Verification of Autonomous Systems*

Aaron Dutle, J Tanner Slagel, John Siratt
NASA Langley Research Center

* Components

What this work entails

Formal Verification: The use of mathematically rigorous tools and techniques to verify the correctness of a digital design.

Two main products of this enterprise:

1. Verification of the correctness of an algorithm, piece of software, architecture, operational concept, etc.
2. Development or advancement of a tool or technique that can be used in verification of an algorithm, etc.

Verification of Autonomous Path Planning	Verification of RTA Frameworks	Numerical verification of AI/ML components
Type 1: Verification performed on three distinct algorithms: Route restructuring, simple in-flight recapture planning, and Bellman-Ford verification.	Type 1: Verification of simplex RTA framework with instantiations Type 2: Development of Plaidypvs, for formalized reasoning of hybrid systems.	Type 1: Verified activation functions, concrete example of numerical worst-case. Type 2: Enhancement of PRECiSA for NN applications, NN transformation technique.

Formal Verification of Autonomous Path Planning

Aaron Dutle[1], Esther Conrad[1], Jai Aslam[2], Paolo Masci[3], Andrew Peters[1]

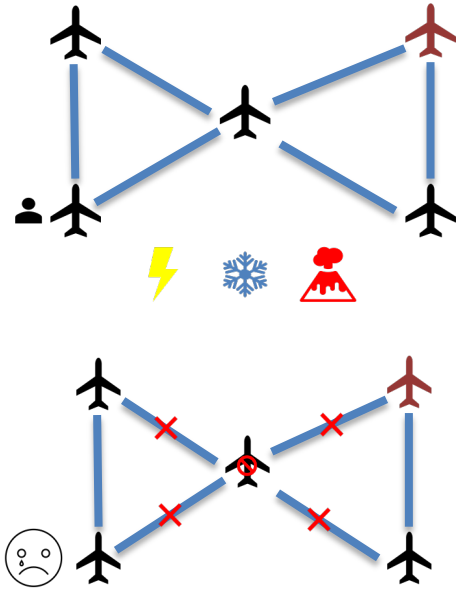
[1] NASA Langley Research Center

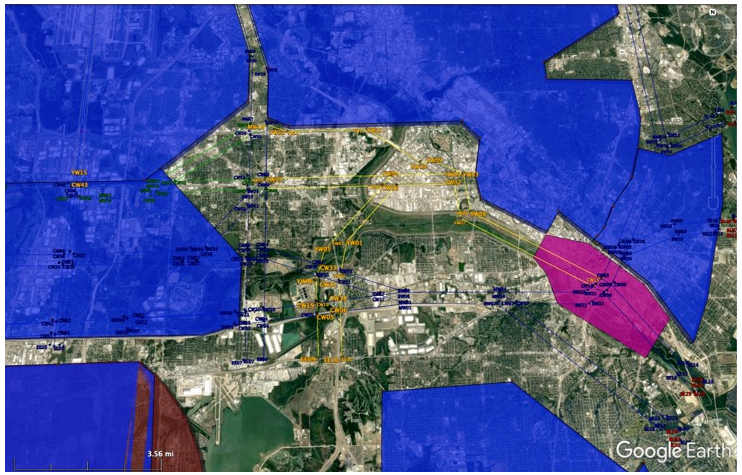
[2] Analytical Mechanics Associates

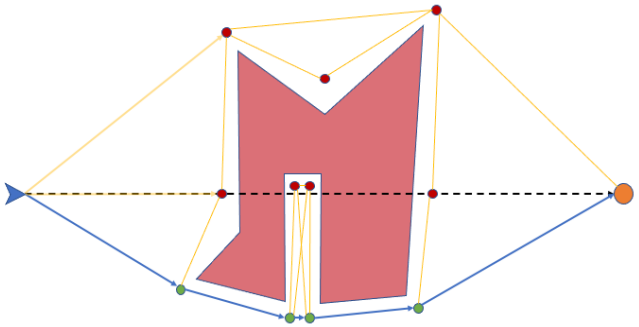
[3] NASA Langley OSTEM Intern

Path Planning Applications

Reconfiguration of route structures due to weather, mechanical, or other issues.

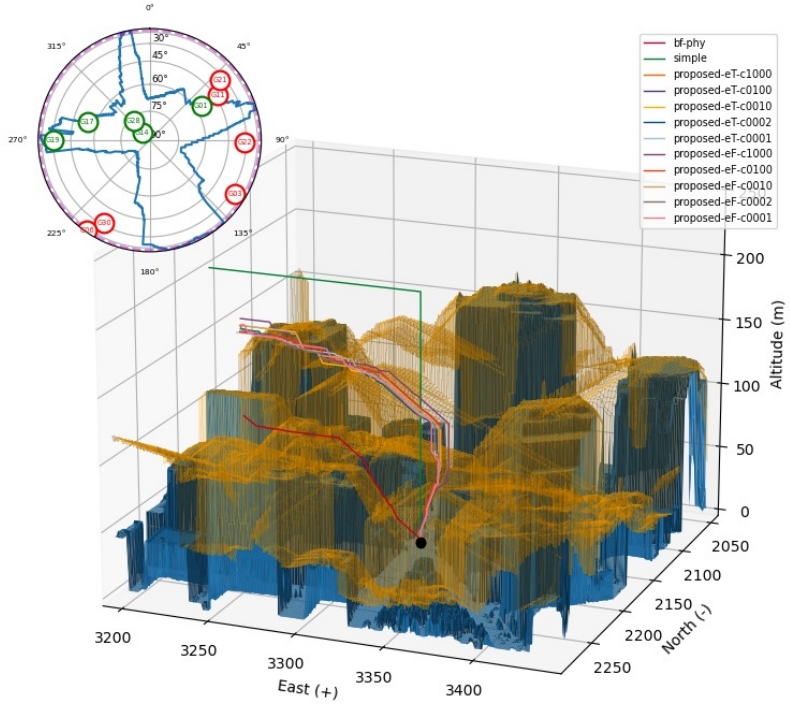




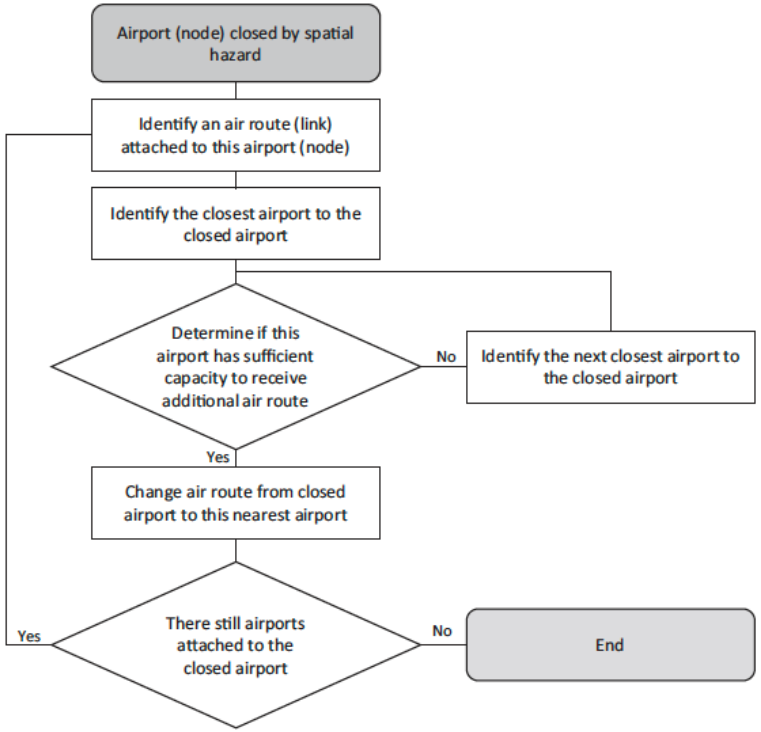


Replanning for in-flight autonomous route recapture, avoiding geofenced areas, as done in ICAROUS.

NavQ performs pre-flight planning to maximize forecast GNSS availability to support autonomous navigation.



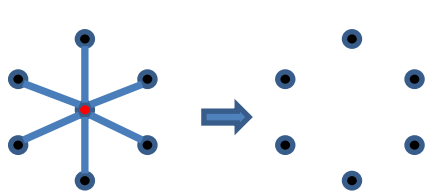
Verification of an adaptive airspace rerouting algorithm

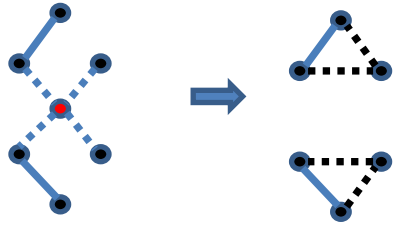


An adaptive airspace restructuring algorithm from:

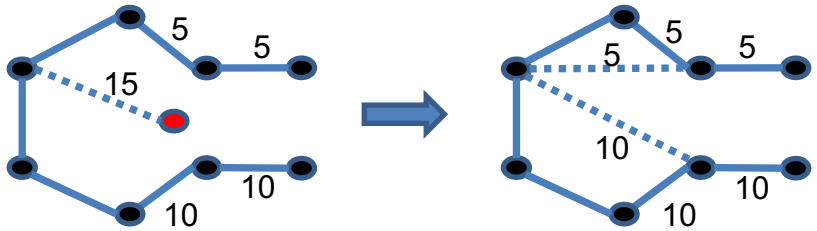
Dunn, Sarah, and Sean M. Wilkinson. "Increasing the resilience of air traffic networks using a network graph theory approach." *Transportation Research Part E: Logistics and Transportation Review* 90 (2016): 39-50.

- Performed a *Level-1* formal analysis of the algorithm—fully specified the system in PVS, and discovered some issues:
  - Self-loops or duplicate edges can be created by the algorithm.
  - The algorithm can fail if a ”hub” goes down.
  - The route network can become disconnected.

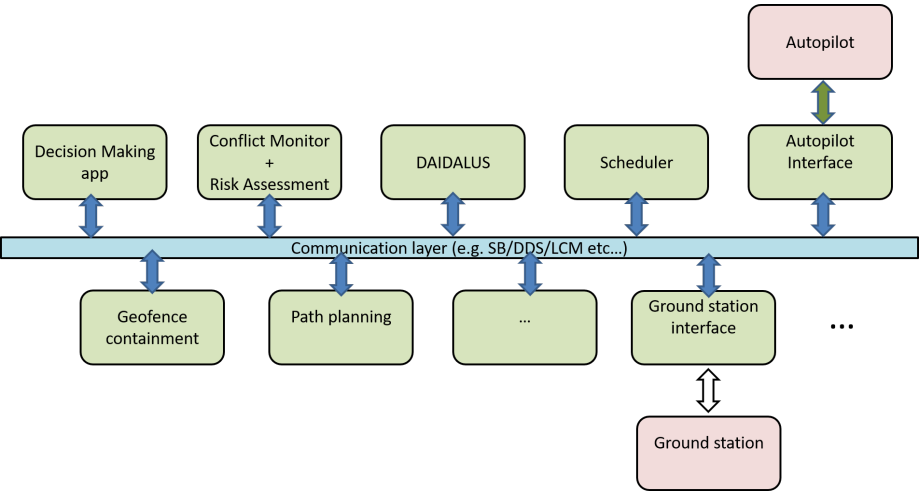




- Created and specified a more flexible generalization of the algorithm.
  - Use weights on route edges to signify the number of flights.
  - Allows for re-routing to multiple destinations.



