Formal Verification of Safety-Critical Aerospace Systems

Saswata Paul⋆, Elkin Cruz⋆, Airin Dutta⋆, Ankita Bhaumik⋆, Erik Blasch†, Gul Agha‡, Stacy Patterson⋆, Fotis Kopsaftopoulos⋆, and Carlos Varela⋆,⋄

***Abstract*—Developments in autonomous aircraft, such as *electrical vertical take-off and landing* (eVTOL) vehicles and multicopter drones, raise safety-critical concerns in populated areas. This article presents the ASSURE (*Analysis of Safety-Critical Systems Using Formal Methods-Based Runtime Evaluation*) framework, which is a collection of techniques for aiding in the formal verification of safety-critical aerospace systems. ASSURE supports the rigorous verification of deterministic and non-deterministic properties of both distributed and centralized aerospace applications by using formal theorem proving tools. We present verifiable algorithms and software, formal reasoning models, formal *proof libraries*, and a data-driven runtime verification approach for aerospace systems towards a provably safe *Internet of Planes* (IoP) infrastructure.**

***Index Terms*—theorem proving, runtime verification, stochastic systems, distributed systems**

Important Acronyms:

AoA:	Angle of Attack
ASSURE:	Analysis of Safety-Critical Systems Using Formal Methods-Based Runtime Evaluation
DAC:	Decentralized Admission Control
DDDAS:	Dynamic Data-Driven Applications Systems
eVTOL:	Electrical Vertical Take-Off and Landing
FAM:	Failure-Aware Actor Model
IoP:	Internet of Planes
MTR:	Multicopy Two-Hop Relaying
NMAC:	Near Mid-Air Collision
TAP:	Two-Phase Acknowledge Protocol
TLAPS:	Temporal Logic of Actions Proof System
UAM:	Urban Air Mobility

I. Introduction

THE rapid advancement in the field of autonomous aircraft has revolutionized scientific data collection, package delivery, disaster relief, agriculture surveillance, weather tracking, and passenger transportation. As autonomous aircraft technologies evolve to become more accessible and cost-efficient, it will become necessary to rigorously verify their performance to guarantee that they will operate safely, securely, and efficiently.

⋆Rensselaer Polytechnic Institute, Troy, NY, USA.
†Air Force Research Laboratory, Rome, NY, USA.
‡University of Illinois Urbana-Champaign, Champaign, IL, USA.
⋄Corresponding author. Email: cvarela@cs.rpi.edu
This research was partially supported by the Air Force Office of Scientific Research, DDDAS Grant No. — FA9550-19-1-0054 and the National Science Foundation, Grant No. — CNS-1816307.

A. Motivation

Aerospace systems are *safety-critical* and errors in aerospace operations such as flight control and navigation can be catastrophic. Recently, there were two fatal accidents involving the Boeing 737 Max 8 series of aircraft — Lion Air flight 610 on October 29, 2018 (189 fatalities) and Ethiopian Airlines flight 302 on March 10, 2019 (157 fatalities). These were caused by failing sensors sending erroneous data to the *Maneuvering Characteristics Augmentation System* (MCAS), which is a flight stabilization software designed to automatically prevent aircraft from *stalling* during flight by adjusting the *angle of attack* (AoA) (Fig. 1) [1]. Similarly, navigational errors can lead to *mid-air* collisions, such as in 2002 at Überlingen [2], where a Boeing 757 cargo jet and a Tupolev Tu-154 passenger jet collided mid-air (Fig. 2) causing 69 fatalities, and the mid-air collision involving a Cirrus SR-22 airplane and a Swearingen Metroliner airplane near Centennial Airport, Denver, Colorado, USA in May 2021 [3].

The safety-critical nature of aerospace applications warrants requirements for irrefutable system assurance guarantees for any tool or technique used in such applications. Such assurance guarantees can be obtained through rigorous verification and validation (V&V) procedures. Examples of aerospace V&V include software *built-in-testing* (BIT) [4] and hardware sensor failure analysis and mitigation via *analytical redundancy* [5]. However, the complex nature and dynamic operating conditions of such aerospace applications make it impossible to exhaustively test them to provide such strong safety guarantees. Under such circumstances, *formal methods, tools, and techniques*, can be used for the verification of the critical properties of such applications, *e.g.,* by mechanically checking the proofs of correctness guarantees.

B. Formal Methods

In avionics engineering, imprecision can lead to catastrophic errors in aerospace systems, thereby presenting a strong incentive for using precise mathematical techniques for specifying and reasoning about these systems. Formal methods use rigorous mathematical techniques to reason about the critical properties of hardware and software systems. Mathematical logic provides a natural framework for precisely constructing and expressing various concepts in computing and lends itself well to formalization [6]. Formal verification of safety-critical systems requires tools and

Note : This paper has been accepted in the IEEE AESM Magazine
DOI 10.1109/MAES.2023.3238378

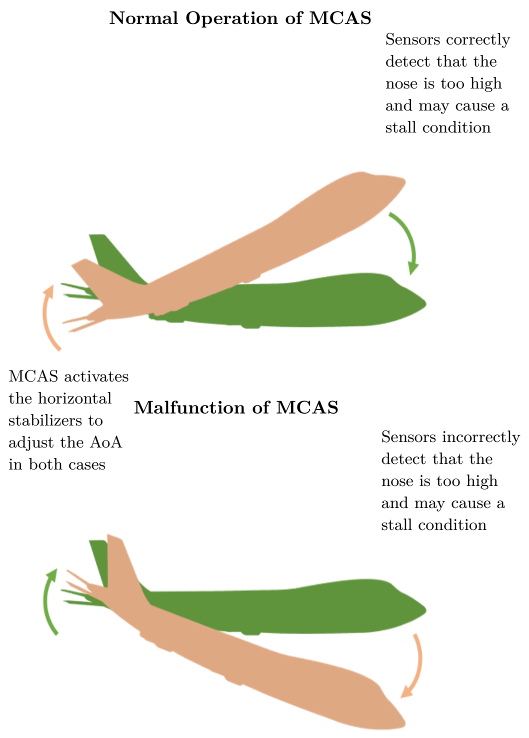

Normal Operation of MCAS
Sensors correctly detect that the nose is too high and may cause a stall condition
MCAS activates the horizontal stabilizers to adjust the AoA in both cases
Malfunction of MCAS
Sensors incorrectly detect that the nose is too high and may cause a stall condition

Fig. 1: Boeing's MCAS system's failure.

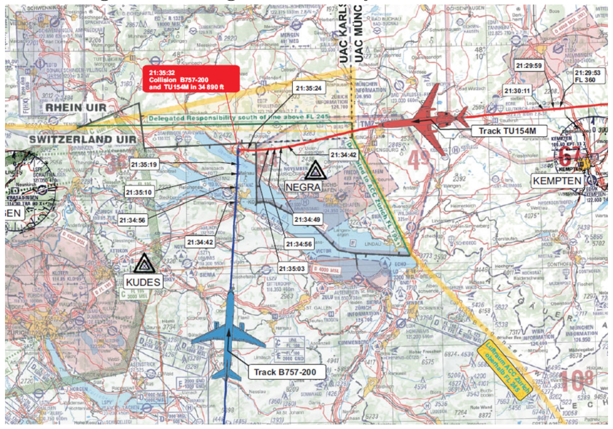


Fig. 2: The flight paths for the 2002 Überlingen mid-air collision incident (Image credits: https://www.skybrary.aero/index.php/T154_/_B752,_en-route,_Uberlingen_Germany,_2002).

techniques for mechanically checking proofs of correctness guarantees. There are two main uses of formal methods in system development:

- *Specification* - the process of describing a system and its required properties using a formal language with mathematically well-defined syntax and semantics.
- *Verification* - the process of determining if the specified system satisfies the required properties using *theorem proving* and/or *model checking*.

1) Theorem Proving: Theorem proving entails creating a set of statements $\mathcal{S}$ that reflects the specification of a system and a set of formulae $\mathcal{F}$ that represents the required correctness properties as constraints on states. The theorem proving technique allows models with possibly infinitely large state spaces as there is no need to exhaustively explore all the reachable states for a given model. Rather, a set of logical *inference rules* is used to construct proofs that can mathematically establish that $\mathcal{S}$ entails $\mathcal{F}$. Infinite state spaces are usually modeled by using *universal quantification* (denoted by the $\forall$ symbol) which implies that a property can be satisfied by all possible members of a *domain*. Theorem proving allows us to reason about constraints on states in addition to individual state instances. Proof assistants and automated theorem provers like Athena [7] and Temporal Logic of Actions Proof System (TLAPS) [8] can automatically search for and check proofs in defined domains [9].

2) Model Checking: In *model checking* [10], an abstract model $\mathcal{M}$ of a system is first constructed in the form of variations of finite state automata and a set of formulae $\mathcal{F}$ is constructed that specifies the correctness properties [9]. It is then established that $\mathcal{M}$ semantically entails $\mathcal{F}$ by exploring the entire set of reachable states for $\mathcal{M}$. If a property does not hold in all of the reachable states, then at least one valid counter-example is produced and the property is said to be false. Models can be designed for both software and hardware systems before the actual systems are developed, but to successfully apply model checking, the models need to have finite state spaces. The task of verification can be fully automated by using model checking software like the TLA$^+$ Model Checker [11].

Compared to model checking, theorem proving is a more expressive approach for verification since theorems are usually defined as logical constraints that hold over infinite system states, which is in contrast to the finite set of states used in model checking. Moreover, model checking is limited by the *state space explosion* problem which makes the verification of many real-world applications with large state space dimensionality intractable.

C. Aerospace Challenges

Aerospace systems are inherently complex involving intricate interactions between different types of avionics components such as multi-modal sensors, control inputs, flight control surfaces, as well as avionics intra-aircraft and inter-aircraft data transfer via fiber optic cables or radio frequency transponders. Some important challenges involved in developing verifiable aerospace systems are:

- Heterogeneous avionics components can vary significantly in their specifications, correctness requirements, and properties. This makes it necessary to employ different types of techniques for the formal verification of aerospace systems.
- The characteristics of aerospace systems are defined by a mixture of both *deterministic* properties, where the system behaviors under different possible operating conditions are well understood; and *non-deterministic* properties, where the behaviors are dependent on unpredictable uncertainties.
- The dynamicity of the operating conditions of real-world aerospace systems makes it difficult to pro-

vide formal guarantees that are valid during deployment. Formal proofs that are developed in the pre-deployment stages may cease to be valid at runtime if the actual operating conditions do not conform to the assumptions made during reasoning.
- There can be a disconnect between the formal specification of an avionics software module and its actual code, which implies that even if the specification is verified to satisfy some property, the code may not be.
- Developing formal specifications of complex aerospace systems requires interdisciplinary domain knowledge and high proficiency in formal methods tools.

