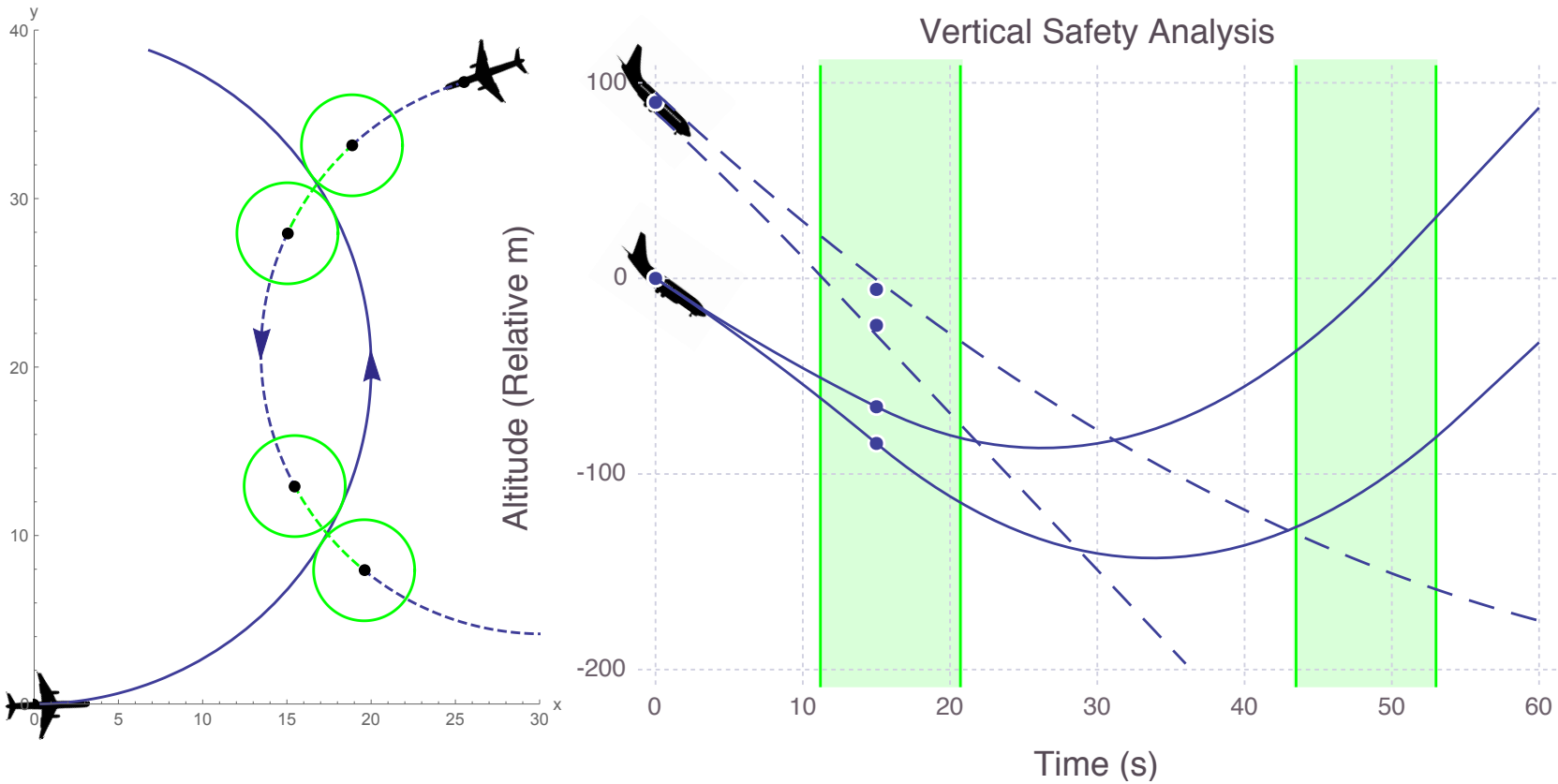
Kouskoulas, et al., *Formally Verified Safe Vertical Maneuvers for Non-Deterministic, Accelerating Aircraft Dynamics,* In Interactive Theorem Proving (ITP) , 2017.

- Predicates used to analyze an existing system's safety properties, not synthesize a controller

Safety Analysis

- Vertical predicates used for safety analysis of ACAS X
  - Use an independent model for for analyzing safety in an existing system
  - Developed using formal methods (FM), rigorous mathematical approaches to modeling and proving properties about a software system
  - Use hybrid system model combining discrete, pilot decisions and sequences of discrete actions with continuous dynamics
  - Use higher-order logic (HOL) and calculus of inductive constructions (CIC) to state safety conjectures in continuous time (Coq theorem prover)
  - With minor changes, predicates satisfy requirements for provably correct controller synthesis described in our approach.