Design and verification of a simple path-planning algorithm

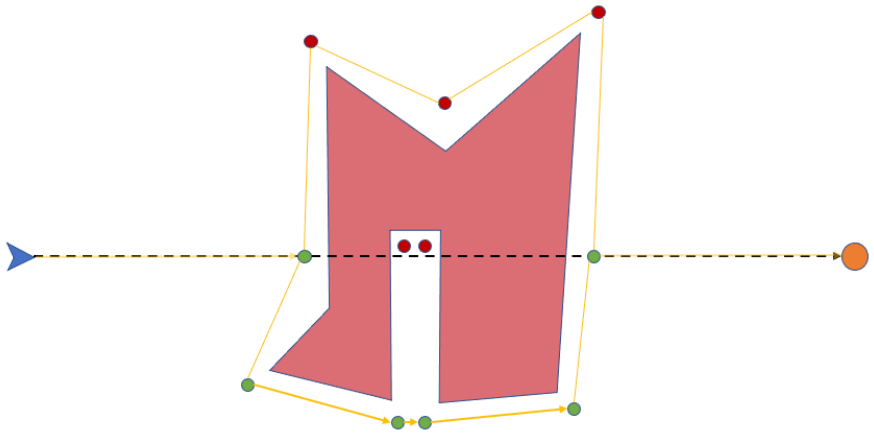


ICAROUS path planning

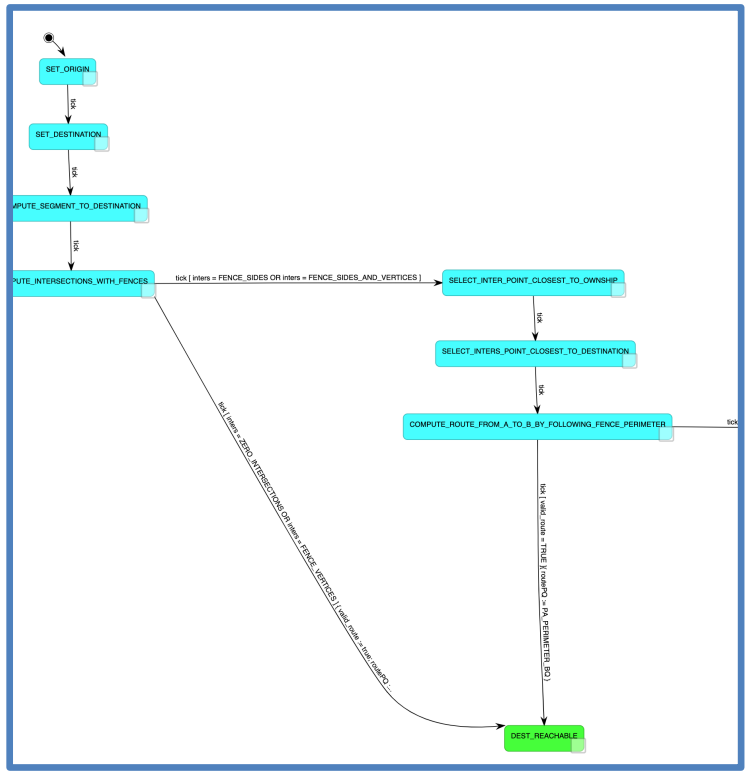
- ICAROUS is an integrated, extensible system for the uncrewed operation of air vehicles. It employs and connects different modules to provide information and feedback to the pilot (human or automated).
- Functions include conflict avoidance, geofence awareness, sensor integration, decision-making, **path planning**, and others.
- Path planning is used when the current planned route has a predicted conflict with a geofence or aircraft, generally after a conflict avoidance maneuver that makes the original plan no longer usable.
- The previous existing path planning algorithm in ICAROUS is based on the A* algorithm.
  - The algorithm was developed and tested rapidly, without formal verification, and with a certain class of vehicle assumptions.
  - Some undesirable behavior was discovered in simulation of UAM scenarios (repeated replanning, excessive stand-off from geofences.)
  - A simpler alternate algorithm was proposed, formally specified, and is being integrated.

Design and verification of a simple path-planning algorithm

- If the current segment intersects a polygon, add the entry and exit points to the polygon.
- Fly to the entry point and follow the outline of the polygon to the exit point.
  - Standoff distance from the polygon is assumed.
  - Assumes that the waypoint is reachable (returns “unreachable” if not).
  - Computes both paths around the polygon and chooses the shorter.
  - Cuts off some non-convexities (when a very tight corridor/turn exists).



The waypoints in green are added to the flight plan based on the algorithm.

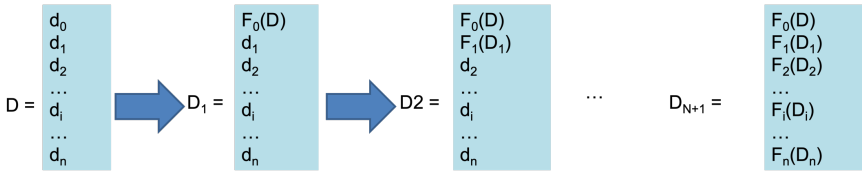


- Created a formal model from the pseudocode algorithm, as a state machine emucharts diagram.
- Model has been translated (automatically) into:
  - PVS model (Interactive theorem prover)
  - NuSMV model (model checker)
  - C code (integration into ICAROUS)

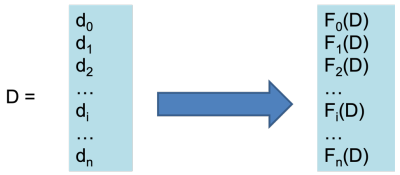
Verification of Bellman-Ford and applications

Bellman-Ford is a classic algorithm in graph theory for finding the smallest weight paths.

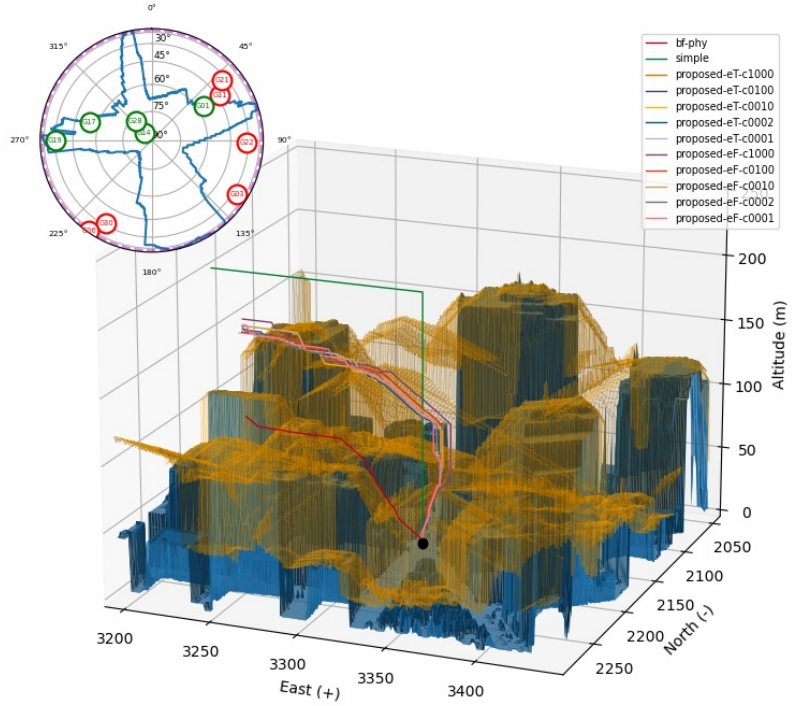
- Uses a specified *source* or *sink* vertex and finds smallest weight paths *from* or *to* every other location in the graph.
- The design allows for an *embarrassingly parallel* implementation, by performing updates at distinct vertices simultaneously.
- A parallel and a sequential version of the general algorithm were specified in PVS, and several correctness properties were proven about each.



The sequential form of the algorithm only requires keeping one copy of the final data, but every calculation must be performed in sequence.



The parallel form of the algorithm can perform multiple calculations simultaneously, but two copies of the data must be maintained between iterations.



NavQ uses a modified version of the Bellman-Ford algorithm to find a paths maximizing navigation quality in urban areas.

NavQ adaptation of Bellman-Ford

- The NavQ adaptation of Bellman-Ford minimizes a measure of *risk* based on the number of GNSS satellites visible at a given time.
- To account for the movement of satellites, the algorithm incorporates changes in risk over time.
- Although significant testing was done, formal analysis was desired to ensure proper behavior.
- Formal specification and verification efforts found a bug in the algorithm description*, and an example of non-optimality of the result of application.

In the NavQ adaptation, risk in one time epoch can change in the next time epoch.

Algorithm 3 Relax Edges.

input
arrays (3D volumes)
$step_c$ (in meters)
$step_n$ (in meters)

```
1:  procedure RELAX EDGES
2:      for All voxels i in the volume, do
3:          for each neighbor n of i, do
4:              ddf ← ||n, i||
5:              step_t ← ad[n] + ddf
6:              if step_c <= step_t < step_n then
7:                  if (ar[i] > ar[n] + risk[i] × ddf) then
8:                      pn[i] ← n
9:                      ar[i] ← ar[n] + risk[i] × ddf)
10:                     ad[i] ← step_t
11:                 else
12:                     if (ar[i] == ar[n] + risk[i] × ddf)&(ad[i] > step_t) then
13:                         pn[i] ← n
14:                         ar[i] ← ar[n] + risk[i] × ddf)
15:                         ad[i] ← step_t
16:                     end if
17:                 end if
18:             end if
19:         end for
20:     end for
21: end procedure
```

A portion of the NavQ specification. The NavQ implementation differs from the standard Bellman-Ford algorithm enough to warrant a formal analysis.

* The implemented code was correct, the algorithm description was not.

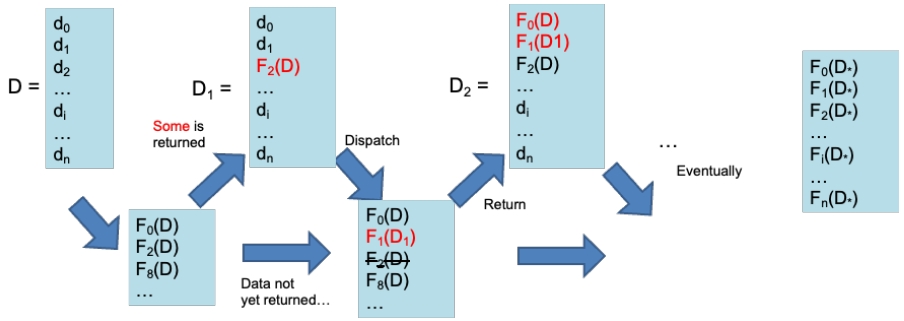
Results and future of path planning verification

Collaborations:

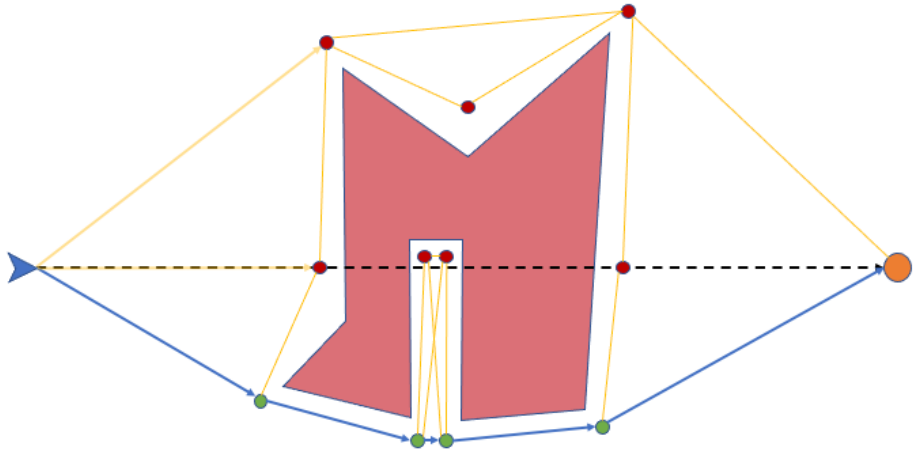
- IASMS path planning development within SWS.
  - Paths that mitigate multiple disparate risks.
  - Avoid non-feasible paths (performance limitations of aircraft).
- Inter-agency agreement (effective 4/2/2025) with Air Force Research Lab to work on further verified path-planning.
- ICAROUS team, integrate BF-based route recapture.

Future work:

- Verify properties of a parallel implementation that keeps only one copy of data (general parallel version).
- Investigate other path planning methods.
  - Based on alternate graph algorithms.
  - Lattice planning.



A generic parallel Bellman-Ford algorithm can hold one set of data but still be run as a parallel computation.



Route recapture using Bellman-Ford.

Presentations:
Invited presentation at the Joint Mathematics Meetings, Jan 2024. *Formal Verification, Distributed Computing and Path Planning Algorithms.*

55th conference on Combinatorics, Graph Theory and Computing, March 2024. *Formalization of the Bellman-Ford Algorithm for Airspace Applications.* Paper submitted to post-proceedings.