*D. ASSURE: **A**nalysis of **S**afety-Critical **S**ystems **U**sing Formal Methods-Based **R**untime **E**valuation*

Owing to the challenges posed by the complexity of aerospace systems and their dynamic operating conditions, the formal verification of such systems warrants the use of different types of existing techniques and the development of new techniques to address these challenges. Therefore, this article presents the ASSURE (***A**nalysis of **S**afety-Critical **S**ystems **U**sing Formal Methods-Based **R**untime **E**valuation*) framework, which is a collection of formal techniques for the verification of real-world safety-critical systems. These techniques can be used to specify critical aerospace applications ranging from autonomous navigation to flight state estimation and verify their correctness using mathematically rigorous formal logic. The primary contributions of this article are – (1) surveying different ASSURE techniques for the verification of aerospace systems: *Hoare logic* [12]-based techniques (using Dafny [13]), *temporal logic* [14] based techniques (using TLAPS), constructive techniques using dependent type theory (using Agda [15]), and *first-order logic* theorem proving techniques (using Athena); and (2) presenting ASSUREs reusable library-based theorem proving approach with *dynamic data-driven* [16] verification to reason about stochastic aerospace systems (using Athena).

The rest of the article is structured as follows: Section II describes ASSURE techniques for the verification of deterministic properties of aerospace systems by presenting: (1) a formally verified deterministic algorithm for strategic conflict detection and avoidance and (2) models for reasoning about deterministic properties of *distributed coordination protocols* that can be used for autonomous air traffic management; Section III describes ASSURE techniques for the verification of probabilistic properties of aerospace systems by presenting: (1) models for reasoning about probabilistic properties of aerospace applications such as fixed-wing aircraft stall detection, hexacopter rotor fault detection, and multi-aircraft coordination, (2) a proof library in Athena for reasoning about probabilistic properties of airborne distributed protocols, (3) formal *correctness envelopes* that can divide the operational state space into regions where a formal proof may or may not hold, and (4) a runtime verification approach based on *the Dynamic Data-Driven Applications Systems* (DDDAS) [16] paradigm; Section IV presents a brief discussion on the different verification techniques we have developed as a part of ASSURE; and Section V concludes the article and presents some potential directions for future work.

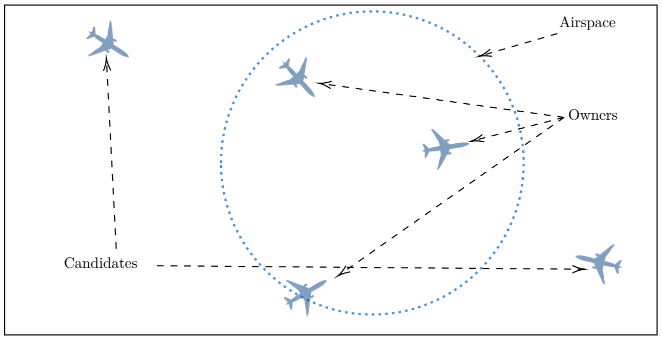


Fig. 3: Aircraft implementing DAC for an airspace.

II. Verifying Deterministic Properties of Aerospace Systems

A deterministic system is a system in which there is no randomness involved in the change of states, *i.e.*, under a given input, the system will always produce the same output as long as the initial state is the same. There are several important applications of deterministic systems that are required for the safe operation of aerospace systems. To motivate deterministic system analysis, this section introduces an autonomous air traffic management technique called *decentralized admission control* (DAC) and describes the formal verification of some deterministic properties relevant to DAC.

DAC is an autonomous *separation assurance* technique that allows a group of aircraft to collaboratively plan safe operations through well-defined four-dimensional airspaces for *urban air mobility* (UAM) scenarios. In DAC, an airspace has a set of *owner* aircraft which already have the authorization to carry out a predefined set of *flight plans* through the airspace. The set of flight plans of the owners is *safe*, *i.e.*, there is no possibility of *near mid-air collisions* (NMAC) between any subset of flight plans in the set. One or more *candidate* aircraft can then use the knowledge of this safe set to compute a *conflict-free flight plan* that is compatible with the safe set (Fig. 3). A candidate can then propose its conflict-free plan to the owners to gain authorization to use the airspace. When a candidate is authorized to use an airspace, the set of owners changes to include the candidate and all future candidates must use this new set of owners to compute their conflict-free proposals.

In DAC, participating aircraft must coordinate in a decentralized manner over the *Internet of Planes* (IoP) — a *vehicle-to-vehicle* (V2V) network of aircraft, ground stations, and satellites. The IoP is characterized by different aspects such as the distributed coordination protocols used by the aircraft, and the nature of airborne communication. To allow for a provably safe IoP infrastructure, we use the ASSURE framework to verify such aspects of the IoP.

Decentralized coordination in DAC for authorizing a single candidate involves two consecutive stages:

1) *Consensus*, which requires a set of agents to reach an agreement on some value and is a fundamental problem in distributed systems. There may be multiple candidates concurrently competing for admission into an airspace, but the owners can only admit the candidates sequentially because the competing candidates do not consider each other in their proposals and admitting one candidate potentially invalidates the proposals of all other candidates. Hence, a *distributed consensus protocol* must be used to ensure that only a single candidate's proposal is chosen at a time.
2) *Knowledge Propagation* is the act of sending a fact throughout a network to attain a desired *state of distributed knowledge*. When a candidate is authorized, the set of owners changes to include the candidate, and all aircraft *relevant* to the airspace must be informed about the new set of owners so that they can learn the new value. Aircraft relevant to an airspace comprises the set of new owners and the set of candidates expected to try to enter the airspace in the future (this includes the candidates who may have been previously denied authorization). Consensus only guarantees that a sufficiently large subset of all relevant aircraft will learn about the agreement. Therefore, after a candidate is authorized, a *distributed knowledge propagation protocol* must propagate the knowledge of the agreement to all relevant aircraft.

Several types of deterministic correctness properties are relevant for DAC, *e.g.,* there must be a guarantee that the conflict-free proposal computed by the candidates accurately avoids NMACs with the owners' flight plans. Similarly, there are two types of deterministic properties of interest for the distributed protocols used in DAC: (1) *the distributed protocol will correctly achieve its intended goal* (called *safety*) and (2) *the distributed protocol will eventually achieve its intended goal* (called *eventual progress*). The next section highlights some examples of formal verification of such deterministic properties.

A. Deterministic Algorithms

For critical properties, such as the property that the proposals computed by the candidates must have no conflict with the owners' flight plans, it is necessary to ensure that the property is never violated. Therefore, algorithms that are used to compute such proposals must be deterministic in order to ensure that the property is valid for all inputs to the algorithm. In [17], we presented `solve`, a deterministic algorithm to compute NMAC-free proposals for DAC.

An NMAC is a potentially catastrophic interaction between two or more aircraft that is caused by the loss of safe separation between the aircraft. A distance-based method for detecting NMACs uses a cylindrical "hockey puck"-shaped volume around each aircraft called the *well-clear volume* (with diameter D and height H) [18]. During

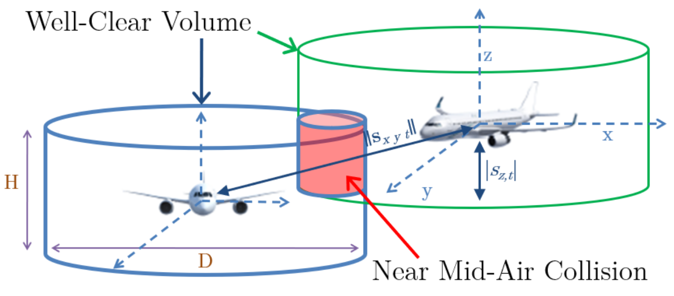


Fig. 4: An NMAC between two aircraft.

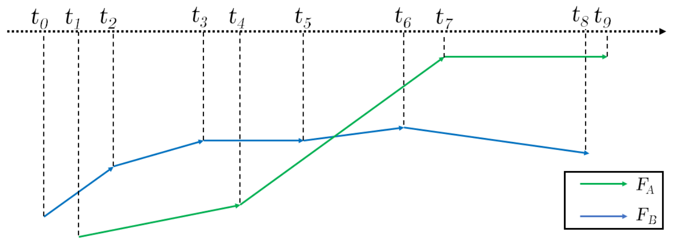
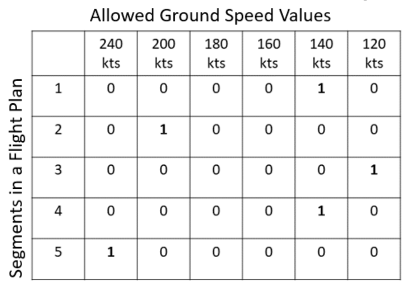


Fig. 5: Periods of interest for two flight plans.



Allowed Ground Speed Values

Segments in a Flight Plan	240 kts	200 kts	180 kts	160 kts	140 kts	120 kts
1	0	0	0	0	**1**	0
2	0	**1**	0	0	0	0
3	0	0	0	0	0	**1**
4	0	0	0	0	**1**	0
5	**1**	0	0	0	0	0

Fig. 6: Sample assignment matrix for the ground speeds.

an NMAC, the well-clear volumes of the involved aircraft intersect each other. At any time t, an NMAC arises if two conditions simultaneously hold: $\|\mathbf{s}_{xy,t}\| < D$ and $|s_{z,t}| < H$, where $\|\mathbf{s}_{xy,t}\|$ and $|s_{z,t}|$ represent the relative horizontal and vertical positions at t. This conflict detection logic works only when the states of two aircraft a and b at a time t_0 are available as S_{a,t_0} and S_{b,t_0}, along with a known *period of interest* during which both aircraft must maintain constant ground speed. However, candidates in DAC must be capable of detecting conflicts between two or more multi-segment flight plans where each flight plan is a sequence of multiple constant ground speed segments. Given two flight plans F_a and F_b, it is, therefore, necessary to discretize the temporal dimension into periods of interest (*e.g.*, Fig. 5, shows the top-view of two flight plans). The times corresponding to the way-points of the flight plans discretize the temporal dimension (represented by the dotted line) into consecutive periods of interest, demarcated by time points. Periods of interest are only valid if two flight plans have temporal overlap during that period (*e.g.*, in Fig. 5, the periods t_0 to t_1 and t_8 to t_9 are not valid periods of interest).