Example Vertical Safety Analysis



Vertical Safety Predicate

$$\begin{aligned}\Gamma((A,B,C),t_b,t_e) \equiv t_b \leq t_e \rightarrow \\ (A>0 \wedge ((0 \leq D \wedge (R_1 > t_e \vee R_2 < t_b)) \vee D < 0) \vee \\ A<0 \wedge (0<D \wedge R_2 < t_b \wedge R_1 > t_e) \vee \\ A=0 \wedge (B>0 \wedge -C/B < t_b \vee B<0 \wedge -C/B > t_e \vee B=0 \wedge C>0))\end{aligned} \tag{1}$$

$$\begin{aligned}\Phi(Q_1,L_1,t_{t1},Q_2,L_2,t_{t2},t_b,t_e) \equiv \\ \Gamma(Q_1-Q_2-P,\max(t_b,0),\min(t_e,t_{t1},t_{t2})) \wedge \Gamma(L_1-L_2-P,\max(t_b,t_{t1},t_{t2}),t_e) \wedge \\ (t_{t1}>t_{t2} \rightarrow \Gamma(Q_1-L_2-P,\max(t_b,\min(t_{t1},t_{t2})),\min(t_e,\max(t_{t1},t_{t2})))) \wedge \\ (t_{t1}<t_{t2} \rightarrow \Gamma(L_1-Q_2-P,\max(t_b,\min(t_{t1},t_{t2})),\min(t_e,\max(t_{t1},t_{t2}))))\end{aligned} \tag{2}$$

$$\begin{aligned}\Psi = \bigwedge_{j\in\{1,\ldots,n\}} \Bigg(\Bigg(\bigwedge_{i\in\{1,\ldots,m\}} \Phi(Q_i^{\text{Own}\downarrow}, L_i^{\text{Own}\downarrow}, t_{ri}^{\text{Own}\downarrow}, Q_i^{\text{Int}\uparrow}, L_i^{\text{Int}\uparrow}, t_{ri}^{\text{Int}\uparrow}, \tau_{ij}, \upsilon_{ij}) \Bigg) \vee \\ \Bigg(\bigwedge_{i\in\{1,\ldots,m\}} \Phi(Q_i^{\text{Int}\downarrow}, L_i^{\text{Int}\downarrow}, t_{ri}^{\text{Int}\downarrow}, Q_i^{\text{Own}\uparrow}, L_i^{\text{Own}\uparrow}, t_{ri}^{\text{Own}\uparrow}, \tau_{ij}, \upsilon_{ij}) \Bigg) \Bigg)\end{aligned} \tag{3}$$

Results

- Comprehensive safety proof is not possible for ACAS X under our model assumptions
- Disproved global safety by generating counterexamples
  - A counterexample is a point in the state space where guaranteed safe advice is possible, but the system does not issue it
- ACAS X trades off safety under our acceleration assumptions for particular operational behavior
  - Excessive alerting may lead to unresponsive pilots

Safety Controller Synthesis Case Study

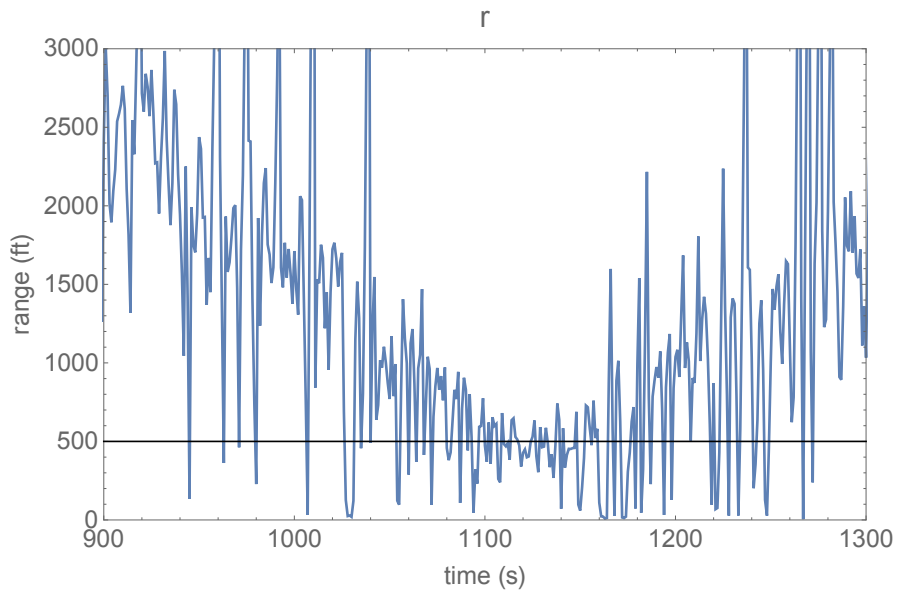
- It may be that dangerous collisions that need correction only exist in a fraction of a percent of the state space – critical area is important to handle correctly
- Evaluate ACAS X system performance improvement with additional safety controller
  - Create pathologically dangerous encounters, chosen specifically in critical parts of the state space where ACAS X gives advice that allows a collision under model assumptions
    - ACAS X could issue advice that definitively avoid collisions under the model assumptions, but choses alternative advice
  - Modified proofs and predicates used for safety analysis to fit the framework described in this paper
    - Formal, mathematical guarantees of correctness
  - Synthesized controlled using safety proofs