Formal Verification of Runtime Assurance

J Tanner Slagel[1], Lauren White[1], Mariano Moscato[2], César Muñoz[1], Nicolas Crespo[3], Aaron Dutle[1]

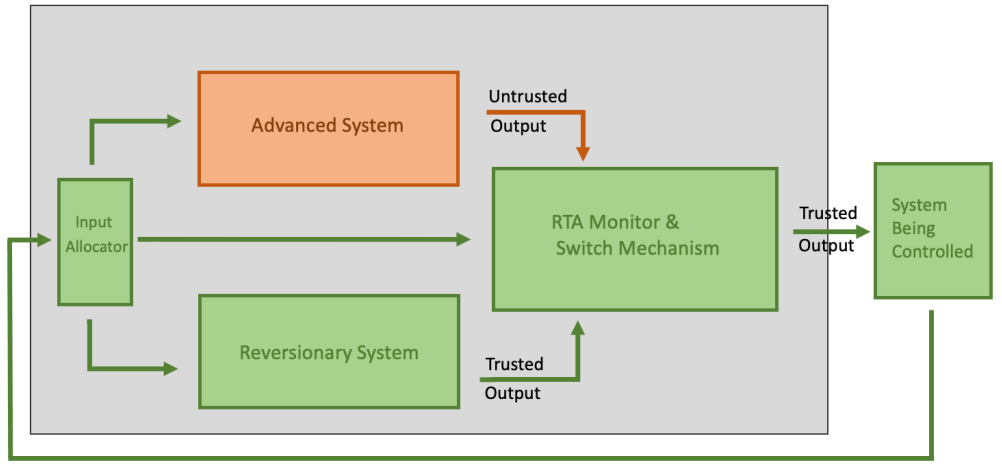
[1] NASA Langley Research Center

[2] Analytical Mechanics Associates

[3] NASA Langley OSTEM Intern

Overview

- Runtime monitoring is a well-studied area of formal methods, founded on the idea that a system that cannot be verified to *always possess* some property can often more easily be *monitored* as to whether the property is maintained.
- Runtime assurance assesses a system that employs runtime monitoring and provides some reasoning that the system under assessment can be considered operating properly.
- Often, a runtime monitor is used in a *simplex* architecture, where an untrusted component is allowed to operate, but is monitored. If the monitor is triggered, a trusted component takes over operation.
- The goal of this milestone is to provide formal verification of example simplex architecture.



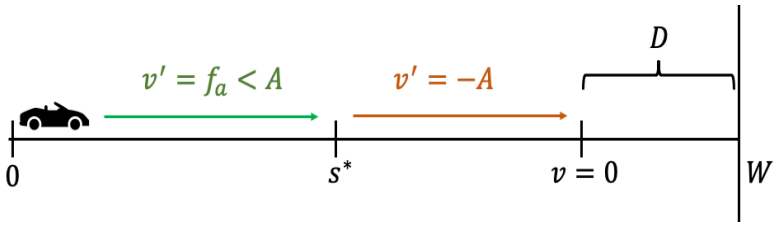
Basic structure of a simplex architecture system

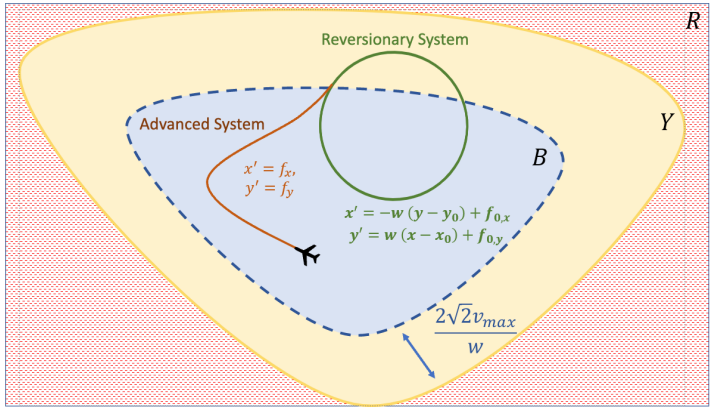
Approach

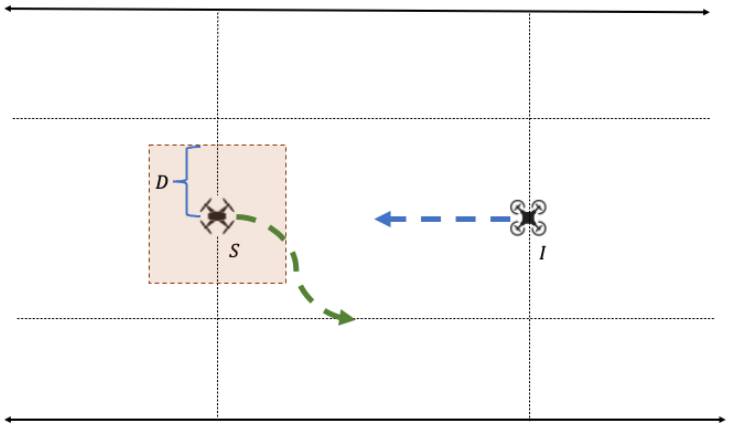
Develop a framework that many simplex RTA systems will fit, and verify parametric properties about the framework.

Instantiate the framework with example systems of interest, and properties specific to each.

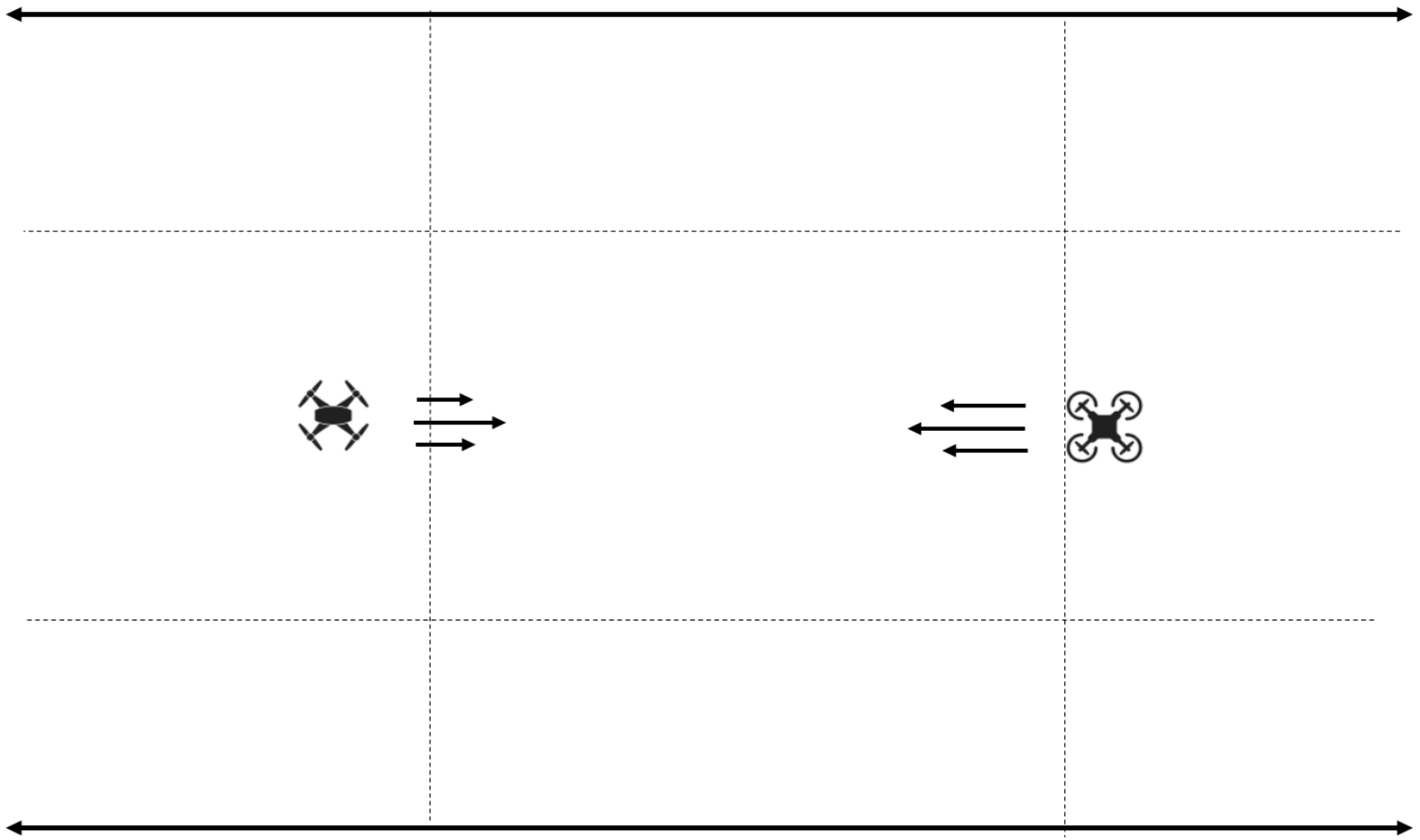
1D *autobraking* fallback.	Geofenced operations with a *return to safe region* fallback.	Productive conflict avoidance







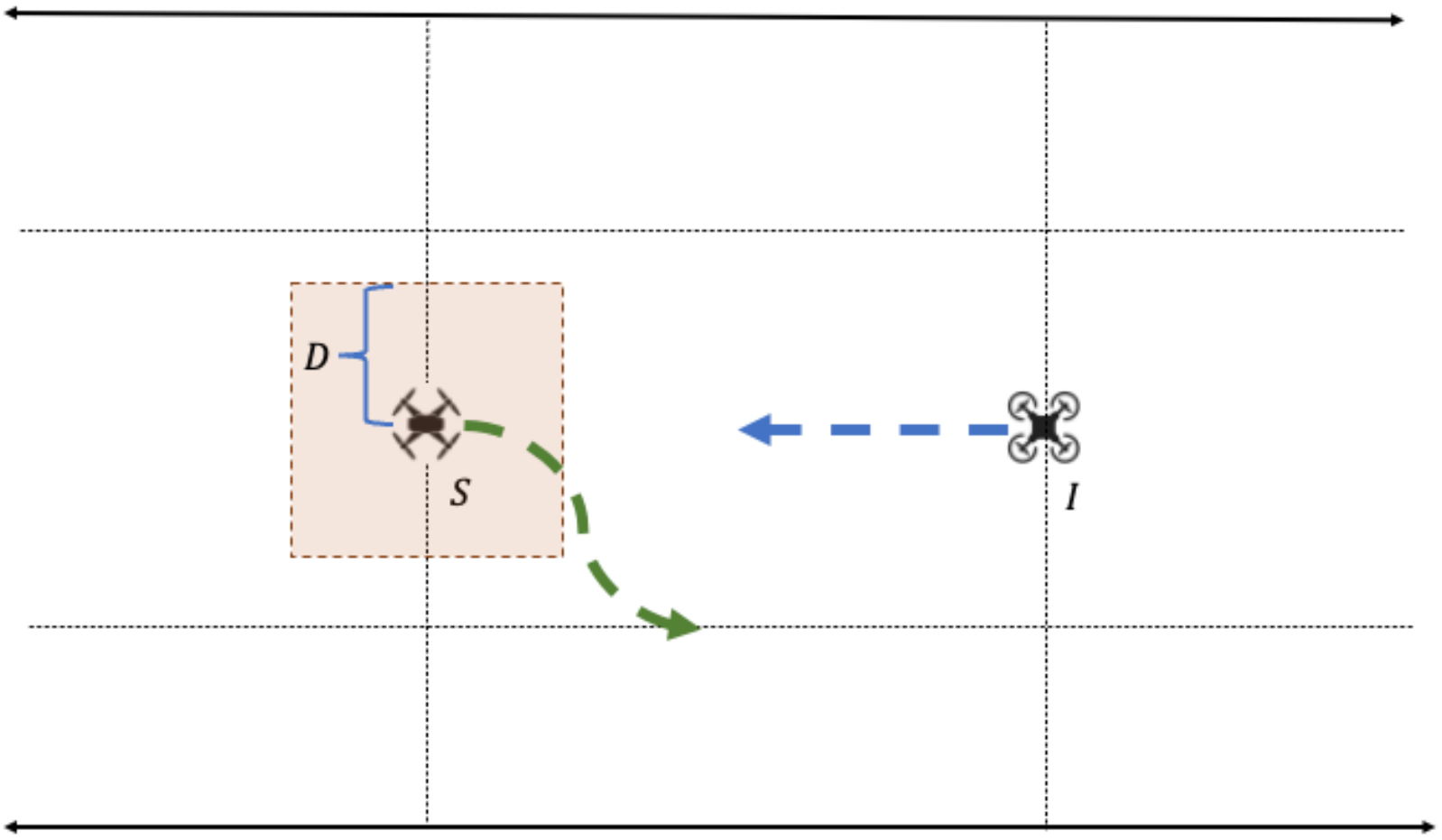
Productive conflict avoidance



Assumptions:

- max velocity V_{max}
- max acceleration A
- sample rate τ

Productive conflict avoidance



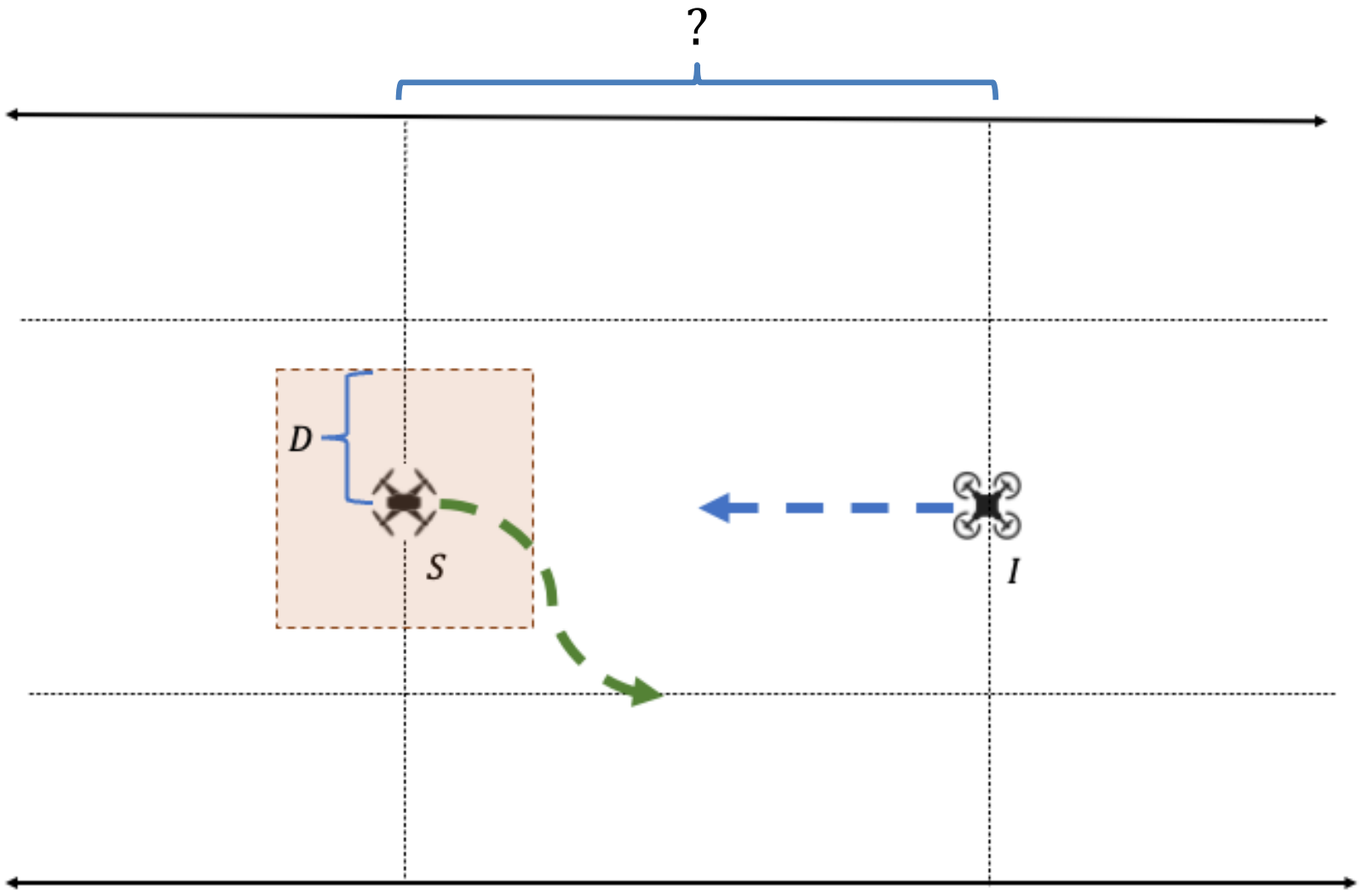
Assumptions:

- max velocity V_{max}
- max acceleration A
- sample rate τ

Safety Requirement:

Distance between I and S is at least D in both coordinate directions.

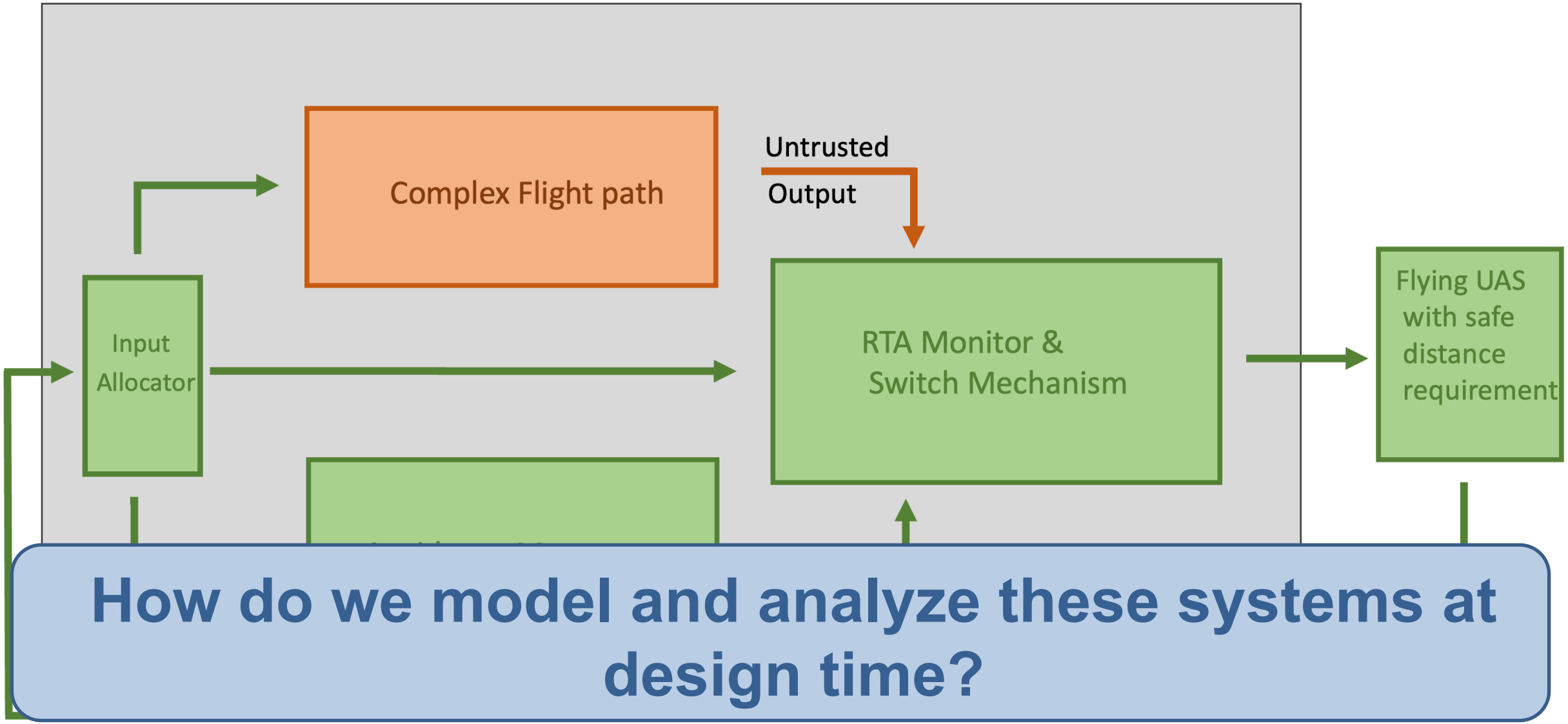
Productive conflict avoidance



Design Requirement:

- What distance apart does the avoidance maneuver need to satisfy safety requirement?
- What does the sample rate τ need to be?

Productive conflict avoidance



How do we model and analyze these systems at design time?

Modeling Concept

- Model the systems as Hybrid Programs (HPs)
- Prove properties using differential dynamic logic (dL)
- Most HP and dL systems do not support the approach taken for this work.
  - Need support for proving properties of arbitrary HP components to model the black-box untrusted controller.
  - Need support for instantiation of generic HPs with particular instances, to use the proof of safety for the framework in verifying properties of the example systems.
  - Used the NASA-developed **Plaidypvs** tool (an embedding of dL in the PVS theorem prover) because of these constraints.

Modeling Concept

- **dL:** Differential Dynamic Logic for hybrid programs
- **PVS:** Interactive theorem prover

Result: Plaidypvs

- Formally verified soundness of **dL**
- Fully operational in **PVS**
- Leveraging features of **PVS** to extend **dL**

Hybrid Programs

Hybrid programs allow **formal specification** of hybrid systems:

- Discrete jump set:

$$(x_1 := \theta_1, \ldots, x_n := \theta_n)$$

- Differential equations:

$$\{x_1' := \theta_1, \ldots, x_n' := \theta_n \;\&\; \chi\}$$

  - $\{x_i\}_{i=1}^n$ variables
  - $\{\theta_i\}_{i=1}^n$ assignments (e.g. – functions of existing variable values)
  - χ first order formula that describes domain

Example:

$$(x := 0\,, y := c\,); \{x' = y, y' = -x \;\&\; y \geq 0\}$$

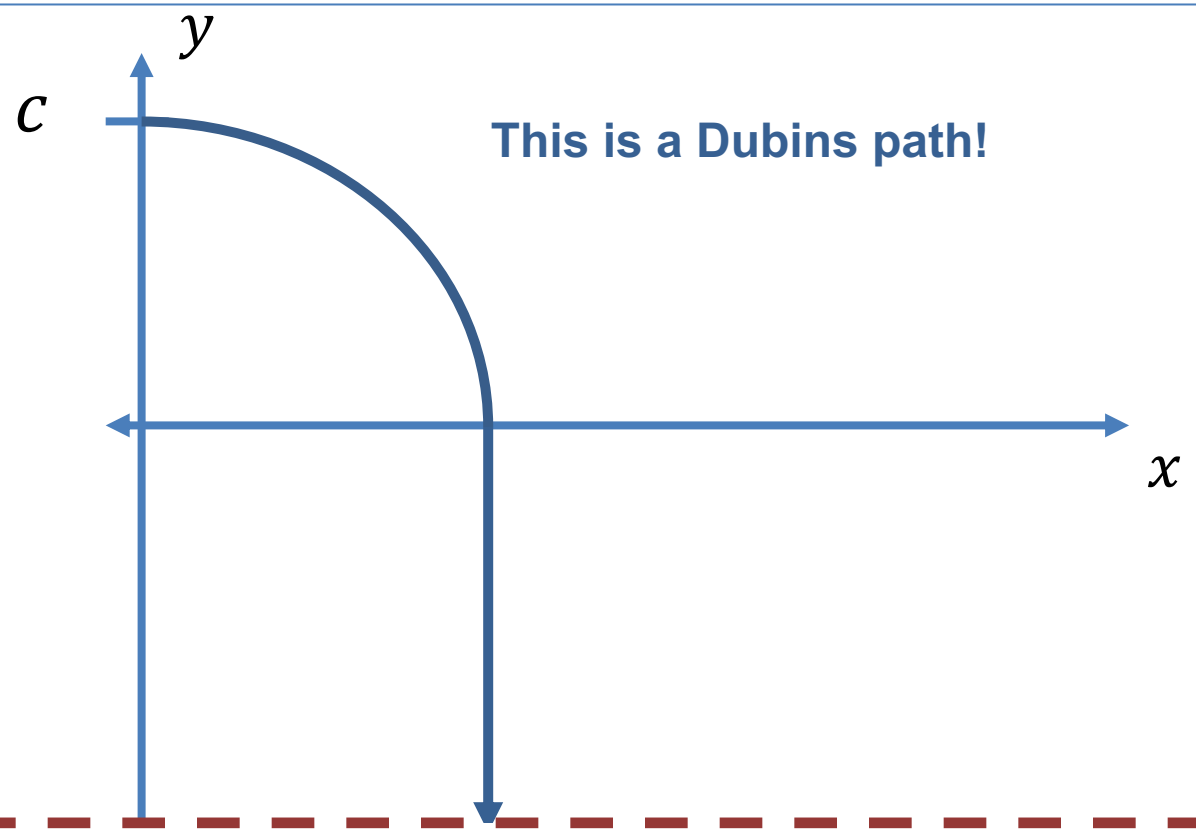
Hybrid Programs

For hybrid programs Hp_1, Hp_2, first-order formula χ:

- Choice $(Hp_1 \cup Hp_2)$
- Sequence $(Hp_1; Hp_2)$
- Repeat $(Hp_1)^*$
- Test $(?\chi)$

Example: $\left\{ \begin{array}{c} \left((?y > 0); \{x' = y, y' = -x \ \& \ y \geq 0\}\right) \\ \cup \\ \left((?y \leq 0); \{y' = -c\}\right) \end{array} \right\}^*$

Hybrid Programs (continued)



This is a Dubins path!