`solve` is a recursive algorithm that can be used by a candidate to compute an NMAC-free flight plan proposal for DAC, given a set of flight plans $\Phi = \{F_1, ..., F_N\}$ for the owners. In our approach, the candidate first decides on a desired set of waypoints in 3D space. Then, a conflict-free flight plan is computed by assigning a suitable *ground speed* to each segment between the waypoints such that $(\|\mathbf{s}_{xy,t}\| < D) \wedge (|s_{z,t}| < H)$ is never true with respect to any plan in Φ. A suitable set of ground speed assignments is denoted by a matrix where each row represents a segment in the flight plan and each column represents a possible value of ground speed (Fig. 6).

Using Athena, we have mechanically verified the following deterministic properties for `solve` using inductive formal proofs over the algorithm's specifications:

Theorem 1. *(Safety) If* `solve` *returns a complete assignment matrix M, then adding the flight plan F corresponding to M to the set of traffic flight plans Φ results in a safe set of flight plans.*

Theorem 2. *(Completeness) If* `solve` *returns an incomplete assignment matrix M, then there does not exist a valid assignment of ground speeds M, such that adding its corresponding flight plan F to the set of traffic flight plans Φ results in a safe set of flight plans.*

B. Models for Reasoning about Distributed Systems

Formal specifications of algorithms are usually expressed using logical models that support reasoning about correctness properties. We have used the *actor model* [19], [20] and the *temporal logic of actions* (TLA) [14] to reason about consensus and knowledge propagation respectively.

1) Actor Model: The actor model is a theoretical model of concurrent computation that can be used for reasoning about distributed systems. *Actors* are self-contained, concurrently interacting entities of a computing system that communicate by message passing [21]. In [22], we have used the actor model to formally reason about eventual progress in the *Synod consensus protocol* as it assumes *asynchronous communication* and *fairness*, which are helpful for reasoning about progress in networks like the IoP.

In asynchronous communication, message transmission delays are unbounded, so message transmission delays cannot be differentiated from processing delays or in the extreme case, agent failures. The IoP is an open network in which aircraft may experience permanent or temporary communication failures that may render them unable to send or receive any messages. Failures may be caused by all messages to and from an aircraft getting delayed because of transmission problems or due to internal processing delays in the aircraft. Therefore, for formal reasoning, it must be possible to specify if an actor has failed at any given time, *i.e.*, if it is incapable of sending or receiving messages. To specify such failures, in [22], we introduced a *failure-aware actor model* (FAM), which allows us to formally reason about temporary and permanent failures.

The Synod consensus protocol was proposed in [23] as the fundamental consensus protocol used by *Paxos*. Synod assumes an asynchronous, non-Byzantine[1] system model in which agents may fail and restart, and have stable storage. Synod assumes that messages can be duplicated, lost, and can have arbitrary transmission times, but cannot be corrupted. It consists of two logically separate sets of agents: *proposers*, which are the agents that can propose values for agreement, and *acceptors*, which are agents that can vote on which value should be agreed upon. A proposer learns that its proposal has been *chosen* if it receives *voted* messages from a quorum of acceptors.

Synod is ideal for consensus in DAC because it guarantees the safety property that only one value will be agreed upon by the acceptors. However, because of the asynchronous nature of communication in the IoP, it becomes challenging to guarantee progress in Synod since there is no *unique leader*. The Fischer, Lynch, and Paterson (FLP) impossibility result [24] implies that in purely asynchronous systems, where agent failures cannot be differentiated from message delays, leader election cannot be guaranteed. Therefore, it is important to identify the fundamental conditions under which progress can be formally guaranteed in leaderless protocols like Synod.

We have identified a set of conditions (called `CND`) under which *eventual progress* can be guaranteed — *"eventually, a nonfaulty proposer must propose a proposal number, that will not be interrupted by higher-numbered proposals, to a quorum of nonfaulty acceptors, and the Synod-specific messages between these agents must be eventually received"* [22]. Using FAM, we have formally specified the interactions between Synod agents and verified the following eventual progress property in Athena (Fig. 7).

Theorem 3. *(Eventual Progress in Synod) Given* `CND`*, a proposer p will eventually learn that a proposal number b has been chosen.*

2) Temporal Logic of Actions: TLA is a model that can be used to specify the behavior of concurrent systems and reason about their correctness properties [14]. In TLA, a system has a set of states and the transition between states are caused by *actions*. An action is a mathematical expression that contains *primed* and *unprimed* variables where unprimed variables represent the current state and primed variables represent the future state after the action has taken place [25].

Any knowledge propagation protocol used in DAC should ensure a *safe state of knowledge* in which all aircraft can "feel safe" to operate. In knowledge logic [26], the expression $k_i\Phi$ represents that *an agent i knows the fact* Φ and $E\Phi$ represents that *every agent E in the system knows* Φ. The expression $E^2\Phi$ represents a higher state of knowledge than $E\Phi$ and implies that every agent knows two facts— Φ and $E\Phi$. In the context of DAC, if Φ

[1]A Byzantine failure is one where an agent can appear to fail or give erroneous data to some agents while appearing to work correctly to some other agents. In a non-Byzantine system model, no Byzantine failures are considered, *i.e.*, an agent either fails or communicates following the protocol's specification from the perspective of all participating agents.

represents the new set of flight plans of the owners after consensus, then the absence of $E^2\Phi$ will create a scenario in which the autonomous aircraft will not be able to trust that the operation of DAC will be safe. Fagin *et al.* [26] succinctly explain this concern with the following example — *even if all drivers in a society know the rules for following traffic lights and follow them, that is not enough to make a driver "feel safe". This is because unless a driver knows that everyone else knows the rules and will follow them, then the driver may consider it possible that some other driver, who does not know the rules, may run a red light.* Hence, in [27], we have specified $E^2\Phi$ as a safe state of knowledge for DAC and presented a *Two-Phase Acknowledge Protocol* (TAP) that can be used by aircraft to attain $E^2\Phi$ after consensus has been reached on Φ.

TAP assumes an asynchronous, non-Byzantine system model in which agents operate concurrently and may experience communication failures. TAP also assumes *reliable messaging* [28] where message delivery is guaranteed. Messages can be duplicated but cannot be corrupted. In TAP, there is a non-empty set of *propagators*, and a logically separate non-empty set of *replicas*. Each propagator knows which agents are in the set of all replicas. A single value Φ is known to all propagators. $E^2\Phi$, in the context of TAP, implies that *all replicas know that all other replicas know* Φ. The goal of every propagator is to propagate Φ and eventually learn that $E^2\Phi$ has been achieved.

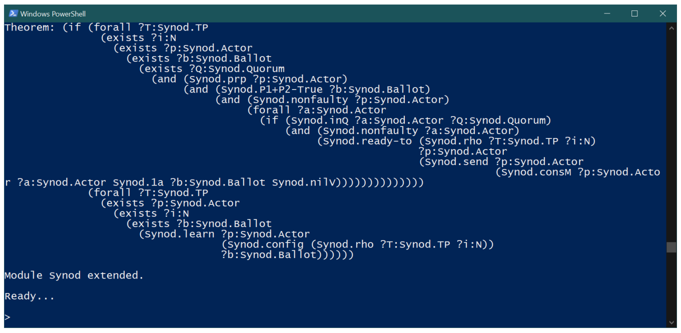
Under two additional conditions beyond the system model of TAP — *no replica can permanently fail* and *at least one propagator does not permanently fail* — we have mechanically verified the following correctness properties of TAP using TLAPS (Fig. 8):

Theorem 4. *(Safety in TAP) A propagator can learn that $E^2\Phi$ has been attained if and only if all replicas know $E\Phi$.*

Theorem 5. *(Eventual Progress in TAP) Eventually, some propagator will learn that $E^2\Phi$ has been attained.*

III. Verifying Probabilistic Properties of Aerospace Systems

The complex nature of aerospace systems and the uncertain operating conditions presented by real-world conditions imply that guarantees of deterministic correctness cannot always be provided for all aspects of such systems. Guarantees of probabilistic correctness can still be provided for such *stochastic* aerospace systems using statistical theory [29]. In the absence of guarantees of deterministic properties, guarantees of probabilistic properties can still help pilots or flight control computers make important operational decisions. As an example, because of the asynchronous nature of communication in the IoP, it is impossible to deterministically bound the time that may be required for consensus in DAC. However, if there is a guarantee that consensus will take at most 1 second with high probability (*e.g.*, 99.999999999%), then a candidate can decide to only compute flight plans that start after 1 second. Similarly, for avionics that must detect the stall



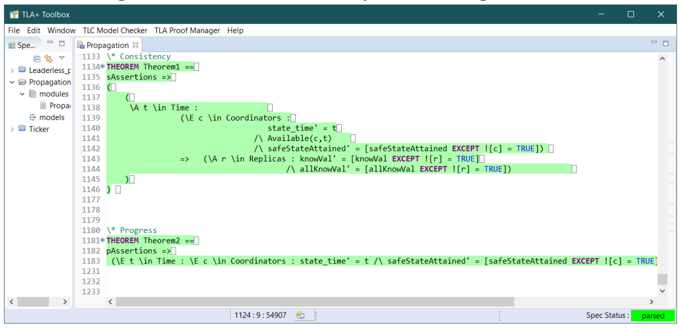
```
Theorem: (if (forall ?T:Synod.TP
               (exists ?i:N
                 (exists ?p:Synod.Actor
                   (exists ?b:Synod.Ballot
                     (exists ?Q:Synod.Quorum
                       (and (Synod.prp ?p:Synod.Actor)
                            (and (Synod.P1+P2-True ?b:Synod.Ballot)
                                 (and (Synod.nonfaulty ?p:Synod.Actor)
                                      (forall ?a:Synod.Actor
                                        (if (Synod.inQ ?a:Synod.Actor ?Q:Synod.Quorum)
                                            (and (Synod.nonfaulty ?a:Synod.Actor)
                                                 (Synod.ready-to (Synod.rho ?T:Synod.TP ?i:N)
                                                                 ?p:Synod.Actor
                                                                 (Synod.send ?p:Synod.Actor
                                                                             (Synod.consM ?p:Synod.Acto
r ?a:Synod.Actor Synod.1a ?b:Synod.Ballot Synod.nilv))))))))))))))
             (forall ?T:Synod.TP
               (exists ?p:Synod.Actor
                 (exists ?i:N
                   (exists ?b:Synod.Ballot
                     (Synod.learn ?p:Synod.Actor
                                  (Synod.config (Synod.rho ?T:Synod.TP ?i:N))
                                  ?b:Synod.Ballot))))))

Module Synod extended.

Ready...

>
```