In-model Performance

- In simulation, the safety controller intervened and eliminated all mid-air collisions resulting in fully safe operation under our dynamics assumptions

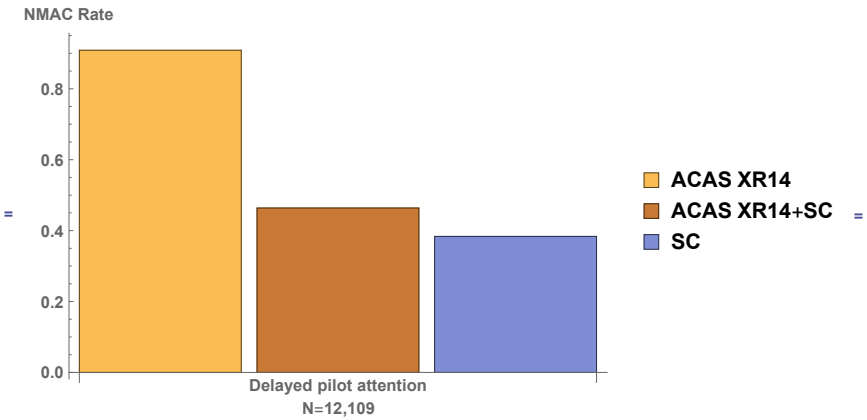
Off-model Performance

- What happens to controller performance when the dynamics model is violated?
  - Mathematical analysis for a CPS relies upon model
- Introduce unharacterized, off-model noise

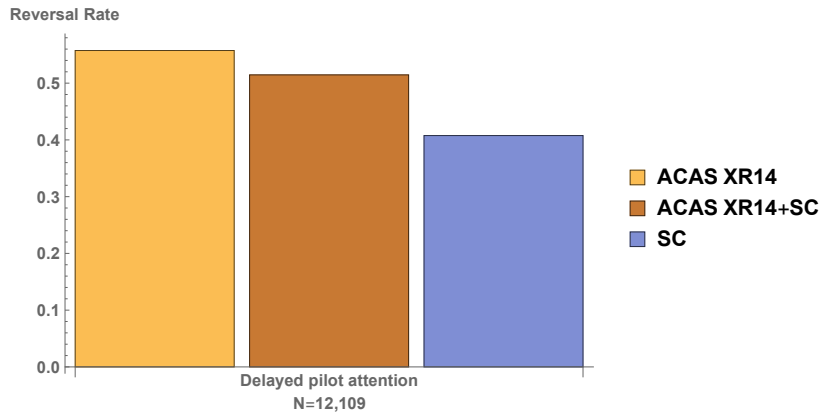


Horizontal range during a two-aircraft encounter output by the ACAS X Run14 surveillance and tracking module over time. True range would be a smooth curve.

Off-model Performance



Safety performance on a set of stressing encounters passing through from critical states comparing ACAS X to various configurations of the safety controller (SC).



Advisory reversals on a set of stressing encounters passing through critical states comparing ACAS X to various configurations of the safety controller (SC).

Outline

- Introduction
- Approach
- Controller Synthesis
- Implementation and Results
- **Conclusions**

Conclusions for ACAS X case study

- Approach produces a practical safety controller for ACAS X that shows good performance when handling uncharacterized noise
- It does not adversely affect ACAS X in other parts of the state space
- Encounters sampled by regular testing of ACAS X do not contain a statistically significant representation of state-space critical conditions

General Conclusions

- Promising approach to producing safety controller for complex behavior in CPS systems
  - Works with a finite dictionary of safety-preserving action sequences
  - Allow primary controller freedom to pursue objectives until safety cannot be guaranteed
  - Safety-preserving action sequence always available under model assumptions
  - Control actions during off-model system evolution chosen to maximize safety insofar as possible, given future uncertainty
  - Independent of primary controller algorithm, so safety controller doesn't depend on its behavior