Example: $\left\{ \begin{array}{c} \left((?\, y > 0); \{x' = y, y' = -x \ \& \ y \geq 0\} \right) \\ \cup \\ \left((?\, y \leq 0); \{y' = -c \ \} \right) \end{array} \right\}^*$

dL: Differential Dynamic Logic

dL allows **formal reasoning** of hybrid programs:

- For hybrid program Hp and predicate P
  - All runs $[Hp]\mathrm{P}$
  - Some runs $\langle Hp\rangle\mathrm{P}$

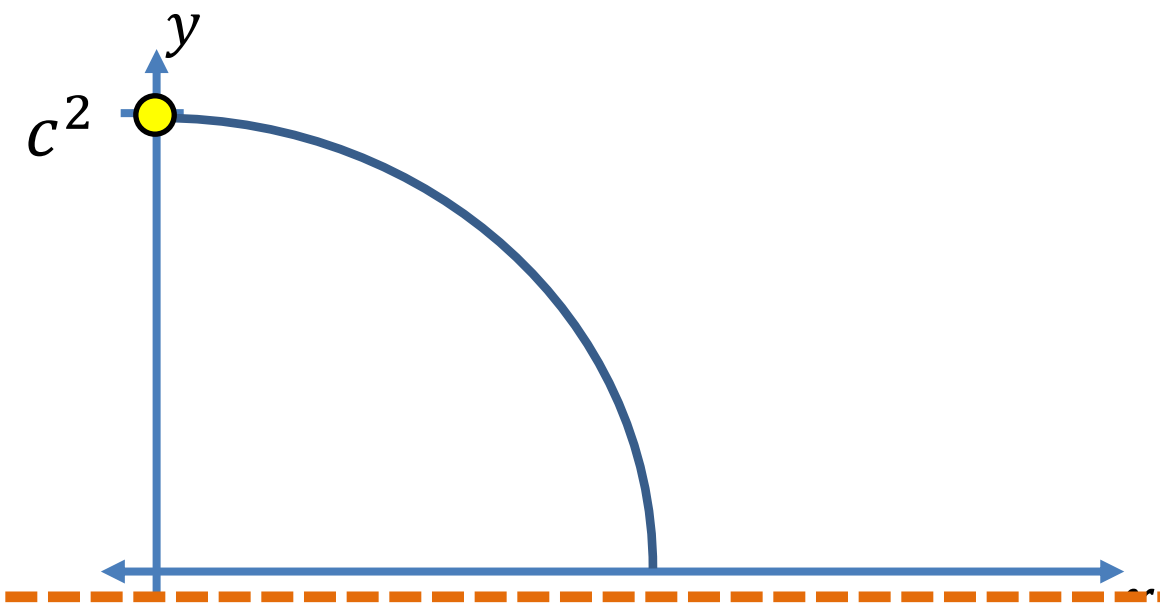
Example: Let

$$Hp \equiv \left((?\,y > 0); \{x' = y, y' = -x \;\&\; y \geq 0\}\right)$$

$$\mathrm{P} = (x^2 + y^2 = c^2),$$

then

$$\mathrm{y = c\,, x = 0} \rightarrow [Hp]P \qquad \mathrm{y = c\,, x = 0} \rightarrow \langle Hp\rangle(\mathrm{y = 0})$$



Example: Let

$$Hp \equiv \left((?\, y > 0); \{x' = y, y' = -x \;\&\; y \geq 0\}\right)$$

$$\mathrm{P} = (x^2 + y^2 = c^2),$$

then

$$\mathrm{y} = \mathrm{c}\,, \mathrm{x} = 0 \rightarrow [Hp]P \qquad \mathrm{y} = \mathrm{c}\,, \mathrm{x} = 0 \rightarrow \langle Hp \rangle (\mathrm{y} = 0)$$

dL: Differential Dynamic Logic – Rule Schema

Union axiom:

$$\frac{[Hp_1]P \wedge [Hp_2]P}{[Hp_1 \cup Hp_2]P}$$

Loop rule:

$$\frac{\Gamma \vdash J \qquad J \vdash [\alpha]J \qquad J \vdash P}{\Gamma \vdash [\alpha^*]P}$$

Differential invariant rule:

$$\frac{\Gamma, q(x) \vdash p(x) \qquad q(x) \vdash [x' := f(x)](p(x))'}{\Gamma \vdash [x' = f(x) \,\&\, q(x)]p(x)}$$

….and many more!

dL: Proof

Differential invariant rule:

$$\frac{\Gamma, q(x) \vdash p(x) \qquad q(x) \vdash [x' := f(x)](p(x))'}{\Gamma \vdash [x' = f(x) \,\&\, q(x)]p(x)}$$

$$y = c\,, x = 0 \;\vdash [\{x' = y, y' = -x\}](x^2 + y^2 = c^2)$$

Differential invariant rule:

$$\frac{\Gamma, q(x) \vdash p(x) \qquad q(x) \vdash [x' := f(x)](p(x))'}{\Gamma \vdash [x' = f(x) \,\&\, q(x)]p(x)}$$

$$y = c\,, x = 0 \;\vdash [\{x' = y, y' = -x\}](x^2 + y^2 = c^2)$$

dL: Proof

Differential invariant rule:

$$\frac{\Gamma, q(x) \vdash p(x) \qquad q(x) \vdash [x' := f(x)](p(x))'}{\Gamma \vdash [x' = f(x) \,\&\, q(x)]p(x)}$$

$$\frac{y = c, x = 0 \vdash x^2 + y^2 = c^2 \qquad \vdash [x' := y, y' := -x](2\,x\,x' + 2\,y\,y' = 0)}{y = c\,, x = 0 \vdash [\{x' = y, y' = -x\}](x^2 + y^2 = c^2)}$$

Apply Di rule

dL: Proof

Differential invariant rule:

$$\frac{\Gamma, q(x) \vdash p(x) \qquad q(x) \vdash [x' := f(x)](p(x))'}{\Gamma \vdash [x' = f(x)\,\&\,q(x)]p(x)}$$

$$\frac{\vdash\ 0^2 + c^2 = c^2 \qquad \vdash\ 2\,x\,y + 2\,y\,(-x) = 0}{\dfrac{y = c, x = 0\ \vdash x^2 + y^2 = c^2 \qquad \vdash [x' := y, y' := -x](2\,x\,x' + 2\,y\,y' = 0)}{y = c\,, x = 0\ \vdash [\{x' = y, y' = -x\}](x^2 + y^2 = c^2)}}$$

Apply substitution

Apply Di rule

dL: Proof

Differential invariant rule:

$$\frac{\Gamma, q(x) \vdash p(x) \qquad q(x) \vdash [x' := f(x)](p(x))'}{\Gamma \vdash [x' = f(x)\,\&\,q(x)]p(x)}$$

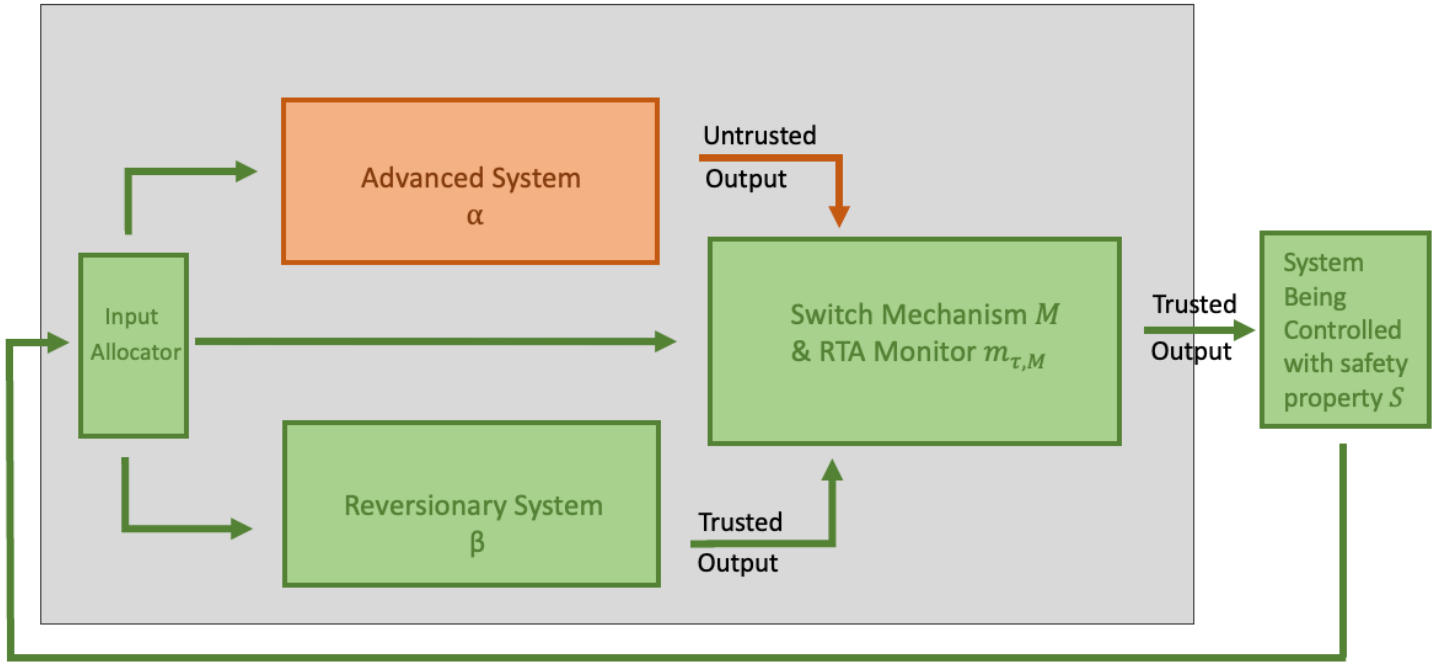
$$\frac{\dfrac{\dfrac{\dfrac{}{\vdash 0 = 0}\checkmark}{\vdash 0^2 + c^2 = c^2}\checkmark \qquad \vdash 2\,x\,y + 2\,y\,(-x) = 0}{y = c, x = 0 \vdash x^2 + y^2 = c^2 \qquad \vdash [x' := y, y' := -x](2\,x\,x' + 2\,y\,y' = 0)}}{y = c, x = 0 \vdash [\{x' = y, y' = -x\}](x^2 + y^2 = c^2)}$$

Arithmetic!