Fig. 7: Verification of Synod using Athena.



```
\* Consistency
THEOREM Theorem1 ==
sAssertions =>
(
    (
     \A t \in Time :
                 (\E c \in Coordinators :
                                state_time' = t
                             /\ Available(c,t)
                             /\ safeStateAttained' = [safeStateAttained EXCEPT ![c] = TRUE])
                 =>   (\A r \in Replicas : knowVal' = [knowVal EXCEPT ![r] = TRUE]
                                     /\ allKnowVal' = [allKnowVal EXCEPT ![r] = TRUE])
    )
)



\* Progress
THEOREM Theorem2 ==
pAssertions =>
 (\E t \in Time : \E c \in Coordinators : state_time' = t /\ safeStateAttained' = [safeStateAttained EXCEPT ![c] = TRUE]
```

Fig. 8: Verification of TAP using TLAPS.

state of an aircraft, it will be useful to have a guarantee that states with a high probability that the system will correctly detect the stall/no stall state over a specified subset of operating conditions [30]. This section presents some ways to provide such guarantees of probabilistic properties that can be useful at runtime.

A. Models for Reasoning

Probabilistic properties can be guaranteed for systems by modeling the systems stochastically. Stochastic estimation follows two methods — *data-driven models* that are based on runtime observations or *theoretical models* based on analytical principles. Both data-driven and theoretical methods utilize measurements, physical knowledge, and context [31]. Below, we present some of our work on using such models for reasoning about aerospace systems.

1) Data-Driven Models: It is possible to observe the operating conditions of aerospace systems by collecting data from sensors at runtime or through simulation [32]. The data can be used to infer important statistical properties and create data-driven models about the operating conditions. We have used data-driven models for stall detection in fixed-wing aircraft [33] and for rotor fault detection in hexacopters [34].

a) Fixed-Wing Aircraft Stall Detection: A data-driven method for stall detection uses response signals recorded from piezoelectric sensors placed on the wings of an aircraft to detect the stall states [35]. Given dynamic noise-corrupted data collected from a sample of the admissible flight states; each state can be characterized by a specific *airspeed* and *angle of attack* (AoA)[2] and kept

[2]The angle between the chord of the wing and the relative airflow.

constant for the duration of the data collection. Hence, it is possible to develop appropriate methods capable of monitoring and detecting aerodynamic stalls (Fig. 9) without the use of pitot-static[3], angle of attack, or gyroscopic sensor information [36]. We have developed a mathematical model for stall detection which is based upon experimental wind tunnel data that was obtained by controlling the AoA and airspeed configuration of a model wing and recording the mean and variance of signal energy. Specific angle of attack/airspeed configurations correlate with aeroelastic properties[4], allowing certain signal energy distributions to be associated with stall/no-stall conditions [37].

Our collective-Gaussian stall model is defined as: *Given the sample space* $\mathbb{R}$*, a collective-Gaussian stall model is the collective-probability model for stall classification:* $M_{stall} = \langle \Theta_{states}, \{N(\mu_\theta, \Sigma_\theta)\}, P_{states}, L_{stall}, C_{stall} \rangle$, where Θ_{states} is the set of air flight configurations for which there is data, $\{N(\mu_\theta, \Sigma_\theta)\}$ is a multivariate-normally distributed random variable for the air flight configuration $\theta \in \Theta_{\text{states}}$, P_{states} is the probability density function that determines the probability for an air flight configuration $\theta \in \Theta_{\text{states}}$ to occur, $L_{\text{stall}} = \texttt{stall}, \texttt{no-stall}$; and $C_{\text{stall}} : \Theta \to L_{\text{stall}}$ is a tag function for each flight state.

b) Hexacopter Rotor Fault Detection: We adopted a model of a rigid-body regular hexacopter (Fig. 10) that uses a summation of forces and moments to calculate accelerations [38]. This model allows for the simulation of abrupt rotor failures by ignoring the failed rotors' inflow states and setting the output rotor forces and moments to zero. A proportional-integral-derivative (PID)-based feedback controller was implemented on the nonlinear model to stabilize the aircraft altitude and attitudes, as well as track desired trajectories. We developed a velocity-based control design which can perform well even in the event of rotor failures, with no adaptation in the control laws themselves [38]. We then implemented a continuous Dryden wind turbulence model to simulate realistic flight conditions in our experiments.

The Dryden model provided the main source of simulated data for 5 m/s forward speed under light, moderate, and severe levels of turbulence. Observations were then made from the roll/pitch/yaw signals after simulating abrupt rotor failures [39]. In the case of front rotor failure, the hexacopter pitched down without any substantial change in roll angle, as the loss of rotor 1 thrust did not significantly affect the aircraft roll equilibrium. However, in the case of side rotor failure, both the pitch and roll attitudes changed and the roll attitude compensation was observed to be underdamped[5]. The deviation of roll attitude in rotor 2 failure was reversed for rotor 6 failure, because of unbalanced roll moment in rightward and leftward directions, respectively. The heading of the aircraft was observed to deviate in different directions with the failure of the front rotor compared to the side rotor. This was due to the different rotor spin directions of front and side rotors, and consequently the direction of the hub torque generated by each rotor. We developed a knowledge-based fault detection algorithm that uses the knowledge of the hexacopter configuration along with principles of force and moment equilibrium to predict how the aircraft roll, pitch, and yaw attitudes will behave under instantaneous loss of thrust and consequent moment imbalance in an event of rotor failure. Therefore, the decision-making follows the deviation of signals from their 99% confidence intervals under abrupt failures of rotors 1, 2, and 6.

[3]Sensors used to detect ambient air pressure in an aircraft.

[4]Structural effects on aerodynamic forces.

[5]"Underdamped" means that the controller compensated for the disturbance caused by rotor failure by allowing the roll signal to complete a few oscillations before settling into a steady-state value.

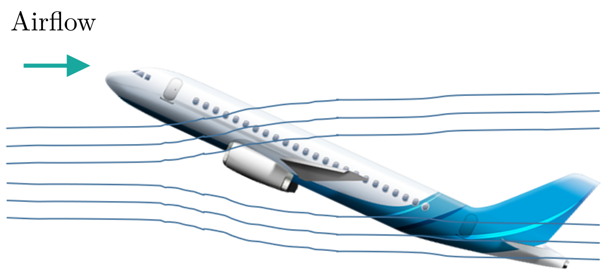


Fig. 9: Stall in a fixed-wing aircraft.

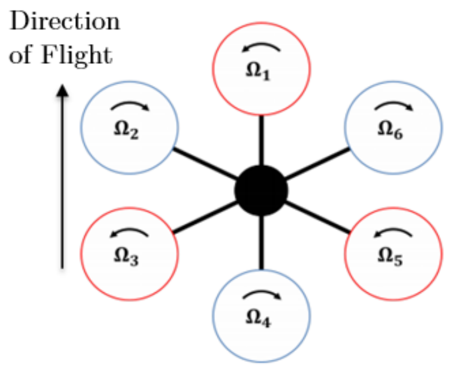


Fig. 10: Schematic diagram of a hexacopter.

2) Theoretical Models: Theoretical models of stochastic systems are based on well-established analytical and statistical results about the behavior of the systems. To develop guarantees that are useful and practical, the theory used for such reasoning must be appropriate for the system analyzed. We have used theoretical models for reasoning about probabilistic progress in distributed protocols [40].

The total time that is required for a distributed protocol to make progress is dependent on three factors : (1) message transmission delays, (2) message processing delays, and the (3) the number of messages involved. In asynchronous settings, message delays are unbounded, making it impossible to provide deterministic bounds on the total time that may be required for progress. In the IoP, aircraft must be able to communicate their own information and relay received information from other aircraft using a *multi-hop ad hoc* approach [41] (Fig. 11). Moreover, an aircraft can be viewed as a single *server* where messages arrive, wait for some time until they are processed, and take some time to be processed, which is consistent with *queueing theory* [42] for reasoning about message processing delays (Fig. 12). Among many possible theoretical models present in the literature, we have used the *Multicopy Two-Hop Relay* (MTR) protocol [43] and the *M/M/1 queue system* [42] to model message transmission and processing delays respectively in *vehicular ad hoc networks* (VANET) [44] like the IoP.

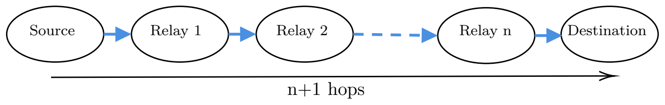


Fig. 11: Message transmission via relaying.

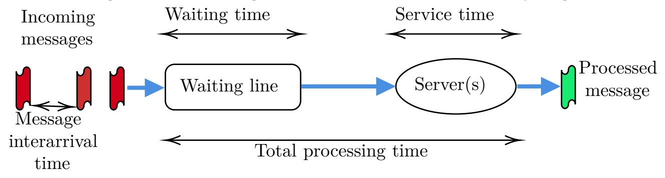


Fig. 12: A typical queuing system.

a) The MTR Protocol for Inter-Aircraft Communication Delays: In the MTR protocol, there are a set of $M+1$ mobile nodes whose trajectories are independent and identically distributed (i.i.d.). A *source* node can directly transfer a message to a *destination* node if the destination is within its transmission range, or, it can transmit copies of the message to one or more *relay* nodes. A relay node can transmit a copy of a message directly to the destination node if the two are within the transmission range of the relay node. A relay node, however, cannot transmit a copy of a message to another relay node. Each message, therefore, makes a maximum of two hops — it is either transmitted directly from the source to the destination, or it is transmitted through one intermediate relay node. The inter-meeting times of all pairs of nodes, *i.e.*, when they are reachable from each other, are i.i.d. and are *exponentially distributed* with the rate parameter λ_{MTR}. Each untransmitted copy of a message has a *time-to-live* (TTL) after which a relay drops it.