Apply substitution

Apply Di rule

Specifying RTA in Plaidypvs

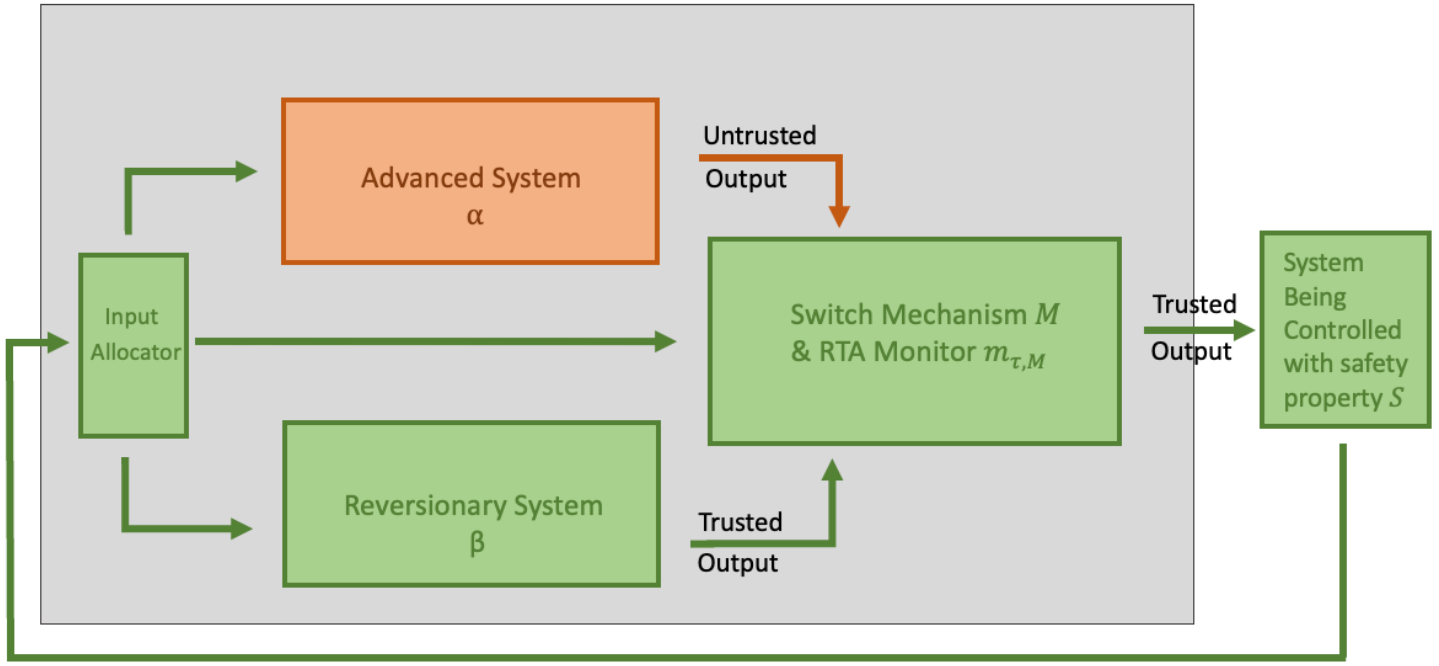


Define

$$RTA(\alpha,\beta)_{\tau,M} = \left(\left(?M\,;m_{\tau,M}(\alpha)\right) \cup (?\neg M\,;\beta)\right)^*$$

While M is true run α with timed monitor **and** While M is not true run β with monitor

Specifying RTA in Plaidypvs



RTA rule in Plaidypvs:

$$\frac{\Gamma \vdash S \wedge (M \vee G)\,, \quad S \vdash \left[m_{\tau,M}(\alpha)\right]\big(S \wedge (M \vee G)\big), \quad G \vdash [\beta^*]S}{\Gamma \vdash \left[RTA(\alpha,\beta)_{\tau,M}\right] S}$$

RTA rule in Plaidypvs

For a user given property G that carries over when switching from the advanced system to the reversionary system,

$$\Gamma \vdash S \wedge (M \vee G)$$

$$+$$

$$S \vdash [m_{\tau,M}(\alpha)](S \wedge (M \vee G))$$

$$+$$

$$G \vdash [\beta^*]S$$

$$\Gamma \vdash [RTA(\alpha,\beta)_{\tau,M}]\, S$$

IF

the vehicle starts in a safe state,

+

the monitored advanced system controller always ends in a state where the vehicle can recover,

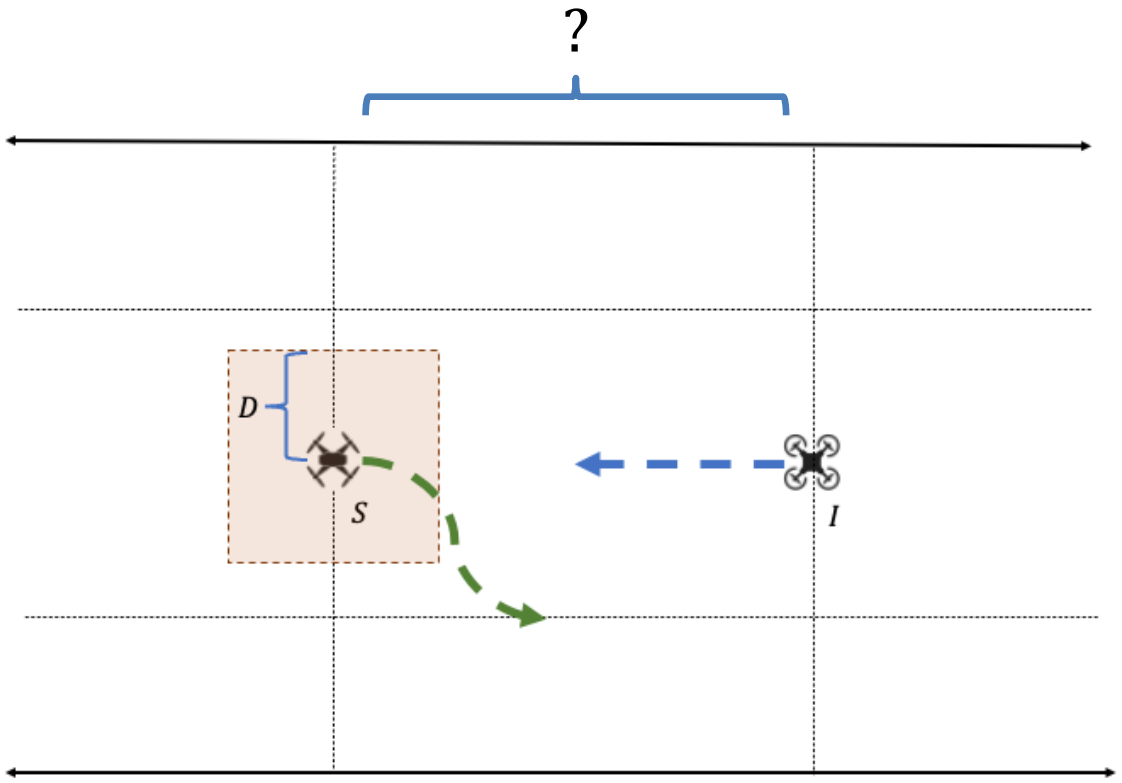
+

and when the vehicle starts in the carry-over state, repeated execution of the reversionary system controller leads to a safe state

THEN

the RTA system satisfies the safety property for every run.

Example: Productive conflict avoidance



Complex System

$$\alpha \equiv (S'_x = V_x, I'_x = I_v),$$

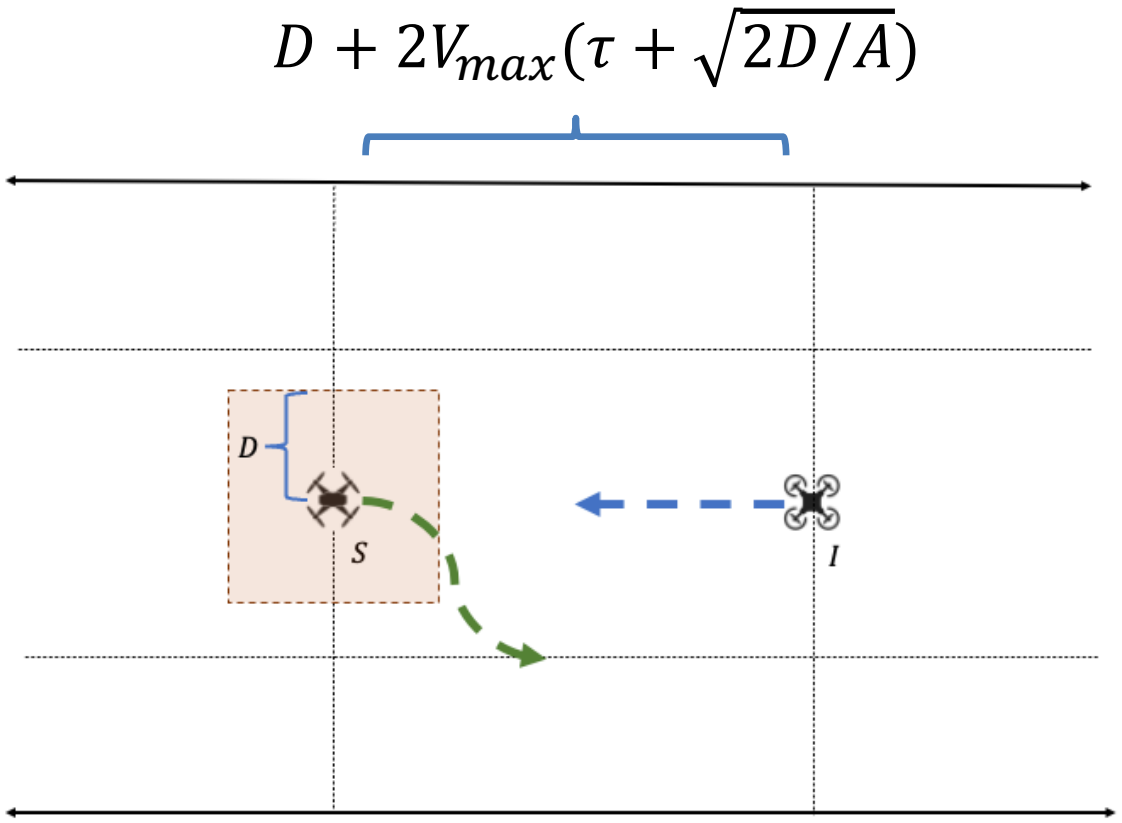
Reversionary System

$$\beta \equiv (S'_x = V_x, S'_y = V_y, V'_y = -A, I'_x = I_v),$$

Design Requirement:

What distance apart does the maneuver need to begin to guarantee well-clear?

Example: Productive conflict avoidance



Complex System

$$\alpha \equiv (S'_x = V_x, I'_x = I_v),$$

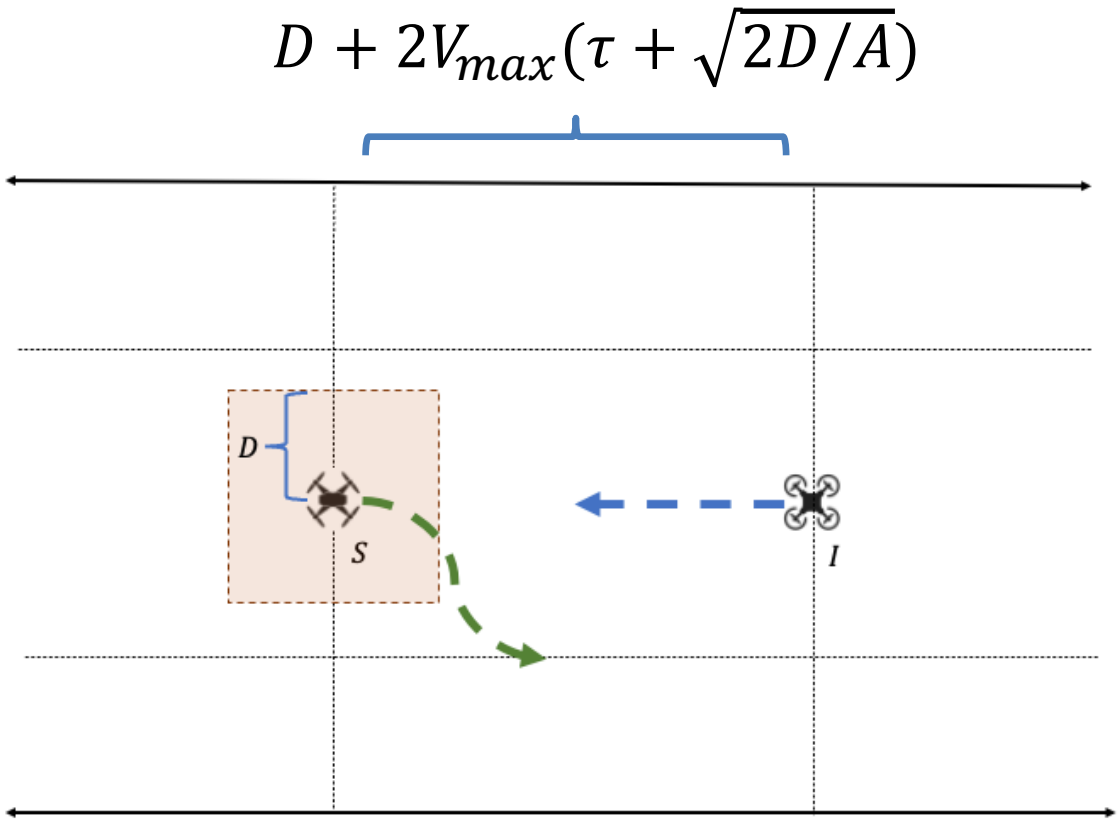
Reversionary System

$$\beta \equiv (S'_x = V_x, S'_y = V_y, V'_y = -A, I'_x = I_v),$$

Design Requirement:

What distance apart does the maneuver need to begin to guarantee well-clear?

Example: Productive conflict avoidance



Complex System

$$\alpha \equiv (S'_x = V_x, I'_x = I_v),$$

Reversionary System

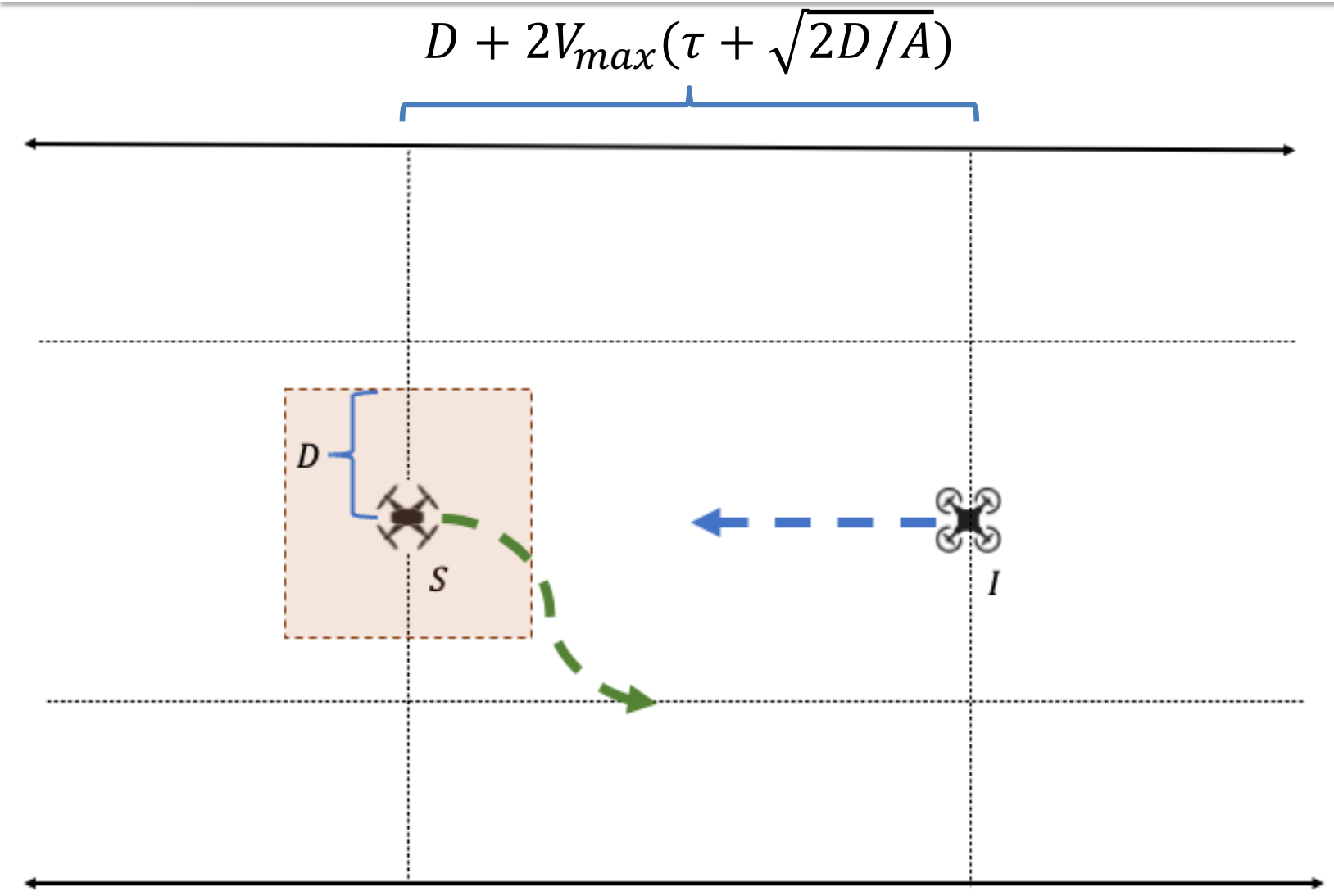
$$\beta \equiv (S'_x = V_x, S'_y = V_y, V'_y = -A, I'_x = I_v),$$

Design Requirement:

The sampling rate must satisfy:

$$\tau \leq \sqrt{\frac{2D}{A}}$$

Example: Productive conflict avoidance



Assumptions

+

Safety Requirements

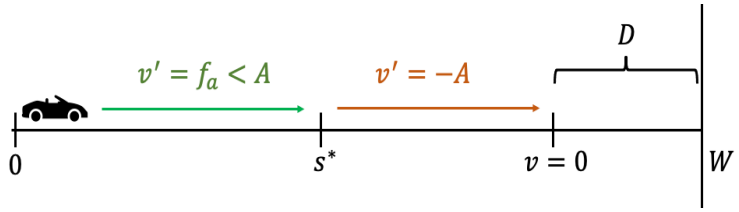
+

Plaidypvs verification

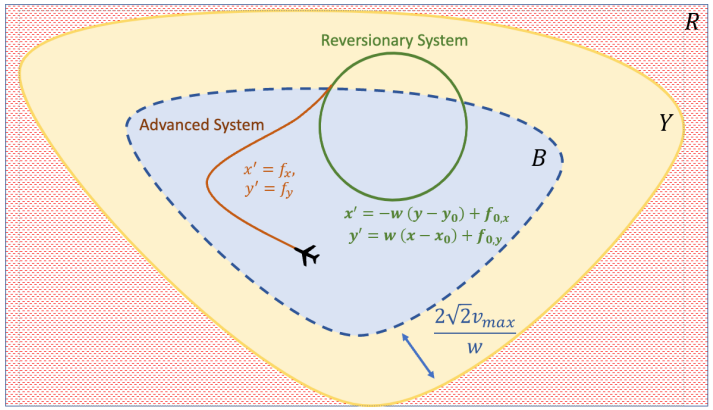
Design Requirements

Summary

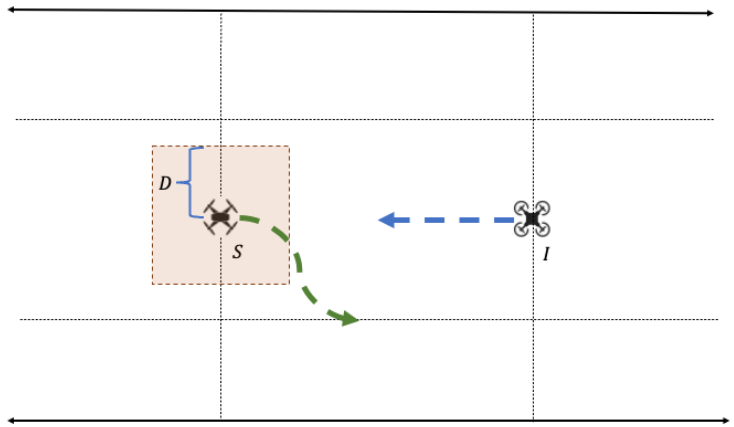
- Verified an RTA framework in Plaidypvs, the embedding of dL in PVS developed by NASA.
- Used the framework to verify instantiations of RTA framework
  - 1D autobraking
  - Geofenced operations
  - Productive conflict avoidance
- Used the framework to verify the safety of a simplified geofence with return to safe fallback
- Results presented as an SWS tech talk, and to NASA and FAA partners at a Research Transition Team meeting.



1D *autobraking* fallback.



Geofenced operations with a *return to safe region* fallback.



Productive conflict avoidance

References

Thank you for your attention!
Questions, comments?

Plaidypvs can

- model and formally reason about hybrid systems
- help extract and define requirements from the system

[1] A Verification Framework for Runtime Assurance of Autonomous UAS. J Tanner Slagel, Lauren M. White, Aaron Dutle, César Muñoz, Nicolas Crespo. To appear at DASC 2024.

[2] A Formal Verification Framework for Runtime Assurance. J Tanner Slagel, Lauren M. White, Aaron Dutle, César Muñoz, Nicolas Crespo. NFM 2024.

[3] A Temporal Differential Dynamic Logic Formal Embedding. Lauren M. White, Laura Titolo, J Tanner Slagel, César Muñoz. CPP 2024.

[4] Embedding Differential Dynamic Logic in PVS. J Tanner Slagel, Mariano Moscato, Lauren White, César Muñoz, Swee Balachandran, Aaron Dutle. LSFA 2023.

[5] Embedding Differential Temporal Dynamic Logic in PVS. Lauren White, Laura Titolo, J Tanner Slagel, TYPES 2023.

[6] Towards an Implementation of Differential Dynamic Logic in PVS. J Tanner Slagel, César Muñoz, Swee Balachandran, Mariano Moscato, Aaron Dutle, Paolo Masci, Lauren White. SOAP 2022.

Neural Network Formal Verification and Numerical Analysis

Work performed by

Anthony Dario[2], Aaron Dutle[1], Taylor Houtz[3], Mariano Moscato[2], Cesar Muñoz[1], John Siratt[1], Laura Titolo[2]

[1] NASA Langley Research Center

[2] Analytical Mechanics Associates

[3] NASA Langley OSTEM Intern

Problem: Machine Learning in Aeronautics

There is interest in deploying machine learning solutions for problems in aeronautics.

But can we do it in a safe way?

Key problems include

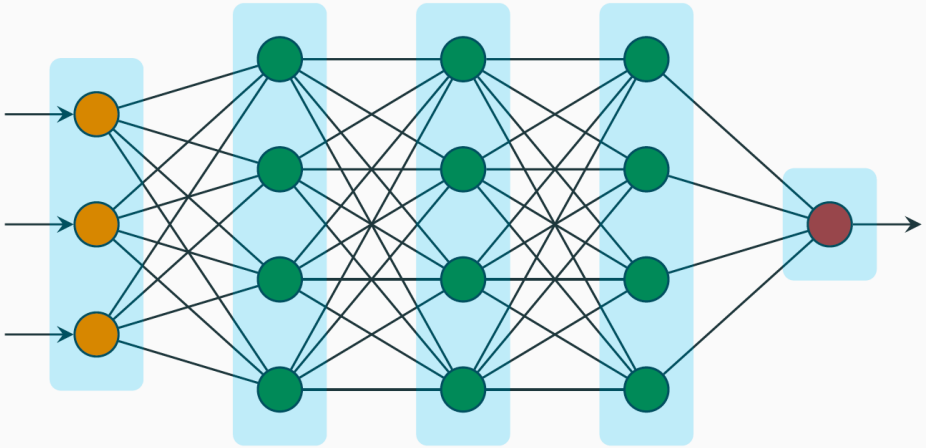
- Unpredictable behavior
- Round-off error in embedded systems

With a focus on feed-forward neural networks, we investigated these issues.

Background: Feed-Forward Neural Networks

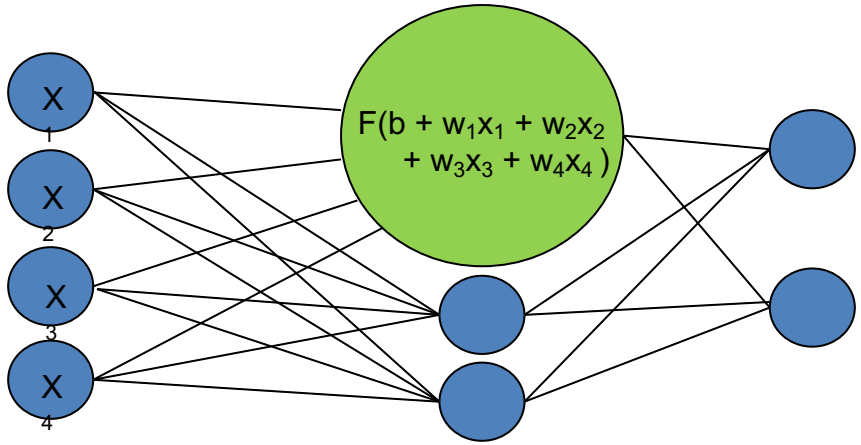
A *feed-forward neural network* (FFNN) is made up of:

- A sequence of **layers,** each taking vector input and giving vector output.
- Each layer contains **neurons** operating in parallel
- Each neuron operates by multiplying the input. vector with a vector of **weights**, adding **bias**, and applying an **activation** function to the result.