By using the specifications of MTR, the following probabilistic bound on the time T_{D_m} taken to deliver a message m using MTR can be proven:

$$P\left(T_{D_m} \leq t\right) = 1 - e^{-\lambda_{MTR}Mt} \tag{1}$$

where T_{D_m} is exponentially distributed with a rate parameter $\lambda_{D_m} = \lambda_{MTR}M$.

b) The M/M/1 Queue System for Aircraft Message Processing Delays: The *M/M/1* queue system [42] enables deriving elegant analytical properties for computing the total time T_{P_m} required for a message m to be processed. T_{P_m} includes the time spent in the queue and the time spent in actually processing the message. Some important analytical properties of the M/M/1 queue are: it is managed by a single server; T_{P_m} is exponentially distributed [45]; the *interarrival times* of messages in the queue are exponentially distributed with a rate parameter λ_a; the *service time* of messages is exponentially distributed with a mean $1/\mu_s$, where μ_s is the rate parameter; and the number of message arrivals in an interval of length τ follows a *Poisson distribution* [46] with a parameter $\lambda_a\tau$. The rate parameter of T_{P_m} can then be obtained as:

$$\lambda_{P_m} = \mu_s - \lambda_a \tag{2}$$

In [40], we have used Athena to develop mechanically verified formal proofs of the following well-known results about the MTR protocol and the M/M/1 queue system:

Theorem 6. *(The Rate Parameter for T_{D_m}) T_{D_m} is exponentially distributed with the rate parameter $\lambda_{D_m} = \lambda_{MTR}M$.*

Theorem 7. *(The Rate Parameter for T_{P_m}) T_{P_m} is exponentially distributed with the rate parameter $\lambda_{P_m} = \mu_s - \lambda_a$.*

The machine-checked proofs of the above theorems can be used as building blocks and "plugged-in" as required for developing higher-level machine-checked guarantees of timely progress for distributed protocols that can be useful for safety-critical UAM applications.

The DDDAS paradigm [16] advocates for a synergetic interaction between purely theoretical and data-driven models. A theoretical model can be enhanced dynamically by collecting runtime data, making it more accurate and computationally less expensive. Similarly, theoretical principles can help inform where it pays off best to collect data.

B. Formal Proof Libraries

Aerospace systems are complex and consist of intricate relationships between various types of sub-systems that work in tandem. Therefore, reasoning about the high-level properties of such systems involves holistic consideration of the various domain-specific aspects involved. Formally proving the correctness of a high-level property using an interactive proof assistant like Athena requires access to formal constructs that are sufficiently expressive to correctly specify such aspects. Note, for formal reasoning about the statistical properties of stochastic aerospace systems, it is necessary to have access to definitions from across mathematics as represented in Fig. 13.

Developing expressive formal constructs in a machine-readable language is a challenging task since it requires: (1) domain knowledge of all aspects of the systems that need to be specified, (2) knowledge of formal logic and reasoning techniques, and (3) proficiency in the language in which the constructs are to be specified. If computer scientists and aerospace engineers have to develop the required formalizations from the ground up every time a new high-level property for a complex system needs to be verified, then the task of verification can become extremely expensive. To address this, we propose the development of domain-specific reusable libraries of formal constructs that can be readily used for reasoning about such high-level

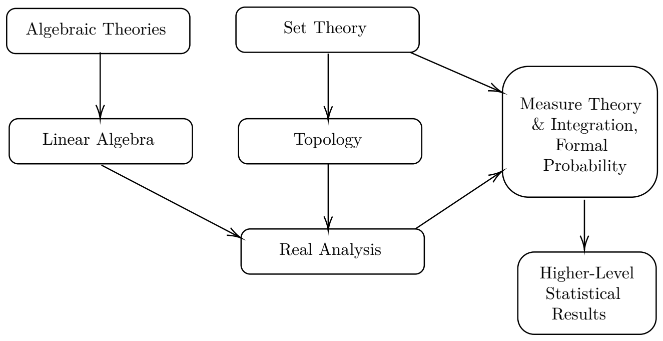

Algebraic Theories
Set Theory
Linear Algebra
Topology
Measure Theory & Integration, Formal Probability
Real Analysis
Higher-Level Statistical Results

Fig. 13: Theories required to reason about high-level statistical properties of stochastic systems [37].

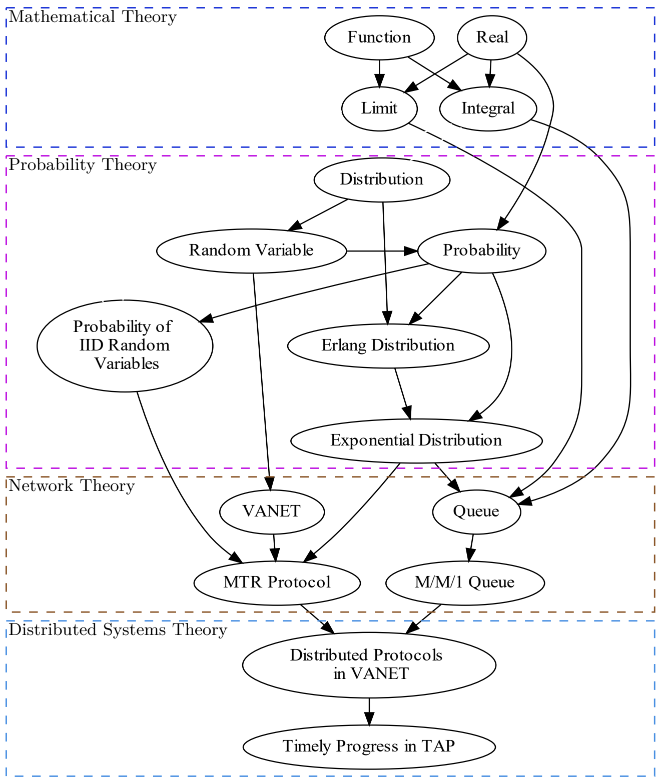


Fig. 14: Hierarchy of reusable theories in Version 0.1 of the ASSURE Athena library.

properties, just like code libraries for regular programming languages. Such libraries can be fairly complicated to develop because of the reasons mentioned earlier, but once reusable theories have been created, they can simply be "plugged-in" for verifying dependent high-level properties.

Mechanically verifying timely progress properties of distributed protocols using theories of the MTR protocol and the M/M/1 queue system requires formal specifications to express constructs such as networks, mobility models, communication models, failure-models, inter-agent interactions, queues, and agent behaviors. Most of these constructs can be used for reasoning about different types of distributed airborne communication protocols as well as other aerospace applications such as stochastic state estimation (*e.g.*, stall and fault detection avionics systems). Therefore, we have developed an open-source proof library in Athena that can be used for reasoning about the probabilistic properties of stochastic aerospace systems [40][6]. As of Version 0.1, our Athena library has specialized theories from different domains to reason about distributed protocols implemented over ad hoc airborne networks. There are theories from four main domains: general mathematics, probability, network theory, and distributed systems. Within each domain, there are theories from various related sub-domains. Fig. 14 depicts the logical hierarchy of version 0.1 of the ASSURE Athena library and the dependencies between the various modules representing the different logical domains and sub-domains. Using the library, we have mechanically verified the probabilistic properties of message delays in the IoP (*e.g.*, Fig. 15).

```
conclude THEOREM-mean-T-value
...
let{
    ...
    mu_s := (Dist.ratePar (Random.pdf (srvcTm Q))
    );
    lambda_a := (Dist.ratePar (Random.pdf (
    cstArRat Q)));
    N_d :=  (mu_s - lambda_a);
    ...
}
(!chain [ T
    = (N * (1.0 / L)) [T=N-x-inv-L]
    = ((L / N_d) * (1.0 / L)) [N=L-by-N_d]
    = (1.0 / N_d) [conn-2-a-by-d-x-b-by-a]
    ])
```

Fig. 15: A snippet from the Athena proof of Eq. 2.

The formal specifications in our Athena library are reusable in different contexts so that further expansion of the library can be aided by using the existing specifications as building blocks rather than requiring them to be redeveloped from scratch. In our experience, efficiently designing reusable formal structures to express interconnected theories requires several iterations where the specifications need to be improved to be more general as new contexts for application are identified. Another challenge is the meaningful modularization of the theories to make it easy to import them independently in different contexts. We are working on extending the library by adding more mechanically verified theories to reason about state estimation applications such as stall and fault detection.

C. *Correctness Envelopes*

Correctness properties of a system can be formally verified using machine-checked proofs that often depend on certain logical preconditions that must be satisfied. Some of these preconditions are data-driven and can be quantified and measured at runtime using data collected from sensors. Such data-driven preconditions allow specification of *correctness envelopes* that represent distinct subsets of the operational state space where some correctness

[6]The library can be found at https://wcl.cs.rpi.edu/assure

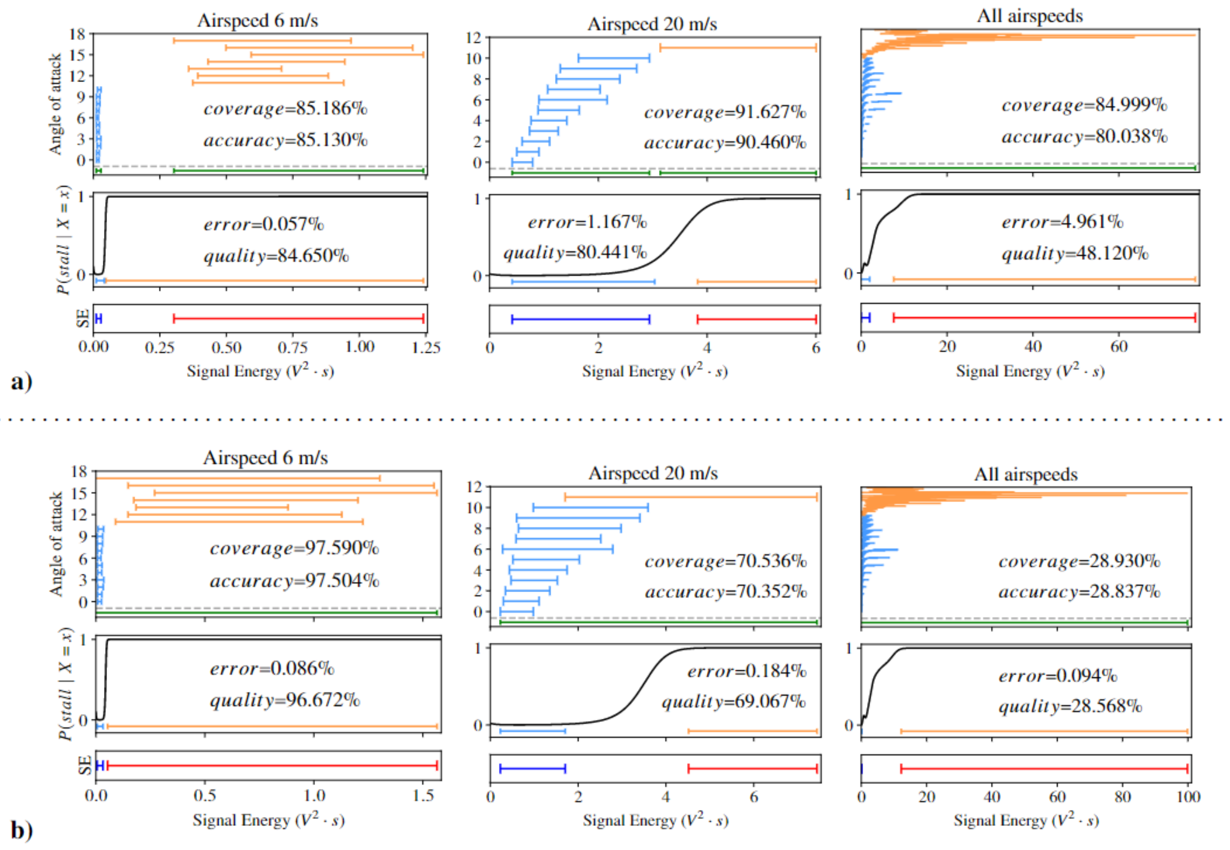


Fig. 16: Safety envelopes (bottom row) as the intersection of z-predictability (top row) and τ-confidence (middle row) for airspeed of $6m/s$, $20m/s$ and all flight states. **a)** $z = 1$, $\tau = 80\%$ **b)** $z = 2$, $\tau = 99\%$. In the bottom row, blue represents no stall, red represents stall, and white represents the area outside the envelope [33].

properties of the system can be irrefutably guaranteed. These envelopes are defined by parameters that can depend on runtime measurements. For specific properties, the envelopes are determined by different types of data, and at any given time, the system may be well within the boundaries of one envelope and outside another envelope. We present two types of correctness envelopes relevant to high-level properties of interest for aerospace systems — *safety envelopes*[7].

1) Safety Envelopes: A formal *safety envelope*, analogous to a flight envelope, is a computable subset of the operational state space that describes the constraints on the operating conditions under which a correctness guarantee is valid [37]. Such a subset can involve both continuous (*e.g.*, membership of sensor data in some interval of the state space) and non-continuous (*e.g.*, normality test of the sample history of a sensor) constraints.

Safety envelopes for stall awareness (Fig. 16) are defined using the intersection of *z-predictability*, capturing the likelihood of the signal energy x being consistent with the M_{stall} data-driven model, and τ*-confidence*, capturing the likelihood of accurate stall classification, as follows:

- ***z-predictability:*** Given a collective-Gaussian stall model M_{stall} (see Section III-A1a) and a parameter space $z \in \mathbb{R}^{+}$, *z-predictability* corresponds to the statement: $S_{z-pred}\left(M_{\text{stall}}, z, \mathbf{x}\right) = \exists \theta \in \Theta_{\text{states}} : D_M\left(\mu_\theta, \mathbf{\Sigma}_\theta, \mathbf{x}\right) < z$ where $D_M\left(\mu_\theta, \mathbf{\Sigma}_\theta, \mathbf{x}\right)$ is the *Mahalanobis distance* $\sqrt{\left(\mu_\theta - \mathbf{x}\right)^{\intercal} \Sigma_\theta^{-1} \left(\mu_\theta - \mathbf{x}\right)}$.
- ***τ-confidence:*** Given a collective-Gaussian stall model M_{stall} and a parameter space $\tau = (0.5, 1]$, the classification function for M_{stall} is given by:

$$C_{cond}\left(M_{\text{stall}}, \tau, x\right) = \begin{cases} \text{stall} & P(\text{ stall } = \text{true} \mid X = x) \geq \tau \\ \text{nostall} & P(\text{ stall } = \text{ false } \mid X = x) \geq \tau \\ \text{uncertain} & \text{in any other case} \end{cases}$$

where the conditional probability of `stall` given x follows Bayes Theorem:

$$P(\texttt{stall = true} \mid X = x) = \frac{P(X = x \mid \texttt{stall = true}) P(\texttt{stall = true})}{P(X = x)}$$

A classification function $C_{cond}\left(M_{\text{stall}}, \tau, x\right) = k$ is τ-*confident iff* $k \neq$ `uncertain`.

Two mechanically verified Agda theorems about safety envelopes for stall classification are given below:

Theorem 8. *(z-predictability within Safety Envelope)*

[7]In this context, "safety" means that nothing bad ever happens, while "progress" means that something desirable happens.

```
program Rotor_check_threshold;
  inputs
   roll,pitch,yaw (t) using closest(t);
  constants
   R_LOW = -0.0153;
   R_HIGH = 0.006;
   P_LOW = -0.0884;
   P_HIGH = -0.0703;
   Y_LOW = -0.0040;
   Y_HIGH = 0.0022;
  outputs
   roll,pitch,yaw,mode at every 0.1 sec;
  errors
   e1: roll;
   e2: pitch;
   e3: yaw;
  modes
   m0: e2 in [P_LOW .. P_HIGH]
       or (e2 in ( .. P_LOW) and e3 in [Y_LOW ..
    Y_HIGH])
     or (e2 in ( .. P_LOW) and e3 in (Y_HIGH .. )
     and e1 in [R_LOW .. R_HIGH]) "Normal";
   m1: e2 in ( .. P_LOW)
       and e3 in ( .. Y_LOW) "Rotor 1 failure";
   m2: e1 in ( .. R_LOW) and e2 in ( .. P_LOW)
       and e3 in ( Y_HIGH .. ) "Rotor 2 failure";
   m3: e1 in (R_HIGH .. ) and e2 in ( .. P_LOW)
       and e3 in (Y_HIGH .. ) "Rotor 6 failure";
end;
```

Fig. 17: A sentinel written in PILOTS.

If an energy signal x belongs to safety envelope $SE_{cond}(M_{stall}, z, x)$, then $\exists\ \theta \in \Theta_{states}$: $\sqrt{(\mu_\theta - \mathbf{x})^\intercal\, \Sigma_\theta^{-1}\, (\mu_\theta - \mathbf{x})} < z$.

Theorem 9. *(τ-confidence within Safety Envelope) If an energy signal x belongs to safety envelope $SE_{cond}(M_{stall}, z, x)$, then* $C_{cond}\,(M_{stall}\,, \tau, x) \neq$ `uncertain`.

2) Progress Envelopes: A formal progress envelope for a distributed algorithm is a computable subset of the system state space where the formal proof of a progress property holds [47]. The envelope is defined by a set of logical constraints parameterized by the system's operational conditions. To illustrate formal progress envelopes, let us consider a hypothetical distributed protocol deployed in the IoP. Assume that there is a progress guarantee that the protocol will make progress in under 8 ms with 98% probability if the message delays are exponentially distributed with a rate parameter λ. Now, at runtime, the message delays are observed over two different time intervals — $[t_0, t_1]$ and $[t_1, t_2]$. It is seen that the message delays do not follow exponential distribution in the first interval but follow an exponential distribution with rate parameter λ in the second interval. Therefore, we say that the first interval is *outside* the progress envelope while the second interval is *inside* the envelope.

D. Runtime Verification of Stochastic Aerospace Systems

We present some data-driven techniques to extend formal proofs beyond the pre-deployment stages so that mathematically rigorous guarantees of probabilistic properties can be provided at runtime.

1) Parameterized Proofs: Mathematically rigorous formal proofs of correctness can be developed manually in the pre-deployment stages of aerospace systems. In order to make these proofs useful at runtime, it is possible to design them as functions of the correctness envelope parameters that can be quantified and measured at runtime [47],*e.g.,* instead of a static proof that states: "*The probability that round-trip message transmission latency will be at most 0.9 seconds is 99%*", a parameterized proof would state: "*If the one-way message transmission and processing delays follow Gaussian distributions $\mathcal{N}(\mu_X, \sigma_X^2)$ and $\mathcal{N}(\mu_Y, \sigma_Y^2)$, then the probability that the round-trip message latency will be at most t seconds is $\boldsymbol{G}(\mu_X, \sigma_X, \mu_Y, \sigma_Y, t)$*", where $\boldsymbol{G}$ is the cumulative distribution function of the combined Gaussian distribution $\mathcal{N}(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$. Modeling the statistical reasoning allows the proofs to be augmented with dynamic parameters at runtime in order to generate properties that are both dynamically useful and formally rigorous. It should, however, be noted, that proofs that are not designed as functions of only runtime observable parameters, cannot be classified as parameterized proofs as they cannot be instantiated using dynamic observations.