A general FFNN, with 5 layers (3 hidden).

A close-up of a single neuron, with weights $\mathbf{w_i}$, bias **b**, and activation function **F**.



In a *quantized* neural net, these operations are on fixed-point numbers.
This can compromise robustness due to rounding.

Analysis of Activation Functions: Formalization in PVS

- Various activation functions are used to introduce non-linearity.
- The choice can affect the complexity of training, execution, and verification.
- Some common activations were formalized in the PVS theorem prover (as real number functions), and certain properties were proven.

Activation Functions specified:

- ELU (Exponential Linear Unit)
- Gaussian Error Linear Unit
- Leaky ReLU
- ReLU (Rectified Linear Unit)
- Sigmoid
- Softmax
- Softplus
- Swish
- Tanh

Types of properties proven:

- Differentiability
- Derivative (where applicable)
- Monotonicity properties
- Upper and lower bounds
- Function values at interesting inputs

ReLU

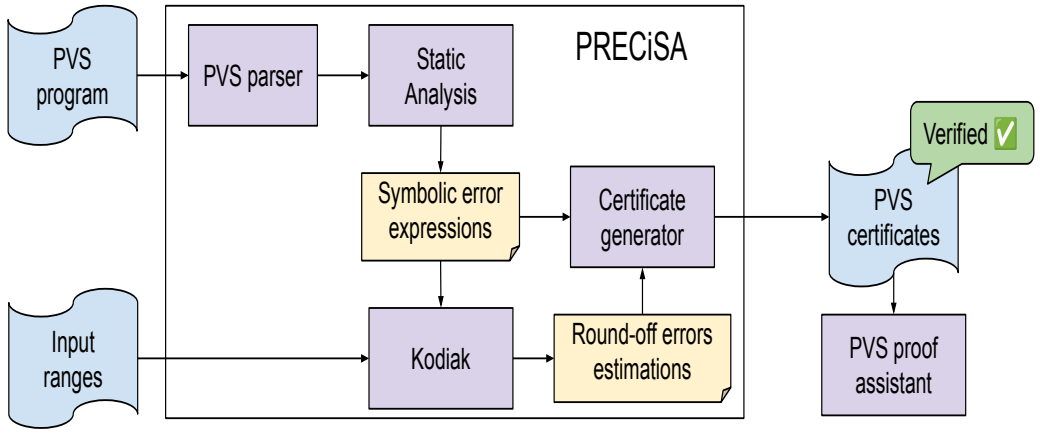
Sigmoid

Analysis of Activation Functions: Bounding Quantized Error

The NASA-developed prototype tool **PRECiSA** automatically computes a sound overestimation of the rounding error that may occur in floating-point programs.

Extending PRECiSA for fixed-point operations allowed for formal round-off error estimation for activation functions in quantized neural networks, such as might be found in embedded systems.

PRECiSA



Activation Function	Double-Precision Error	Fixed-Point[1] Error	Unstable?
Identity	1.776×10^{-15}	2.328×10^{-10}	No
Binary Step	0.0	0.0	Yes
ReLU	1.776×10^{-15}	2.328×10^{-10}	Yes
Leaky ReLU	1.776×10^{-15}	2.328×10^{-10}	Yes
Sigmoid	4.557×10^{-15}	2.328×10^{-10}	No
Hyperbolic Tangent	1.040×10^{-14}	1.107×10^{-9}	No
Softplus	7.661×10^{-15}	5.008×10^{-10}	No
Gaussian	7.438×10^{-15}	7.842×10^{-10}	No
Swish	1.063×10^{-14}	2.793×10^{-9}	No

Numerical Instability: Error Due to Quantization

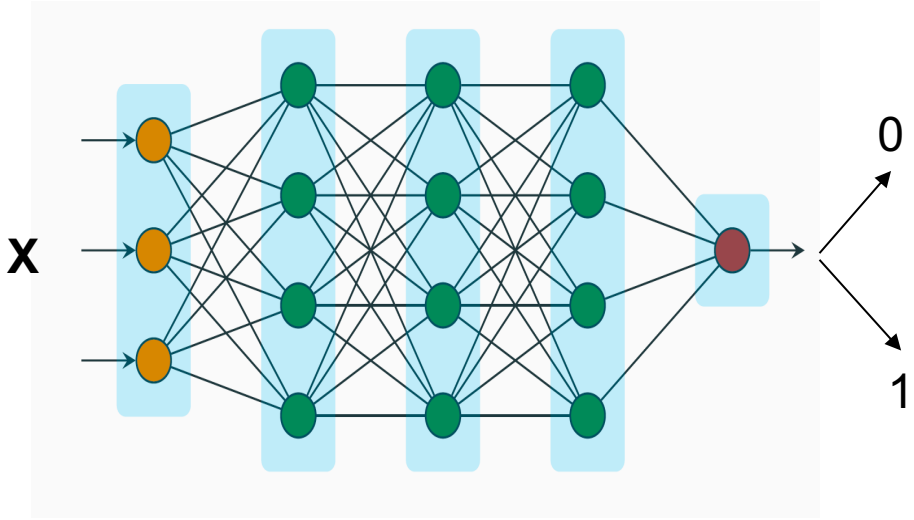
Is error from quantization even possible? How bad could it really be?

In the worst case, as bad as imaginable

Developed a collection of small binary classification NN examples:

- 5-layer with step activation
- 3-layer with step activation
- 2-layer with ReLU activation

Each have the following property:
For a fixed input **X**, weights, and biases, using decimal precision ranging from 1 to 5, the output is class 1 for odd precision, and class 0 for even precision.



A single input can *alternate* class as precision is lowered.

Problems:

- PRECiSA does not natively support matrices yet
- Scalability: thousands of neurons and input variables

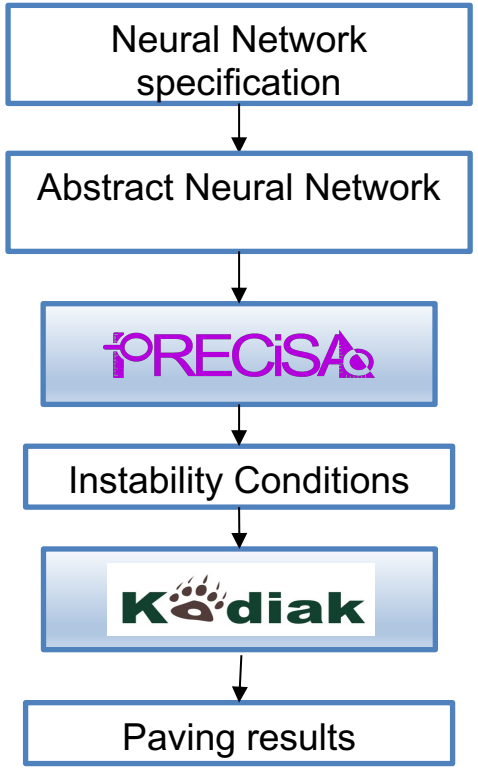
Solution: Generate an abstract neural network

- Layer-based abstraction
- Interval-based abstract domain to abstract the values of weights, bias and neurons in a single layer

Numerical Instability: Static analysis-based test case generation

- Certain activation functions such as Binary Step, ReLU and their variations are discontinuous and can be implemented as if-then-else statements.
- For these functions, a small error in the arguments can cause a large variation in the output (instability).

1. The neural network is approximated using a layer-based abstraction.
2. PRECiSA computes a set of Boolean conditions from the abstract neural network which encodes the cases in which an instability may occur.
3. These Boolean conditions are input to the NASA-developed global optimizer Kodiak that performs a paving.
4. The result of the paving is a set of “boxes” representing combinations of input variables ranges that satisfy the instability conditions.



Transforming ReLU Classifiers: Translating into Logic

Classification is a common problem in deep learning, and ReLU is one of the most common activations for deep neural nets.

Extending existing results for 2-layer networks, we constructively showed that such classifiers are embeddable in a manageable fragment of first-order arithmetic.

The number of inequalities in the naïve translation grows exponentially with the number of neurons; *however*, they are not independent.

The large Boolean combination of inequalities at the right is the transformation of a network with 2-layers, each with 2 neurons, expressed as a formula in PVS.

```
net_translated(x0,x1): boolean =
((((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((0*x0 + 0*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(0*x0 + 0*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0))))) AND
((NOT ((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((30*x0 + -30*x1 + 5 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-2*x0 + -1*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(28*x0 + -31*x1 + 5 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0)))))) AND
(NOT ((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((6*x0 + -6*x1 + 1 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((6*x0 + 3*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(12*x0 + -3*x1 + 1 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0)))))))) OR
((((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((-18*x0 + 18*x1 + -3 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-18*x0 + -9*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(-36*x0 + 9*x1 + -3 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0))))) AND
((NOT ((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((30*x0 + -30*x1 + 5 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-2*x0 + -1*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(28*x0 + -31*x1 + 5 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0)))))) AND
((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((6*x0 + -6*x1 + 1 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((6*x0 + 3*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(12*x0 + -3*x1 + 1 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0)))))))) OR
((((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((60*x0 + -60*x1 + 10 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-4*x0 + -2*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(56*x0 + -62*x1 + 10 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0))))) AND
(((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((30*x0 + -30*x1 + 5 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-2*x0 + -1*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(28*x0 + -31*x1 + 5 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0))))) AND
(NOT ((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((6*x0 + -6*x1 + 1 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((6*x0 + 3*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(12*x0 + -3*x1 + 1 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0)))))))) OR
(((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((42*x0 + -42*x1 + 7 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-22*x0 + -11*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(20*x0 + -53*x1 + 7 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0))))) AND
(((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((30*x0 + -30*x1 + 5 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((-2*x0 + -1*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(28*x0 + -31*x1 + 5 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0))))) AND
((0*x0 + 0*x1 + 0 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
((6*x0 + -6*x1 + 1 > 0 AND ((NOT 2*x0 + 1*x1 + 0 > 0) AND 6*x0 + -6*x1 + 1 > 0)) OR
((6*x0 + 3*x1 + 0 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND (NOT 6*x0 + -6*x1 + 1 > 0))) OR
(12*x0 + -3*x1 + 1 > 0 AND (2*x0 + 1*x1 + 0 > 0 AND 6*x0 + -6*x1 + 1 > 0)))))))))))
```

Transforming ReLU Classifiers: Simplification via PVS

Through a combination of PVS automated methods and manual manipulation, the system of inequalities can be reduced to the following system of 4 inequalities.

```
simplified_net(x0,x1): boolean =
  (6*x0 - 6*x1 + 1 > 0) AND
  (2*x0 + x1 <= 0 OR (28*x0 - 31*x1 + 5 > 0 AND 20*x0 - 53*x1 + 7 > 0))

translation_simplified: THEOREM
  net_translated(x0,x1) IFF simplified_net(x0,x1)
```

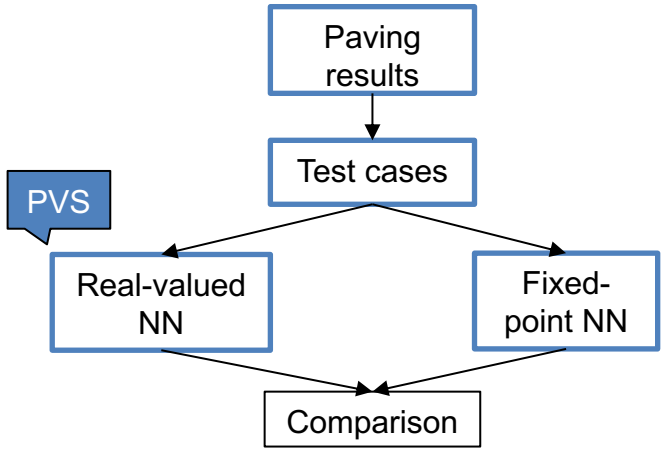
The equivalence proof is fully automatic.

These methods can be used not only to characterize and verify the behavior classifier, but to compress a network into a more efficient computation.

Such compression has the added benefit of reducing the opportunities for round-off error associated with activation functions.

Future work

- PRECiSA
  - Add matrix support.
  - Investigate novel abstraction techniques to scale up analysis.
  - Generate critical test cases for differential testing.



- PVS
  - Develop new library capabilities to better support neural net reasoning.
  - Create proof strategies to further automate simplification of neural net characterizations.

- Transformation
  - Integrate simplification procedures into translation steps for scaling.
  - Implement algorithm and verify in PVS.