2) Correctness Sentinels: A *correctness sentinel* is a runtime-accessible program that can monitor data-streams to detect if the data conforms to correctness envelope constraints [33], [37]. Like any software for safety-critical aerospace systems, the sentinels must be verified to ensure that they can monitor required data correctly. We present two approaches for developing verified sentinels that can be used for monitoring correctness properties at runtime.

a) Sentinels Developed in Domain-Specific Programming Languages: Using a domain-specific programming language, we can design and verify sentinel software. The high-level and declarative nature of domain-specific languages can potentially enable the generation of formal specifications for verification from the code. In [34], we have verified the correctness of a data streaming application developed in the **P**rogramm**I**ng **L**anguage for spati**O**-**T**emporal data **S**treaming (PILOTS). PILOTS is a high-level declarative programming language for the development of applications for analyzing spatio-temporal data streams [48]. We designed a PILOTS program called `Rotor_check_threshold` (Fig. 17) for the hexacopter rotor fault detection application (described in Section III-A1b). `Rotor_check_threshold` takes the pitch, roll, and yaw attitudes as input streams of data and outputs the same along with the estimated mode of the system every 0.1 seconds. These three attitudes are analyzed with error functions that capture patterns in the input streams occurring when individual rotors fail. `Rotor_check_threshold` is a sentinel because it can quickly analyze incoming data streams to check if the data corresponds to individual rotor failure conditions. For the verification of the `Rotor_check_threshold` program, we used Dafny [13], a verification tool based on Hoare logic [12] that uses the Z3 SMT solver [49]. Appropriate pre-conditions, post-conditions, and invariants were identified corresponding to the correctness conditions of the constraints in the state estimation

```
method inductiveProof(start: int, end : int,
    dataStream : array<Error>, A_LOW: real,
    A_HIGH : real)
    returns (modeStream : array<int>)
    requires end >= start && start >= 0
    requires dataStream.Length > 0 ==> dataStream
    .Length - 1 >= end
    requires dataStream.Length == 0 ==>
    dataStream.Length >= end
    ensures modeStream.Length == dataStream.
    Length
    ensures SystemCheck(start, end, dataStream,
    modeStream, A_LOW, A_HIGH)
    {
        modeStream := new int[dataStream.Length];
        if (dataStream.Length == 0)
        {
            assert SystemCheck(0, 0, dataStream,
    modeStream, A_LOW, A_HIGH); }
        else
        {
            var idx := start;
            while(idx < end)
            invariant idx <= end
            invariant SystemCheck(start, idx,
    dataStream, modeStream, A_LOW, A_HIGH)
            decreases end - idx
            {
                var m := ModeAnalyzer(dataStream[
    idx], A_LOW, A_HIGH); modeStream[idx] := m;
                idx := idx + 1; }}}
```

Fig. 18: Dafny specification of a mode estimation function.

model, and were generated as a formal specification. A *PILOTStoDafny* code generator was created to generate Dafny specifications (Fig. 18) from the PILOTS code, and then Dafny was used to prove the correctness of the code with respect to the specifications.

b) Sentinels Automatically Generated from Theory: Another approach is to generate sentinels from the theory. Formal specifications of the correctness properties that a sentinel must monitor are created, then the sentinel's code is generated directly from the specifications. Two advantages are: firstly, the specifications of the sentinels are derived from the theory of the correctness envelopes, ensuring that the sentinels are correct by construction; and secondly, the sentinel specifications are amenable to theorem proving techniques. In [33], we created a sentinel for the stall detection application described in Section III-A1a directly from specifications written in the Agda proof system (Fig. 19). Agda can be used to generate Haskell code that can be executed and tested. To do this, a wrapper was written around the Agda code to pass data from the standard input. The resulting binary can process a continuous stream of data and output to the standard output a stream of booleans representing membership in the safety envelope. The correctness of the Haskell code follows from the correctness of the Agda specifications.

3) Runtime Verification using DDDAS: DDDAS allows a system to dynamically incorporate new data to enhance an existing model using a *feedback loop* [16]. DDDAS with parameterized proofs generates and updates formally rigorous correctness guarantees at runtime. Within the DDDAS paradigm, a *formal feedback loop* consists of the following (Fig. 20):

- The *proof refinement* component receives a parameterized proof with a set of initial parameters and runtime inputs from the sentinels. If possible, it refines the proofs with the runtime parameters and provides new envelopes.
- The *model refinement* component receives new envelopes from the proof refinement component and an initial system model. It updates the system model according to the latest envelopes.
- The *sentinels* analyze the runtime operating conditions of the system against the latest envelopes. If the conditions do not conform to the envelopes, they send the runtime parameters to the proof refinement component and inform the live system about guarantees that hold in runtime.
- The *live system* uses the updated system model from the model refinement component.

The DDDAS feedback loop dynamically updates the formal proofs with parameters that reflect the runtime operating conditions of a system, thereby allowing the development of highly adaptive provably correct applications that can adapt to properties that hold at runtime. In the case of dynamic systems, this feedback loop can be used by developing suites of model-specific parameterized proofs that can change dynamically depending on the model that is being used at runtime. If there is a set of models $\mathbb{M}$, then for each model M in $\mathbb{M}$, there can be a set of parameterized proofs $\mathbb{S}(M)$. If the DDDAS system updates its model to any model M in $\mathbb{M}$ at runtime, the corresponding set of proofs $\mathbb{S}(M)$ can be used to provide guarantees. This approach, however, has its limitations as only a finite number of models and corresponding sets of proofs can be developed in advance. If the system chooses a model $M' \notin \mathbb{M}$ at runtime, then no guarantees can be generated.

To showcase a simple application of the formal DDDAS feedback loop, let us consider an example[8] where there is a machine-verified parameterized property — *"For a random variable X following a Gaussian distribution f_X, $P(X \le v) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$, where $z = \frac{v-\mu_X}{\sigma_X}$ and μ_X and σ_X are the mean and standard deviation of f_X."* Now,

```
theorem1← : ∀(M z x)
    → z-predictable M z x ≡ ⟨x, true⟩
    → Any (λ{⟨⟨α, ν⟩, ⟨nd, p⟩⟩ → x ∈ pi nd z})(Model.fM M)
theorem1← : M z x res ≡ x, true = any-map(proj₁ ∘ proj₂)(
    follows-def← M z x res ≡ x, true)
theorem1→ : ∀(M z x)
    → Any (λ{⟨⟨α, ν⟩, ⟨nd, p⟩⟩ → x ∈ pi nd z})(Model.fM M)
    → z-predictable M z x ≡ ⟨x, true⟩
theorem1→ : M z x proofAny = follows-def→ M z x(
    any-map-rev(proj₁ ∘ proj₂)proofAny)
```

Fig. 19: Agda specifications from which a Haskell sentinel was generated.

[8]Video of a simulation of this example can be found at https://wcl.cs.rpi.edu/assure

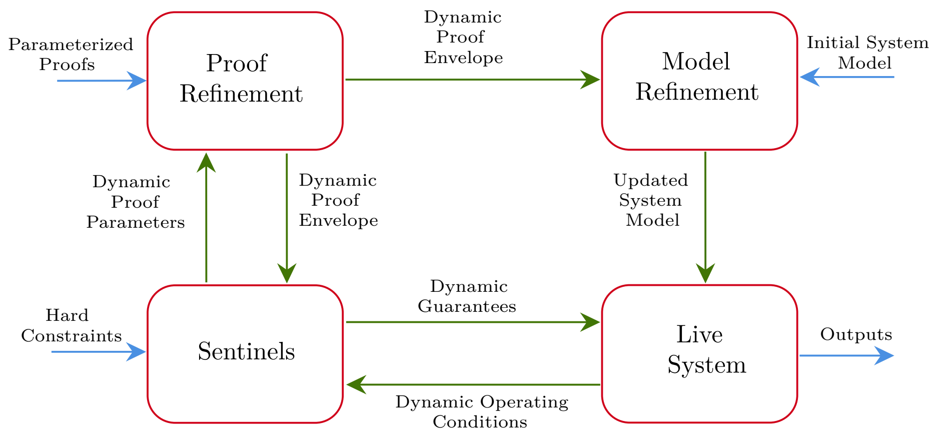


Fig. 20: Formal DDDAS sentinel feedback loop.

let us assume that the live system uses the transmission delay T_{D_m} for a single message m as a model to compute the round-trip delay $2 \times T_{D_m}$. Initially, it is assumed that T_{D_m} is normally distributed with $\mu_{T_{D_m}} = 5$ ms and $\sigma_{T_{D_m}} = 1$. The proof refinement component computes that $P(X \leq 10) = 99.99997\%$ and $P(X \leq 15) = 99.99999\%$. As $P(X \leq 10) > 99\%$, the model refinement component selects $T_{D_m} = 10$ ms and the live system computes a round-trip delay of 20 ms. Now, at runtime, the sentinel monitors the actual values of T_{D_m} and sees that $\sigma_{T_{D_m}} \approx 3.1$. This causes the proof refinement component to compute new values $P(X \leq 10) = 94.41874\%$ and $P(X \leq 15) = 99.92945\%$. Now, the model refinement component selects $T_{D_m} = 15$ ms and the live system computes a round-trip delay of 30 ms.

IV. Discussion

Aerospace systems are complex and involve intricate interactions between various types of sub-systems that allow for superior performance and adaptability. The complex nature of interactions between the sub-systems increases the possibility of malfunctions that can be catastrophic. The techniques we have presented in this article allow rigorous verification of the safety-critical correctness guarantees of such systems using formal logic.

Real-life operating conditions of aerospace systems are rife with uncertainties that cannot be predicted or modeled accurately. Therefore, the probabilistic approach of theorem proving presented in this article can be beneficial as it allows for the development of formal guarantees in the face of such uncertainties. This work creates a good foundation for extending formal theorem proving techniques to more complex statistical theories, such as the *Cramer-Rao Lower Bound* (CRLB), that can be used for statistically analyzing the quality of data-driven models, or for analyzing more sophisticated parametric and non-parametric stochastic state estimation techniques.

Another important aspect of the verification of aerospace systems is the runtime verification of properties with respect to the dynamically changing operating conditions. Runtime verification is useful because even probabilistic models often fall short of accurately modeling the uncertain operating conditions of aerospace systems. Therefore, it is important to have the ability to monitor the formal properties at runtime so that the applications can be aware of what guarantees are actually valid at a given point in time during deployment.

V. Conclusion and Future Work

This article presented ASSURE (***A****nalysis of* ***S****afety-Critical* ***S****ystems* ***U****sing Formal Methods-Based* ***R****untime* ***E****valuation*) — a collection of formal methods-based techniques for the verification of deterministic and non-deterministic properties of safety-critical aerospace systems. ASSURE supports techniques for the verification of both centralized aerospace applications, such as stall detection, and decentralized applications, such as autonomous multi-aircraft coordination. Using the ASSURE approach, formal proofs for the runtime verification of dynamic systems, whose operating conditions cannot be modeled accurately, can be developed. Runtime verification enables monitoring system behavior and instantiating parameterized proofs accordingly at deployment time.

Version 0.1 of ASSUREs Athena library only supports deterministic and probabilistic reasoning about airborne distributed protocols. Hence, one potential direction of future work would be to expand the library with new theories to support reasoning about stochastic state-awareness and failure modeling for aerospace systems. Another direction of future work would be to design an efficient pipeline to generate sentinels for such applications directly from the specifications in the Athena library. Finally, an implementation of the DDDAS formal feedback loop for aerospace applications will allow the safer development of future networked drone and eVTOL hardware systems.

References

[1] P. Johnston and R. Harris, "The Boeing 737 MAX saga: Lessons for software organizations," *Soft. Qual. Prof.*, vol. 21, no. 3, pp. 4–12, 2019.
[2] P. Brooker, "The Uberlingen accident: Macro-level safety lessons," *Saf. Sci.*, vol. 46, no. 10, pp. 1483–1508, 2008.

[3] NTSB News Releases. (2021, May) Investigative update: Wednesday's mid-air collision near Denver. Accessed: Nov. 2021. [Online]. Available: https://www.ntsb.gov/news/press-releases/Pages/NR20210513b.aspx

[4] R. Drees and N. Young, "Role of BIT in support system maintenance and availability," *IEEE Aero. and Elect. Syst. Mag.*, vol. 19, no. 8, pp. 3–7, 2004.

[5] S. Imai, E. Blasch, A. Galli, W. Zhu, F. Lee, and C. A. Varela, "Airplane flight safety using error-tolerant data stream processing," *IEEE Aero. and Elect. Syst. Mag.*, vol. 32, no. 4, pp. 4–17, 2017.

[6] J.-F. Monin, *Understanding formal methods.* Springer Science & Business Media, 2012.

[7] K. Arkoudas and D. Musser, *Fundamental Proof Methods in Computer Science: A Computer-Based Approach.* Cambridge, MA: MIT Press, 2017.

[8] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, "Verifying safety properties with the TLA+ Proof System," in *Int. Joint Conf. on Auto. Reas.* Springer, 2010, pp. 142–148.

[9] M. Ouimet and K. Lundqvist, "Formal software verification: Model checking and theorem proving," *Embedded Systems Laboratory Technical Report ESL-TIK-00214*, 2007.

[10] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem *et al.*, *Handbook of Model Checking.* Springer, 2018, vol. 10.

[11] Y. Yu, P. Manolios, and L. Lamport, "Model checking TLA+ specifications," in *Adv. Res. Work. Conf. on Corr. Hard. Des. and Ver. Meth.* Springer, 1999, pp. 54–66.

[12] C. A. R. Hoare, "An axiomatic basis for computer programming," *Comm. of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[13] K. R. M. Leino, "Dafny: An automatic program verifier for functional correctness," in *Int. Conf. on Logic for Prog. Art. Intel. and Reas.* Springer, 2010, pp. 348–370.

[14] L. Lamport, "The Temporal Logic of Actions," *ACM Trans. on Prog. Lang. and Syst. (TOPLAS)*, vol. 16, no. 3, pp. 872–923, 1994.

[15] P. V. D. Walt and W. Swierstra, "Engineering proof by reflection in Agda," in *Symp. on Impl. and App. of Funct. Lang.* Springer, 2012, pp. 157–173.

[16] F. Darema, "Dynamic Data-Driven Application Systems: A new paradigm for application simulations and measurements," in *Int. Conf. on Comp. Sci. (ICCS).* Springer, 2004, pp. 662–669.

[17] S. Paul, S. Patterson, and C. A. Varela, "Conflict-aware flight planning for avoiding near mid-air collisions," in *The 38th AIAA/IEEE Dig. Avio. Syst. Conf. (DASC)*, San Diego, CA, Sep. 2019, pp. 1–10.

[18] S. M. Lee, C. Park, M. A. Johnson, and E. R. Mueller, "Investigating effects of well clear definitions on UAS sense-and-avoid operations in enroute and transition airspace," in *2013 Avia. Tech., Int., and Op. Conf.*, 2013, p. 4308.

[19] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems.* Cambridge, MA: The MIT Press, 1986.

[20] C. Hewitt, "Viewing control structures as patterns of passing messages," *Art. Intel.*, vol. 8, no. 3, pp. 323–364, 1977.

[21] G. Agha, I. A. Mason, S. Smith, and C. Talcott, "Towards a theory of actor computation," in *Int. Conf. on Conc. Theory.* Springer, 1992, pp. 565–579.

[22] S. Paul, G. A. Agha, S. Patterson, and C. A. Varela, "Verification of eventual consensus in Synod using a failure-aware actor model," in *Proc. of the 13th NASA Form. Meth. Symp. (NFM).* Cham: Springer, 2021, pp. 249–267.

[23] L. Lamport, "The Part-Time Parliament," *ACM Trans. on Comp. Syst. (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[24] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[25] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Boston: Addison-Wesley Longman Publishing Co., Inc., 2002.

[26] R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi, *Reasoning About Knowledge.* MIT press, 2004.

[27] S. Paul, S. Patterson, and C. A. Varela, "Collaborative situational awareness for conflict-aware flight planning," in *The 39th IEEE/AIAA Dig. Avio. Syst. Conf. (DASC)*, 2020, pp. 1–10.

[28] L. Gönczy, M. Kovács, and D. Varró, "Modeling and verification of reliable messaging by graph transformation systems," *Elect. Notes in Theo. Comp. Sci.*, vol. 175, no. 4, pp. 37–50, 2007.

[29] C. C. Insaurralde and E. Blasch, "Framework prototype to test decision support system for avionics analytics," in *2021 IEEE/AIAA 40th Dig. Avio. Syst. Conf. (DASC).* IEEE, 2021, pp. 1–11.

[30] E. Blasch and B. Pokines, "Analytical science for autonomy evaluation," in *2019 IEEE Nat. Aero. and Elect. Conf. (NAECON).* IEEE, 2019, pp. 598–605.

[31] E. Blasch, F. Darema, S. Ravela, and A. Aved, Eds., *Handbook of Dynamic Data Driven Applications Systems.* Springer, 2021.

[32] E. Blasch, É. Bossé, and D. A. Lambert, *High-Level Information Fusion Management and Systems Design.* Artech House, 2012.

[33] E. Cruz-Camacho, S. Paul, F. Kopsaftopoulos, and C. A. Varela, "Towards provably correct probabilistic flight systems," in *Dyn. Data Driven App. Syst.*, F. Darema, E. Blasch, S. Ravela, and A. Aved, Eds. Cham: Springer International Publishing, 2020, pp. 236–244.

[34] A. Bhaumik, A. Dutta, F. Kopsaftopoulos, and C. A. Varela, "Proving the correctness of multicopter rotor fault detection and identification software," in *The 40th AIAA/IEEE Dig. Avio. Syst. Conf. (DASC)*, October 2021.

[35] F. Kopsaftopoulos and F.-K. Chang, "A dynamic data-driven stochastic state-awareness framework for the next generation of bio-inspired fly-by-feel aerospace vehicles," in *Handbook of Dynamic Data Driven Applications Systems.* Springer, 2018, pp. 697–721.

[36] F. Kopsaftopoulos, R. Nardari, Y. Li, and F. Chang, "Data-driven state awareness for fly-by-feel aerial vehicles: Experimental assessment of a non-parametric probabilistic stall detection approach," *Struct. Health Monit.*, pp. 1596–1604, 2017.

[37] S. Breese, F. Kopsaftopoulos, and C. Varela, "Towards proving runtime properties of data-driven systems using safety envelopes," in *The 12th Int. Work. on Struct. Health Monit.*, Stanford, CA, Sep. 2019.

[38] A. Dutta, M. E. McKay, F. Kopsaftopoulos, and F. Gandhi, "Fault detection and identification for multirotor aircraft by data-driven and statistical learning methods," in *2019 AIAA/IEEE Elect. Air. Tech. Symp. (EATS).* IEEE, 2019, pp. 1–18.

[39] A. Dutta, M. McKay, F. Kopsaftopoulos, and F. Gandhi, "Statistical residual-based time series methods for multicopter fault detection and identification," *Aero. Sci. Tech.*, vol. 112, p. 106649, 2021.

[40] S. Paul, S. Patterson, and C. A. Varela, "Formal guarantees of timely progress for distributed knowledge propagation," in *The Third Work. on Form. Meth. for Auto. Syst.*, ser. Electronic Proc. in Theoretical Computer Science, vol. 348. The Hague, Netherlands: Open Publishing Association, 2021, pp. 73–91.

[41] Y. Wang and Y. J. Zhao, "Fundamental issues in systematic design of airborne networks for aviation," in *IEEE Aero. Conf.* IEEE, 2006, pp. 8–pp.

[42] L. Lipsky, "M/M/1 Queue," *Queueing Theory: A Linear Algebraic Approach*, pp. 33–75, 2009.

[43] A. Al Hanbali, A. A. Kherani, and P. Nain, "Simple models for the performance evaluation of a class of two-hop relay protocols," in *Int. Con. on Res. in Net.* Springer, 2007, pp. 191–202.

[44] M. M. Hamdi, L. Audah, S. A. Rashid, A. H. Mohammed, S. Alani, and A. S. Mustafa, "A review of applications, characteristics and challenges in vehicular ad-hoc networks (VANETs)," in *Int. Cong. on Human-Comp. Int., Opt. and Rob. App. (HORA).* IEEE, 2020, pp. 1–7.

[45] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data Networks.* Prentice-Hall International New Jersey, 1992, vol. 2.

[46] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, 2nd ed. Reading, MA: Addison-Wesley, 1994.

[47] S. Paul, F. Kopsaftopoulos, S. Patterson, and C. A. Varela, "Dynamic data-driven formal progress envelopes for distributed algorithms," in *Dyn. Data-Driven App. Syst.*, 2020, pp. 245–252.

[48] S. Imai, A. Galli, and C. A. Varela, "Dynamic data-driven avionics systems: Inferring failure modes from data streams," in *Dyn. Data-Driven Appl. Syst.*, Reykjavik, Iceland, Jun. 2015, pp. 1665–1674.

[49] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Int. Conf. on Tools and Algo. for the Const. and Anal. of Syst.* Springer, 2008, pp. 337–340.